

Computer Graphics

Introduction

Computer graphics is a field related to the **generation of graphics using computers**. It includes the **creation, storage** and **manipulation** of images of objects. These objects come from diverse fields such as physical, mathematical engineering , architectural abstract structures and natural phenomenon. Computer graphics today is largely interactive; that is largely interactive; that is the user controls the contents structure and appearance of images of the objects by using input devices such as a keyboard, mouse, or touch sensitive panel on the screen

Until the early 1980's computer graphics was a small specialized field, largely because the hardware was expensive and graphics based application programs that were easy to use and cost effective were few. Then personal computers with built in raster graphics displays such as the Xerox Star Apple Macintosh and the IBM PC – popularized the use of bitmap graphics for user computer interaction. A bitmap is a ones and zeros representation of the rectangular array of points on the screen. Each point is called a pixel, short for “picture elements”. Once the bitmap graphics became affordable an explosion of easy to use and inexpensive graphics based user interfaces allowed millions of new users to control simple low cost application programs such as word processors, spreadsheets and drawing programs

The concept of a “*desktop*” now became popular metaphor for organizing screen space. By means of a window manager the user could create position and resize rectangular screen areas called windows. This allowed user to switch among multiple activities just by pointing and clicking at the desired window , typically with a mouse. Besides windows, icons which represent data files , application program, file cabinets, mailboxes. Printers , recycle bin and so on, made the user computer interaction more effective . by pointing and clicking the icons, users could activate the corresponding programs or objects which replaced much of the typing of the commands used in earlier operating systems and computer applications . today almost all interactive application programs even those for manipulating text i.e. word processor) or numerical data (e.g. spreadsheet programs) use graphics extensively in the user interface and for visualizing and manipulating the application specific objects.

Even people who do not use computers encounter computer graphics in TV commercials and as cinematic special effects. Thus computer graphics is an integral part of all computer user interfaces, and is indispensable for visualizing 2D, 3D objects in all most all areas such as education , science ,

engineering, medicine, commerce, the military, advertising and entertainment. The theme is that learning how to program and use computers now includes learning how to use simple 2D graphics

Early history of computer graphics

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

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

At that time only the most technology intensive organizations could use the interactive computer graphics where as others used punch cards, a non-interactive system .

Among the reasons for this were these:

- The high **cost of graphics hardware** – at a time when automobiles cost a few thousand dollars, computers cost several millions of dollars, and the first commercial computer displays cost more than a hundred thousand dollars
- The need for large scale **expensive computing resources** to support massive design database
- The difficulty of writing large interactive programs using **batch oriented FORTRAN** programming
- One of a kind , **non-portable software**, typically written for a particular manufacturer's display devices. When software is non-portable, moving to new display devices necessitates expensive and time consuming rewriting of working programs

Thus interactive computer graphics had a limited use when it started in the early sixties but it became very common once the Apple Macintosh and IBM PC appeared in the market with affordable cost.

Representative uses of computer graphics

Computer graphics is used today in many different areas of science, engineering , industry business, education, entertainment, medicine art and training

All of these are included in the following categories

1. User interfaces

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

2. Plotting

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

3. Office automation and electronic publishing

computer graphics has facilitated the office automation and electronic publishing which is also popularly known as desktop publishing, giving more power to the organizations to print the meaningful materials in house. Office automation and electronic publishing can produce both traditional printed (hardcopy) documents and electronic (softcopy) documents that contain text tables, graphs and other forms of drawn or scanned in graphics

4. Computer aided drafting and design

one of the major uses of computer graphics is to design components and systems of mechanical , electrical , electrochemical and electronic devices , including structures such as buildings automobile bodies, airplane and ship hulls, very large scale integrated (VLSI) chips optical systems

and telephone and computer networks . These designs are more frequently used to test structural , electrical and thermal properties of the systems

5. Scientific and business visualization

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

6. Simulation

Simulation is the imitation of the conditions like those, which is encountered in real life. Simulation thus helps to learn or to feel the conditions one might have to face in near future with out being in danger at the beginning of the course. For example, astronauts can exercise the feeling of weightlessness in a simulator, similarly a pilot training can be conducted in a flight simulator . The military tank simulator the naval simulator, driving simulator , air traffic control simulator, heavy duty vehicle simulator and so on are some of the mostly used simulator in practice. Simulators are also used to optimize the system

For example the vehicle, observing the reactions of the driver during the operation of the simulator

7. Entertainment

Disney movies such as Lion King and the beauty and the beast, and other scientific movies like star trek are the best examples of the application of computer graphics in the field of entertainment. Instead of drawing all the necessary frames with slightly changing scenes for the production of cartoon film only the key frames are sufficient for such cartoon film where the in between frames are interpolated by the graphics system dramatically decreasing the cost production while maintaining the quality. Computer and video games such as Fifa, Formula-1, Doom and Pools are few to name where computer graphics is used extensively

8. Art and commerce

Here computer graphics is used to produce pictures that expresses a message and attract attention such as a new model of a car moving along the ring of the Saturn. These pictures are frequently seen at transportation terminals, supermarkets, hotels etc. the slide production for commercial ,

scientific, or educational presentations is another cost effective use of computer graphics. One of such graphics packages is “PowerPoint”.

9. Cartography

Cartography is a subject which deals with the making of maps and charts. Computer graphics is used to produce both accurate and schematic representations of geographical and other natural phenomena from measurement data. Examples include geographic maps, oceanographic charts, weather maps, contour maps and population density maps

Surfer is one of such graphics packages which is extensively used for cartography

Besides these the field of Computer Graphics has been able to find its usage in diversified fields like medical field, tourism field etc where computers are used for storing and processing data. It is also widely used for educational purpose and for creating public and social awareness and for activities related to nation development this is **because information portrayed visually is more expressive and precise.**

A picture speaks thousands of words,

Major components (HW/SW) for computer graphics

Hardware Components:
monitors, mouse and joy-sticker, and hard-copy devices, e.g. high-resolution color printer

For software Components:
Special purpose utilities (device-dependent and device independent)
needed for handling processing

Input Hardware

Tablet

Tablet a tablet is a **digitizer**.

- scan over an object and input a set of discrete coordinate positions.

These positions can then be joined with straight line segments to approximate the shape of the original object.

A tablet digitizes an object detecting the position of a movable stylus (a pencil shaped device) or a puck(a mouse like device with cross hairs for sighting positions) held in the user's hand

Input Hardware



Input Hardware

i. Electrical Tablet

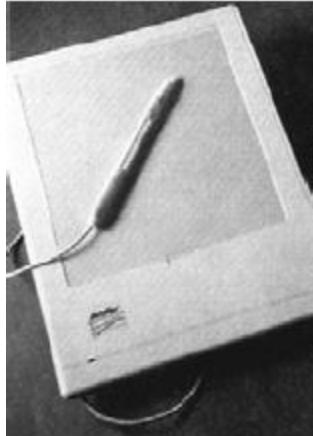
A grid of wires is embedded in the tablet surface

Electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induce an electrical signal in a wire coil in the stylus or puck

The strength of the signal induced by each pulse is used to determine the position of the stylus.

The signal strength is also used to determine roughly how far the stylus is from the tablet

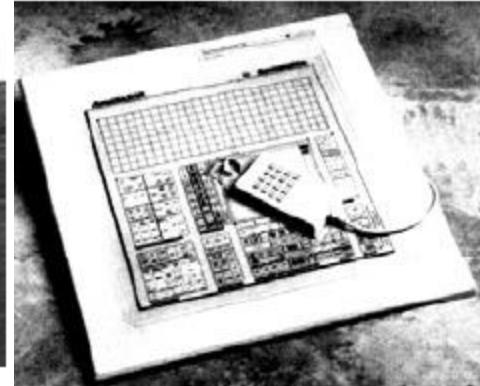
Input Devices: Tablets



desktop tablet
with stylus



artist's digitizer system
with cordless stylus



desktop tablet with a
16-button hand cursor



large tablet with
hand cursor

Input Devices: Tablets

Input Hardware

ii. Sonic Tablet

The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area

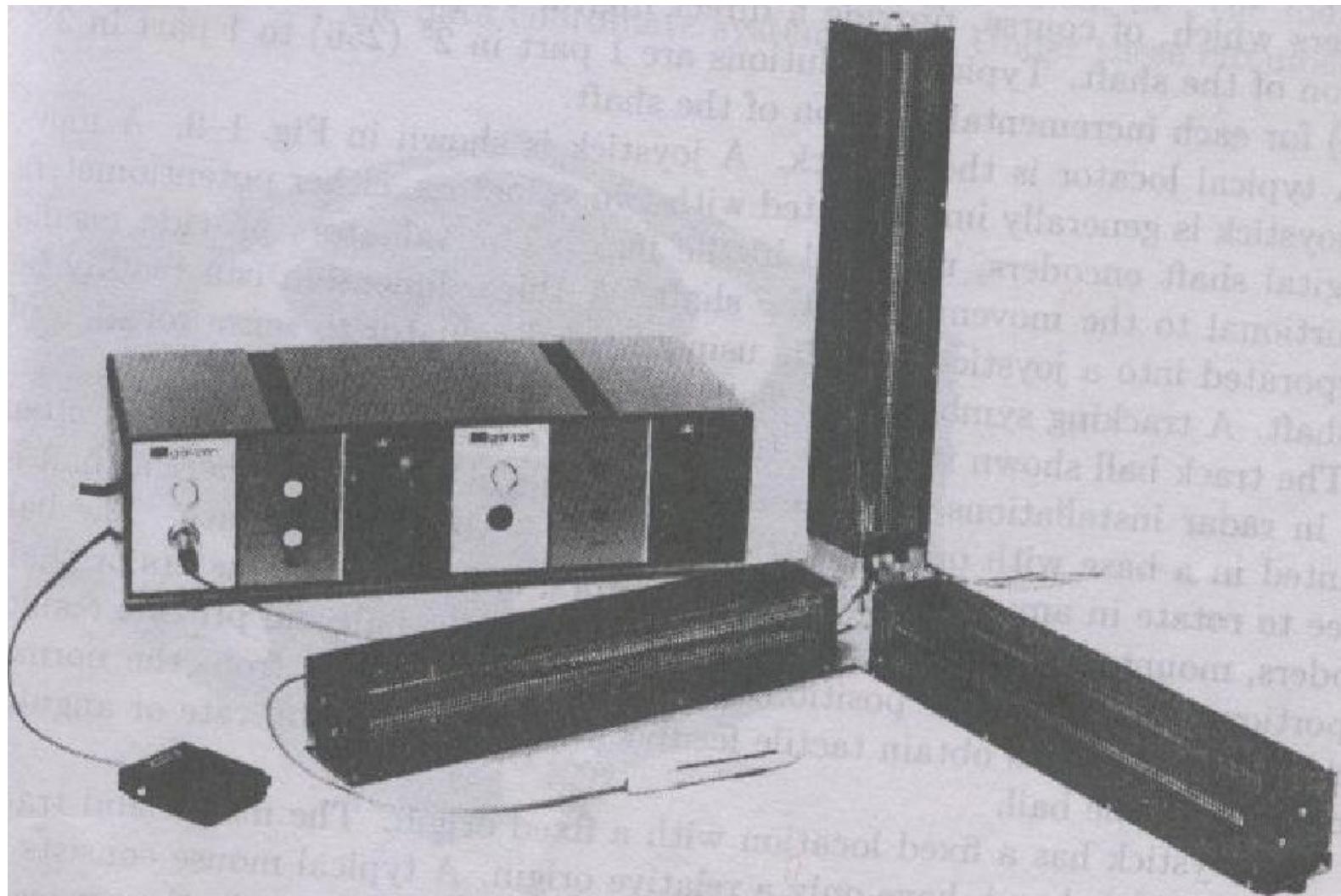
An electrical spark at the tip of the stylus creates **sound bursts**.

The position of the stylus or the coordinate values is calculated using the **delay between** when the spark occurs and when its sound arrives at each microphone

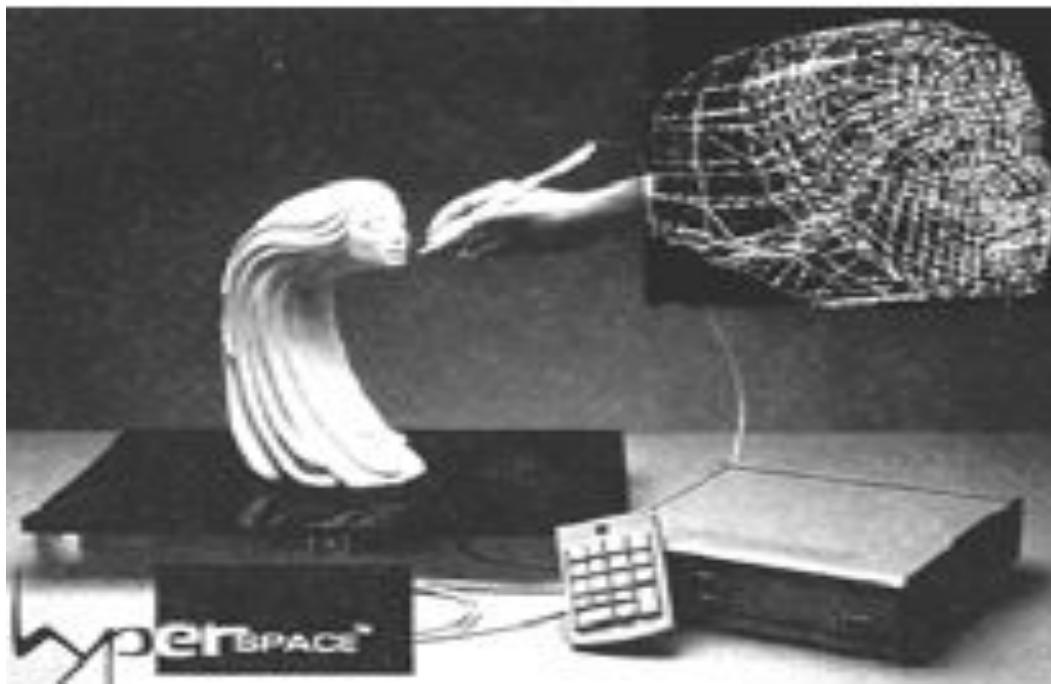
Input Hardware

The main advantage of sonic tablet is that it **doesn't require a dedicated working area** as the microphones can be placed on any surface to form the tablet work area

This facilitates digitizing drawing on thick books because in an electrical tablet this is not convenient for the stylus can not get closer to the tablet surface



3D Digitizers



manual digitizer
with stylus

3D Digitizers



MicroScribe G2
Desktop 3D Digitizer

Input Hardware

iii. Resistive Tablet

The tablet is just a piece of glass coated with a thin layer of conducting material

When a battery powered stylus is activated at certain position it emits high frequency radio signals which induces the radio signals on conducting layer.

The strength of the signal received at the edges of the tablet is used to calculate the position of the stylus

Input Hardware

Several types of tablets are transparent, and thus can be backlit for digitizing x-ray films and photographic negatives.

The Resistive tablet can be used to digitize the objects on CRT because it can be curved to the shape of the CRT.

The mechanism used in the electrical or sonic tablets can also be used to digitize the 3D objects

Input Hardware

Touch Panels

The touch panel allows the user to point at the screen directly with a finger to move the cursor around the screen or to select the icons.

i. Optical Touch Panel

It uses a series of infrared **light emitting diodes (LED)** along one vertical edge and along one horizontal edge of the panel

The opposite vertical and horizontal edges contain **photo detectors** to form a grid of invisible infrared light beams over the display area.

Input Hardware

Touching the screen breaks one or two vertical and horizontal light beams thereby indicating the fingers position

The cursor is then moved to this position or the icon at this position is selected

This is a low resolution panel which offers 10 to 50 positions in each direction

Input Devices: Touch Screens



plasma panels with touch screens

Input Hardware

ii. Sonic Touch Panel

Bursts of high frequency sound waves traveling alternately horizontally and vertically are generated at the edge of the panel .

Touching the screen causes part of each wave to be reflected back to its source

The screen position at the point of contact is then calculated using the time elapsed between when the wave is emitted and when it arrives back at the source

This is a high resolution touch panel having about 500 positions in each direction

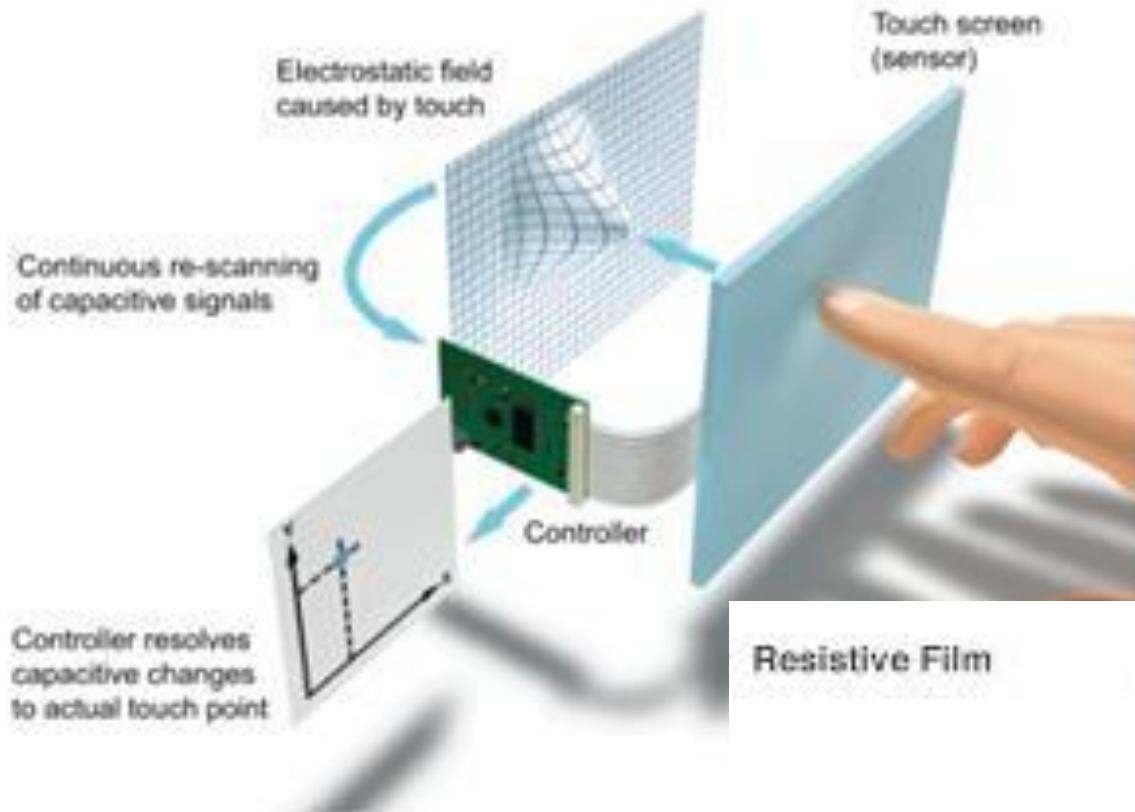
Input Hardware

iii. Electrical Touch Panel

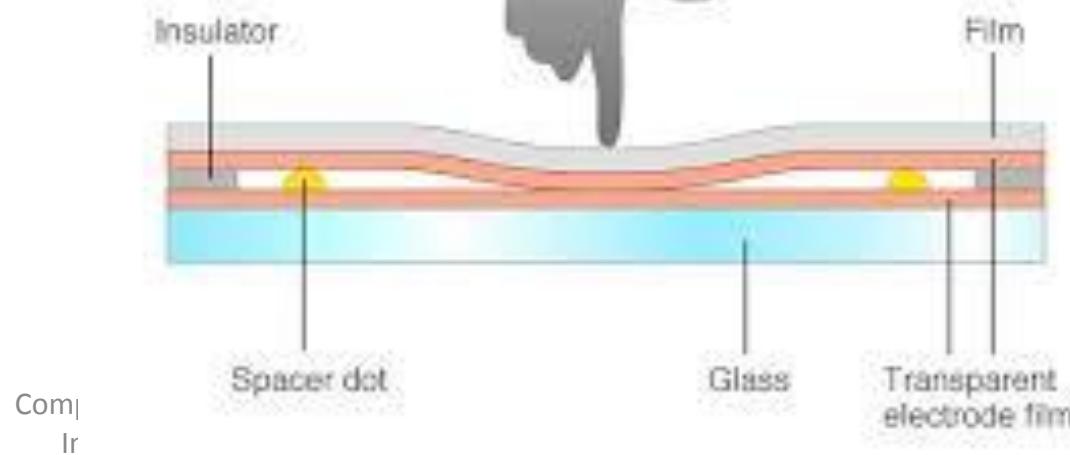
It consists of slightly separated two transparent panel one coated with a thin layer of conducting material and the other with resistive material

When the panel is touched with a finger the two plates are forced to touch at the point of contact thereby creating the voltage drop across the resistive plate which is then used to calculate the coordinate of the touched position

The resolution of the touch panel is similar to that of sonic touch panel



Resistive Film



Input Hardware

Light pen

It is a pencil shaped device to determine the coordinates of a point on the screen where it is activated such as pressing the button .

In raster display 'y' is set at y_{max} and 'x' changes from 0 to x_{max} the first scan line .

For the second line 'y' decreases by one and 'x' again changes from 0 to x_{max} and so on

When activated light pen sees a burst of light at certain position as the electron beam hits the phosphor coating at that position it generates an electric pulse

Input Hardware

This is used to save the video controller's 'x' and 'y' registers and interrupt the computer

By reading the saved valued the graphics package can determine the coordinates of the position seen by the light pen

Drawbacks

- i. Light pen **obscures screen images** as it is pointed to required spot
- ii. Prolong use of it can cause **arm fatigue**
- iii. It cannot report the coordinates of a point that is **completely black** as a remedy one can display a dark blue field in place of the regular image for a single frame time
- iv. It gives sometimes **false reading** due to back ground lighting in a room

Input Devices: Light Pen



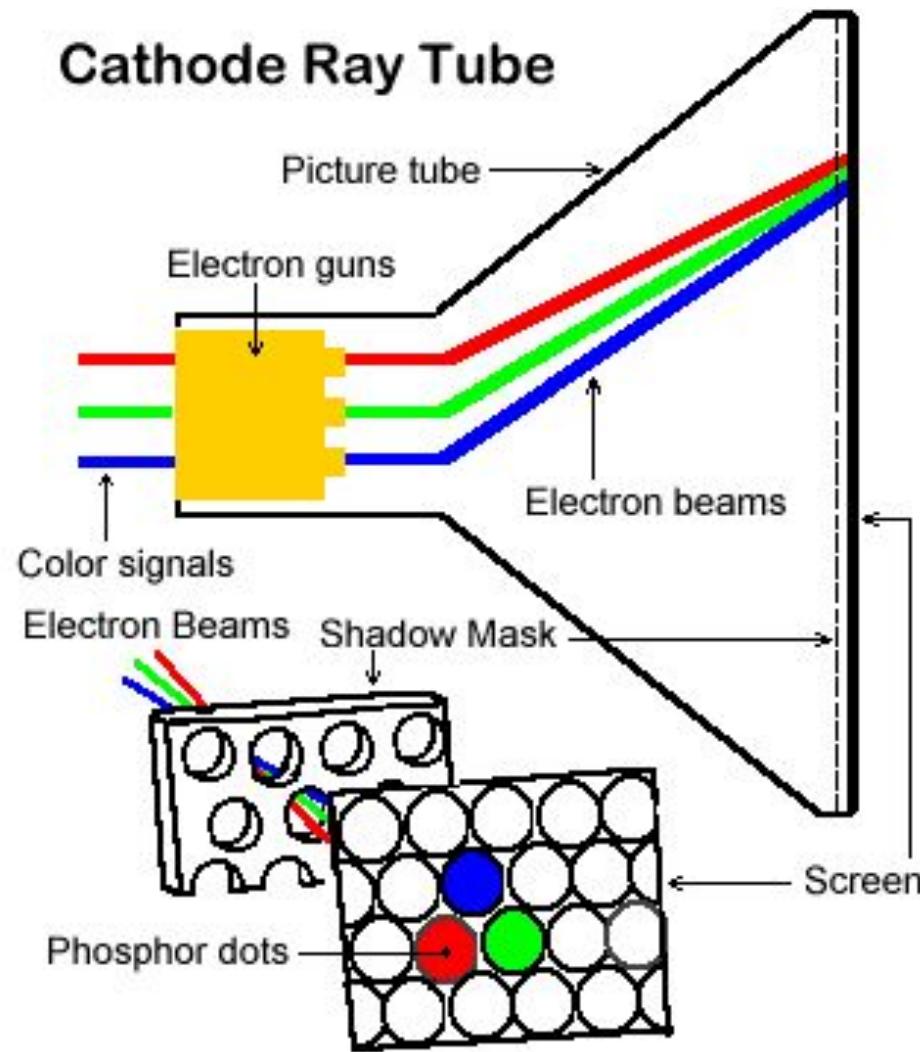
recognizes cathod ray
to calculate position



Types of Displays:

- Emissive Displays
 - The emissive display **converts electrical energy into light energy.**
 - The **image is produced directly on the screen** Phosphors convert electron beams or UV light into visible light
 - e.g.: Plasma, Cathode Ray Tube (CRT) Light-Emitting Diode Displays (LED), Plasma Display Panel (PDP)
- Non-Emissive Displays
 - **Light is produced behind the screen** and the image is formed by filtering this light e.g.: LC-Display (LCD)
 - The Non emissive display uses **optical effects to convert the sunlight or light from any other source to graphic form**

Cathode Ray Tube (CRT)

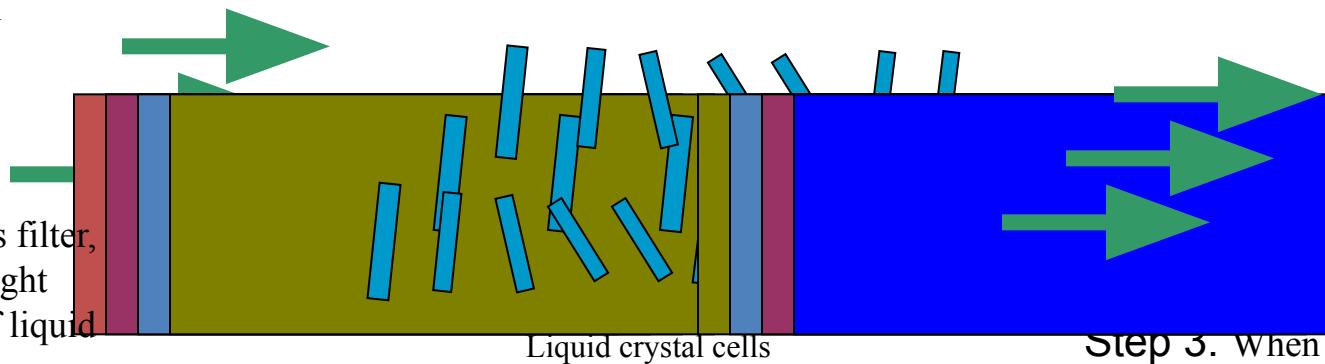


LCD (Liquid Crystal Display)

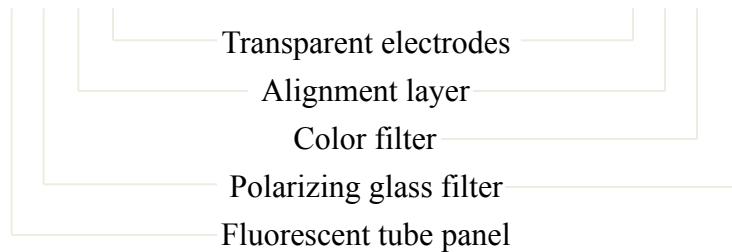
- LCD (Liquid Crystal Display)
 - A type of flat-panel display
 - Uses **liquid crystals** between two sheets of material to present information on a screen
 - An electric current passes through the liquid crystals, **they twist**
 - Depending on **how much they twist, some light waves are passed through** while other light waves are blocked. This creates the variety of color that appears on the screen

How LCD works?

Step 1. Panel of fluorescent tubes emits light waves through polarizing glass filter, which guides light toward layer of liquid crystal cells.



Step 2. As light passes through liquid crystal, electrical charge causes some of the cells to twist, making light waves bend as they pass through color filter.



Step 3. When light reaches second polarizing glass filter, light is allowed to pass through any cells that line up at the first polarizing glass filter. Absence and presence of colored light cause image to display on the screen.

LCD (Liquid Crystal Display)

- LCD monitors produce color using either passive-matrix or active-matrix technology
- **Active-matrix display**, also known as a TFT (thin-film transistor) display, uses a separate transistor to apply changes to each liquid crystal cell and thus display high-quality color that is viewable from all angles

LCD (Liquid Crystal Display)

- **Passive-matrix display** uses **fewer transistors** and requires less power than an active-matrix display
- The color on a passive-matrix display often is **not as bright** as an active-matrix display
- Users view images on a passive-matrix display best when working directly in front of it
- Passive-matrix displays are less expensive than active-matrix displays

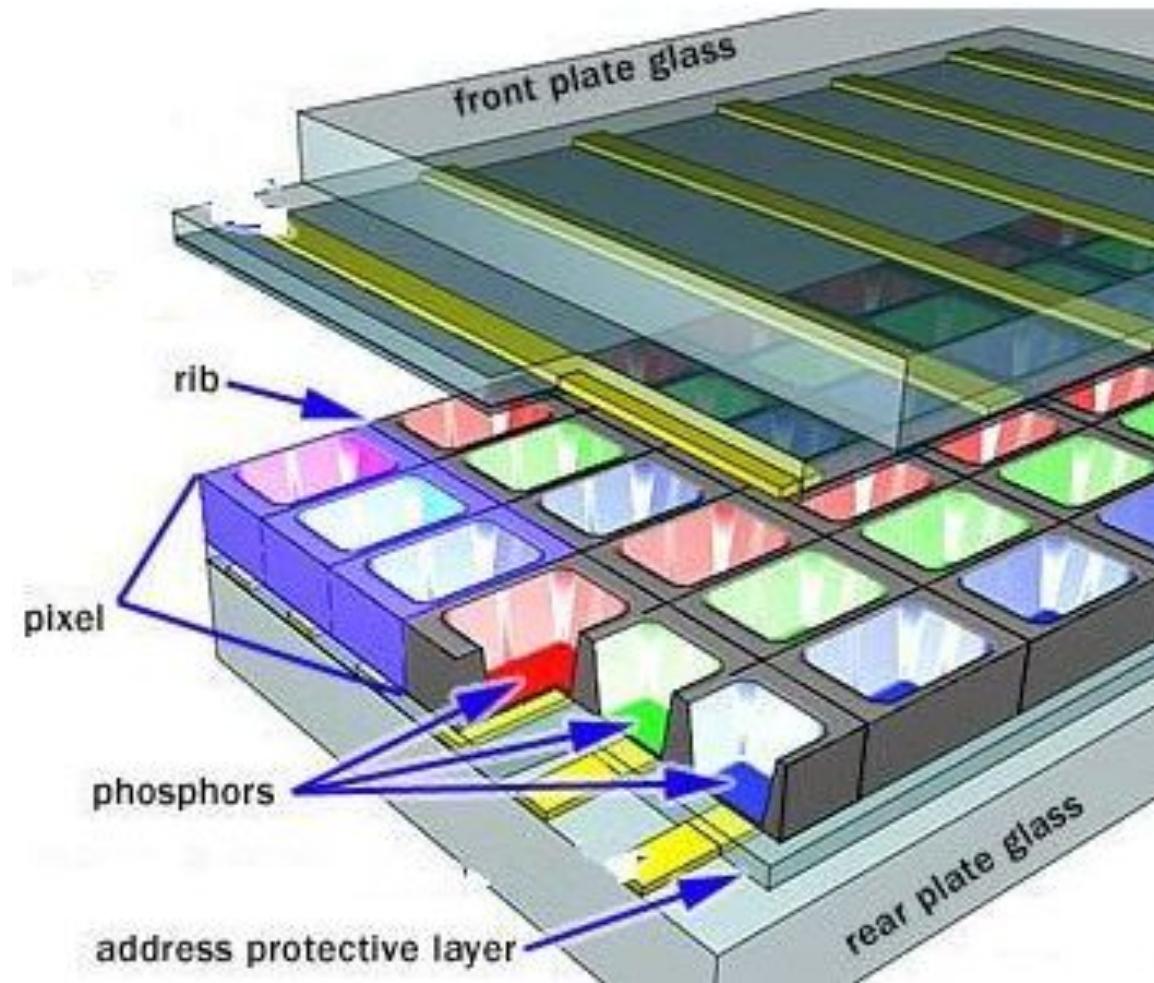
LCD (Liquid Crystal Display)

- An important measure of LCD monitors is the **response time**, which is the time in millisecond (ms) that it takes to turn a pixel on or off
- LCD monitors' response times average 25 ms
- The lower the number, the faster the response time
- **Resolution** and **dot pitch** determines quality of LCD monitor

Gas plasma monitor

- A flat-panel display that uses **gas plasma technology**
- A layer of **gas** between two sheets of material
- When voltage is applied, the **gas releases ultraviolet (UV) light** that causes the pixels on the screen to glow and form an image
- Larger screen sizes and higher display quality than LCD, but much more expensive

Gas plasma monitor



ADVANTAGES:

1. Refreshing is not required.
2. Produce a very steady image free of Flicker.
3. Less bulky than a CRT.

DISADVANTAGES:

1. Poor resolution of up to 60 d.p.i.
2. It requires complex addressing and wiring.
3. It is costlier than CRT.

Hard Copy Devices

Printers

Printed output is referred to as **hard copy** and do not require electric power as they are printed on papers to read after printing and provide permanent readable form information

According to how they print printers can be of different types:

Character printers prints one character of a text at a time

Line printer prints one line of the text at a time

A **page printer** prints one page of the text at a time

According to the technology used printers produce output by either **impact** or **non impact** methods

Impact printers

Impact printers **press the formed character faces** against an inked ribbon onto paper

Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed.

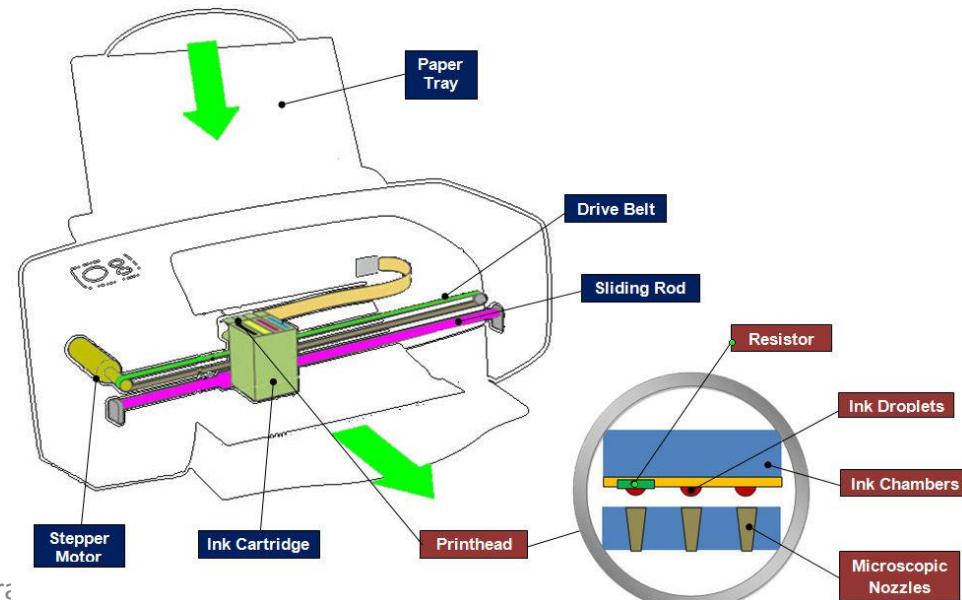
Non-Impact Printers

Non impact printers **use laser techniques, ink-jet sprays** etc to get images onto paper.

Ink-jet Devices

Ink-jet methods produce output by **squirting ink** in horizontal rows across a roll of paper wrapped on a drum.

When a heater is activated a drop of ink is exploded onto the paper

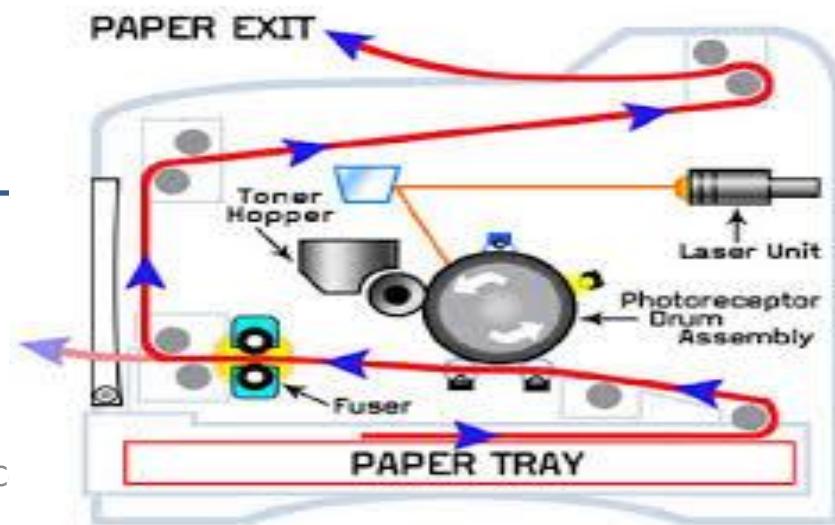


Laser Devices

These are **page printers**

They use **laser beam** to produce an image of the page containing text graphics on a photosensitive drum which is coated with negatively charged photo conductive material

In a laser device a laser beam creates a charge distribution on a rotating drum coated with a photo electric material such as selenium. Toner is applied to drum and then transferred to paper.



Potters

Plotter is a device that draws pictures on paper based on commands from a computer

They are used to produce precise and good quality graphics and drawing under computers control

They use motor driven ink pen or ink jet to draw graphic or drawings

Drawings can be prepared on paper, Vellum or Mylar (Polyester film)

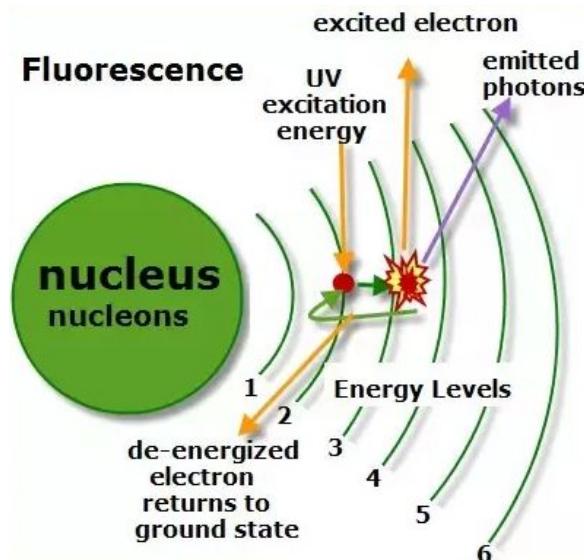


Display Devices

Fluorescence / Phosphorescence

A phosphors fluorescence is the light emitted as the very unstable electrons lose their excess energy whole the phosphor is being struck by electrons

Phosphorescence is the light given off by the return of the relatively more stable excited electrons to their unexcited state once the electron beam excitation is removed



Display Devices



Persistence

A phosphor's persistence is defined as the time from the removal of excitation to the moment when phosphorescence has decay to 10 percent of the initial light output

The range of persistence of different phosphors can reach many seconds

The phosphors used for graphics display devices usually have persistence of 10 to 60 micro seconds

A phosphor with low persistence is useful for animation and a high persistence phosphor is useful to highly complex static pictures

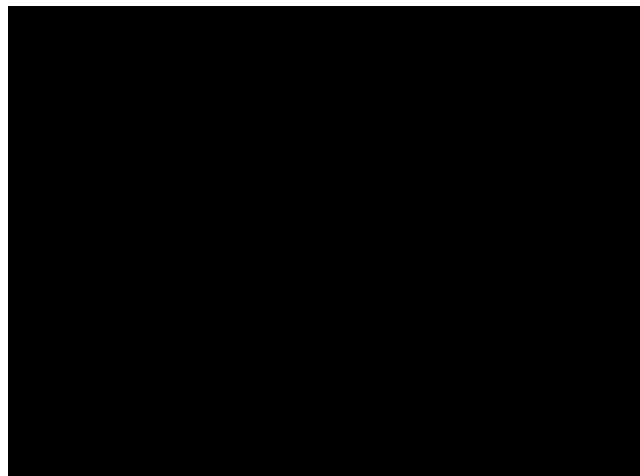
Display Devices

Refresh rate

The refresh rate is the number of times per second the image is redrawn to give a feeling of un-flickering pictures and it is usually 50 per second

As the refresh rate decreases flicker develops because the eye can no longer integrate the individual light impulses coming from a pixel

The refresh rate above which a picture stops flickering and fuses into a steady image is called the critical fusion frequency (CFF)



Display Devices

The factors affecting the CFF are:

- i. Persistence: longer the persistence the lower the CFF But the relation between the CFF and persistence is non linear
- ii. Image intensity: Increasing the image intensity increases the CFF with non linear relationship
- iii. Ambient room light Decreasing the ambient room light increases the CFF with nonlinear relationship
- iv. Wave lengths of emitted light
- v. Observer

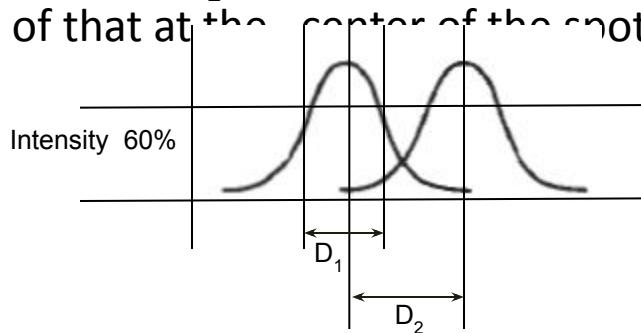
Display Devices

Resolution

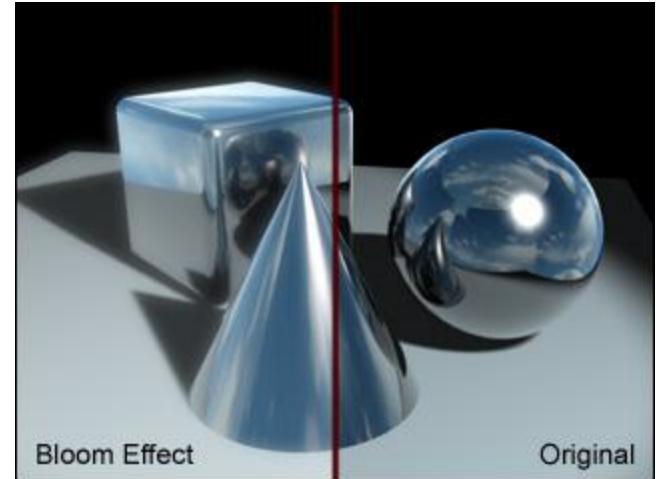
Resolution is defined as the maximum number of points that can be displayed horizontally and vertically without overlap on a display device. Factors affecting the resolution are as follows.

i. Spot profile:

The spot intensity has a Gaussian distribution. So two adjacent spots on the display device appear distinct as long as their separation D_2 is greater than the diameter of the spot D_1 at which each spot has an intensity of about 60 percent of that at the center of the spot.



Display Devices



ii. Intensity:

As the intensity of the electron beam increases the spot size on the display tends to increase because of spreading of energy beyond the point of bombardment

This phenomenon is called *blooming*. Consequently, the resolution decreases.

Thus it is noted that resolution is not necessarily a constant and it is not necessarily equal to the resolution of a pix-map, which is allocated in a buffer memory

Display Devices

Color CRTs

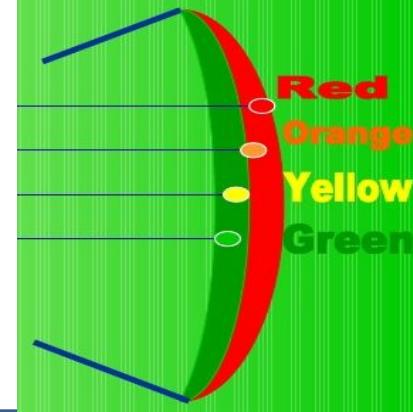
Color depends on the light emitted by phosphor.

Two type:

Beam Penetration Method

Shadow Mask Method

Display Devices



i. Beam Penetration Method:

Two different layers of phosphor coating used Red (outer) and Green (inner)

Display of color depends on the depth of penetration of the electron beam into the phosphor layers

- A beam of slow electrons excites only the outer red layer
- A beam of very fast electrons penetrates thru the red phosphor and excites the inner green layer
- When quantity of red is more than green then color appears as orange
- When quantity of green is more than red then color appears as yellow

Screen color is controlled by the beam acceleration voltage.

Only four colors possible, poor picture quality

Display Devices

ii. Shadow Mask Method

The inner side of the viewing surface of a color CRT consists of closely spaced groups of red, green and blue phosphor dots.

Each group is called a *triad*

A thin metal plate perforated with many small holes is mounted close to the inner side of the viewing surface. This plate is called *shadow mask*

The shadow mask is mounted in such a way that each hole is correctly aligned with a triad in color CRT

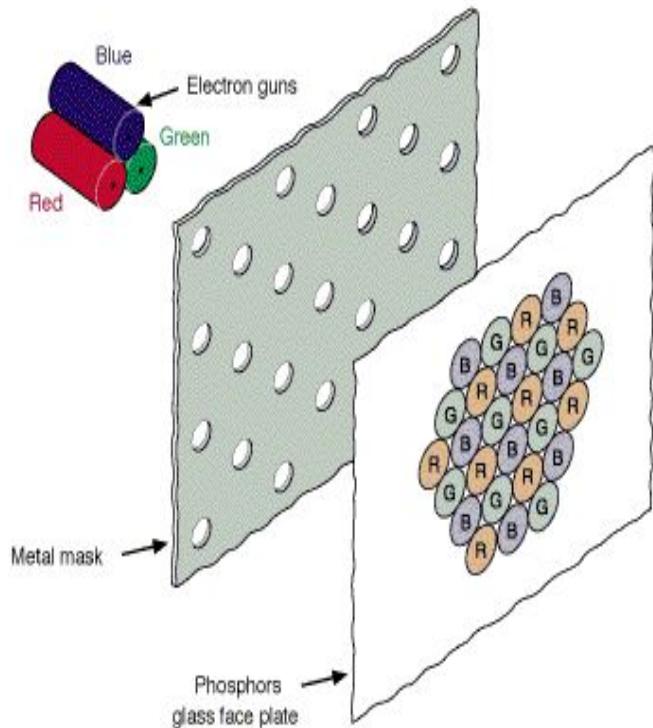
Display Devices

There are three electron guns one for each dot in a triad

The electron beam from each gun therefore hits only the corresponding dot of a triad as the three electron beams deflect

A triad is so small that light emanating from the individual dots is perceived by the viewer as a mixture of the three colors

Thus, a wide range of colors can be produced by each triad depending on how strongly each individual phosphor dot in a triad is excited.



Display Devices

a. A Delta –Delta CRT

A triad has a *triangular (delta) pattern* as are the three electron guns

Main drawback of this type of CRT is that a high precision display is very difficult to achieve because of technical difficulties involved in the alignment of shadow mask holes and the triad on one to one basis

b. Precision Inline CRT

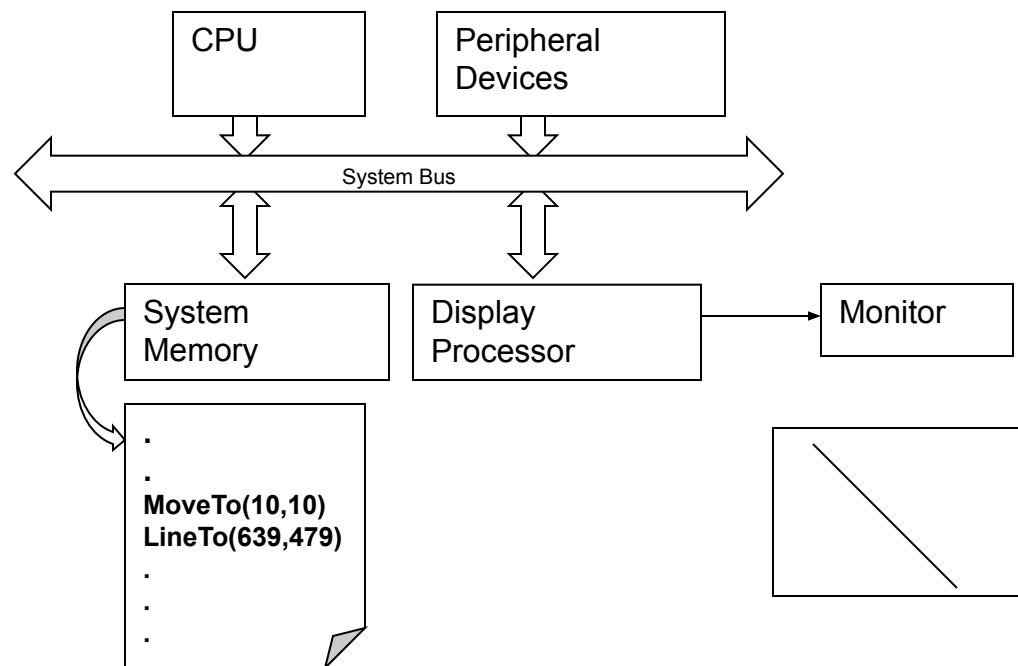
A triad has an *in-line pattern* as are the three electron guns

The introduction of this type of CRT has eliminated the main drawback of a Delta-Delta CRT

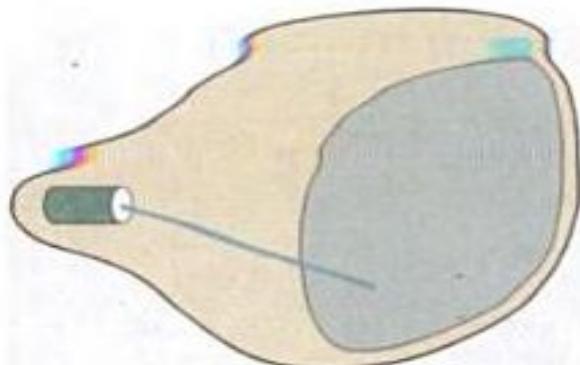
But a slight reduction of image sharpness at the edges of the tube has been noticed

Normally 1000 scan lines can be achieved

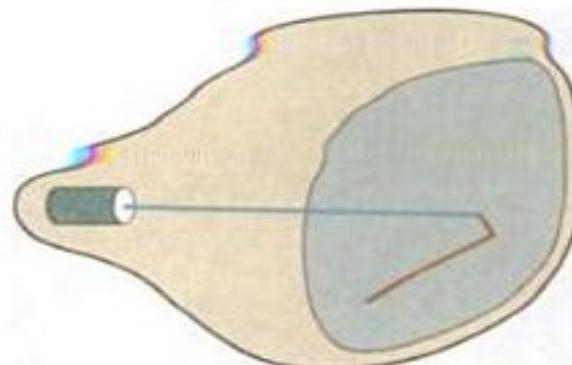
i. Vector Display Technology



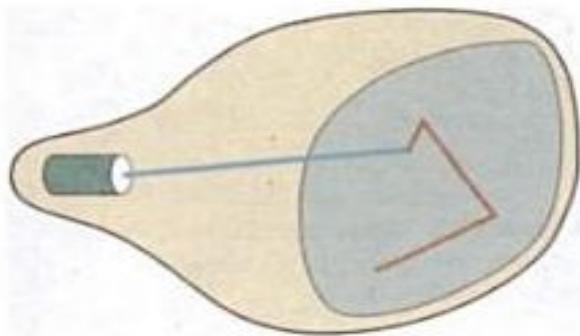
i. Vector Display Technology



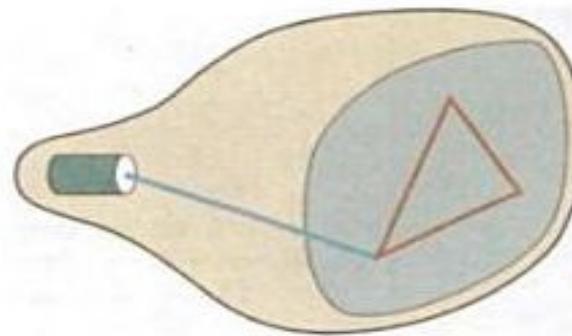
(a)



(b)



(c)



(d)

i. Vector Display Technology

It is also called **random scan, a stroke, a line drawing or calligraphic display**

Advantages:

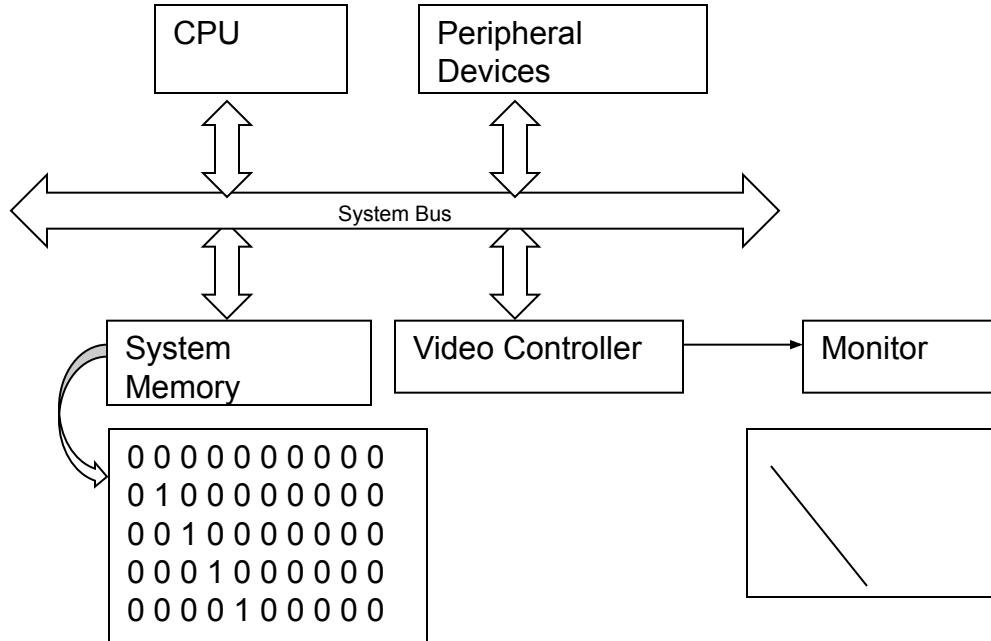
- i. It can produce a **smooth output primitives** with higher resolution unlike the raster display technology
- ii. It is better than raster display for real time dynamics such as **animation**
- iii. For transformation, only the end points has to be moved to the new position in vector display but in raster display it is necessary to move those end points and at the same time all the pixels between the end points must be scan converted using appropriate algorithm
No prior information on pixels can be reused

i. Vector Display Technology

Disadvantages:

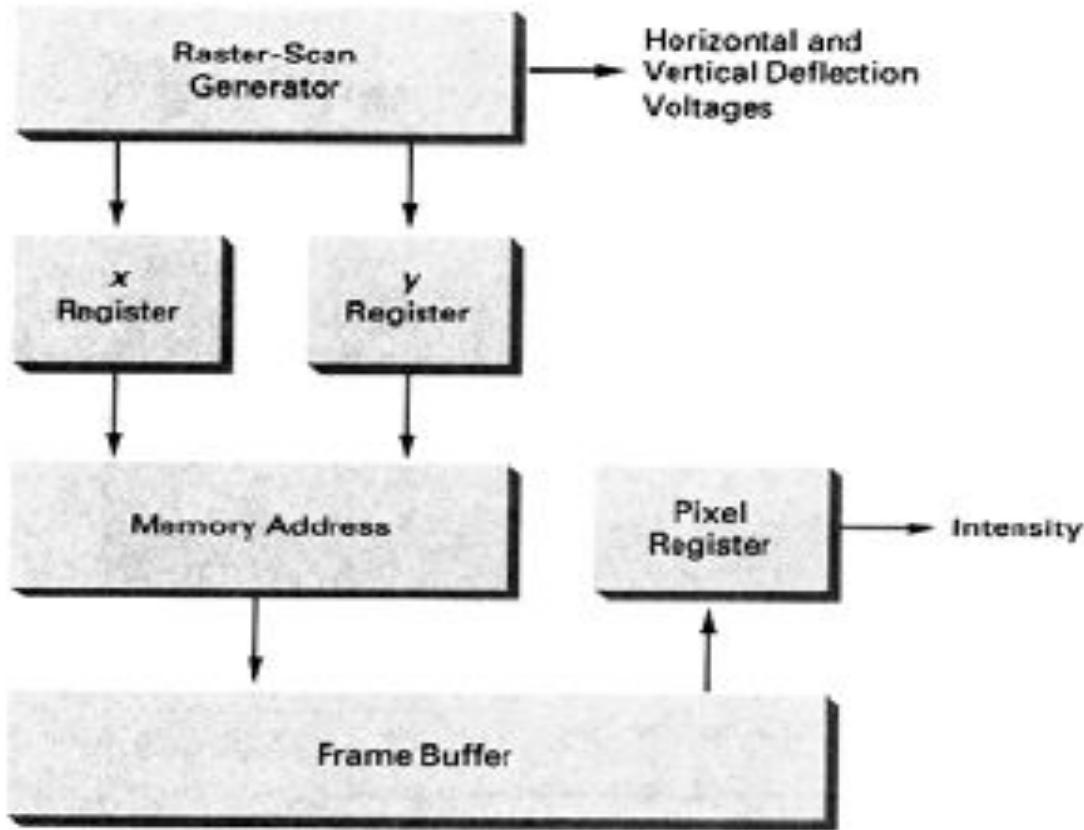
- i. A vector display **can not fill areas** with patterns and manipulate bits
- ii. Time required for refreshing an image depends upon its complexity (**more the lines, longer the time**) the flicker may therefore appear as the complexity of the image increases. The fastest vector display can draw about 100000 short vectors in a refresh cycle without flickering

ii. Raster Display Technology



ii. Raster Display Technology

Raster-Scan: Video Controller



basic video-controller refresh operations

ii. Raster Display Technology

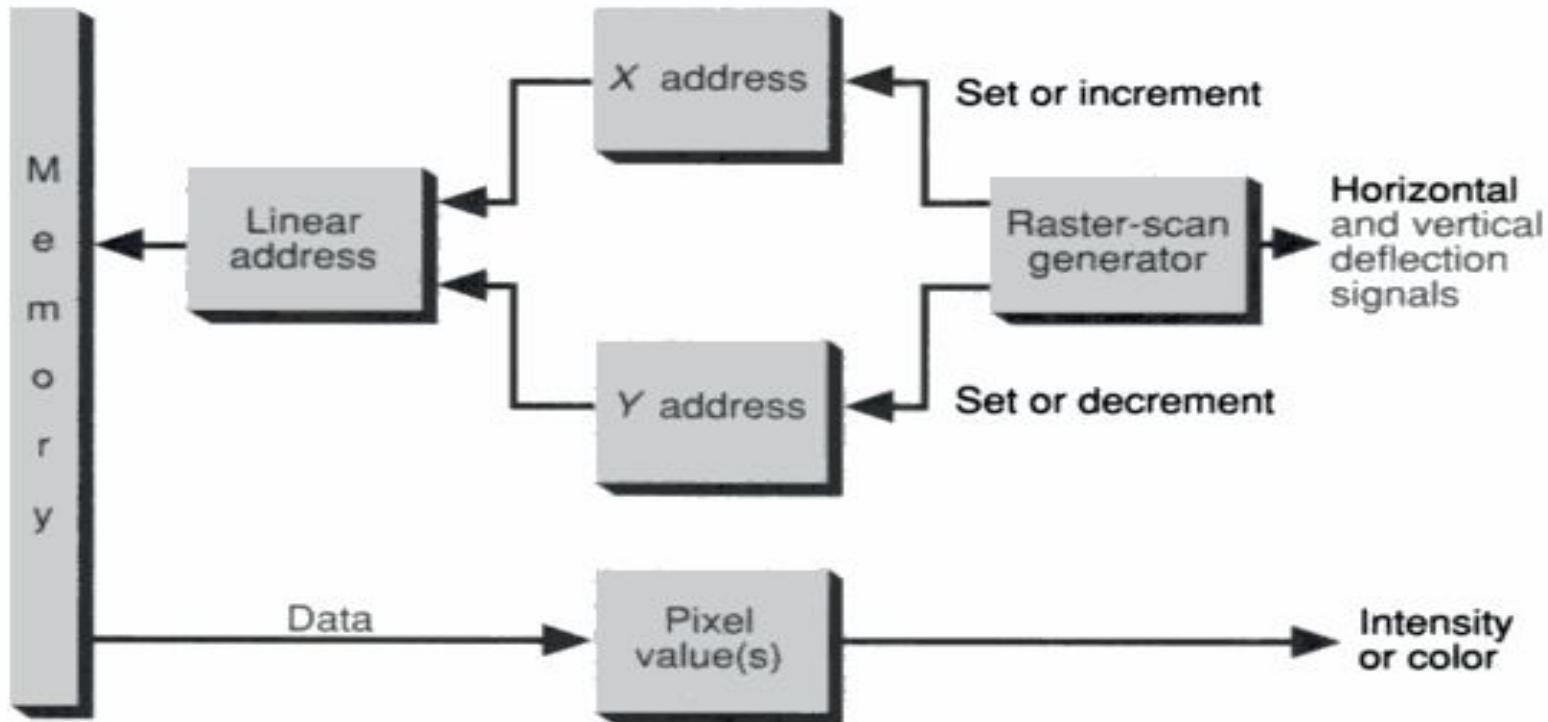


Fig. 4.20 Logical organization of the video controller.

ii. Raster Display Technology

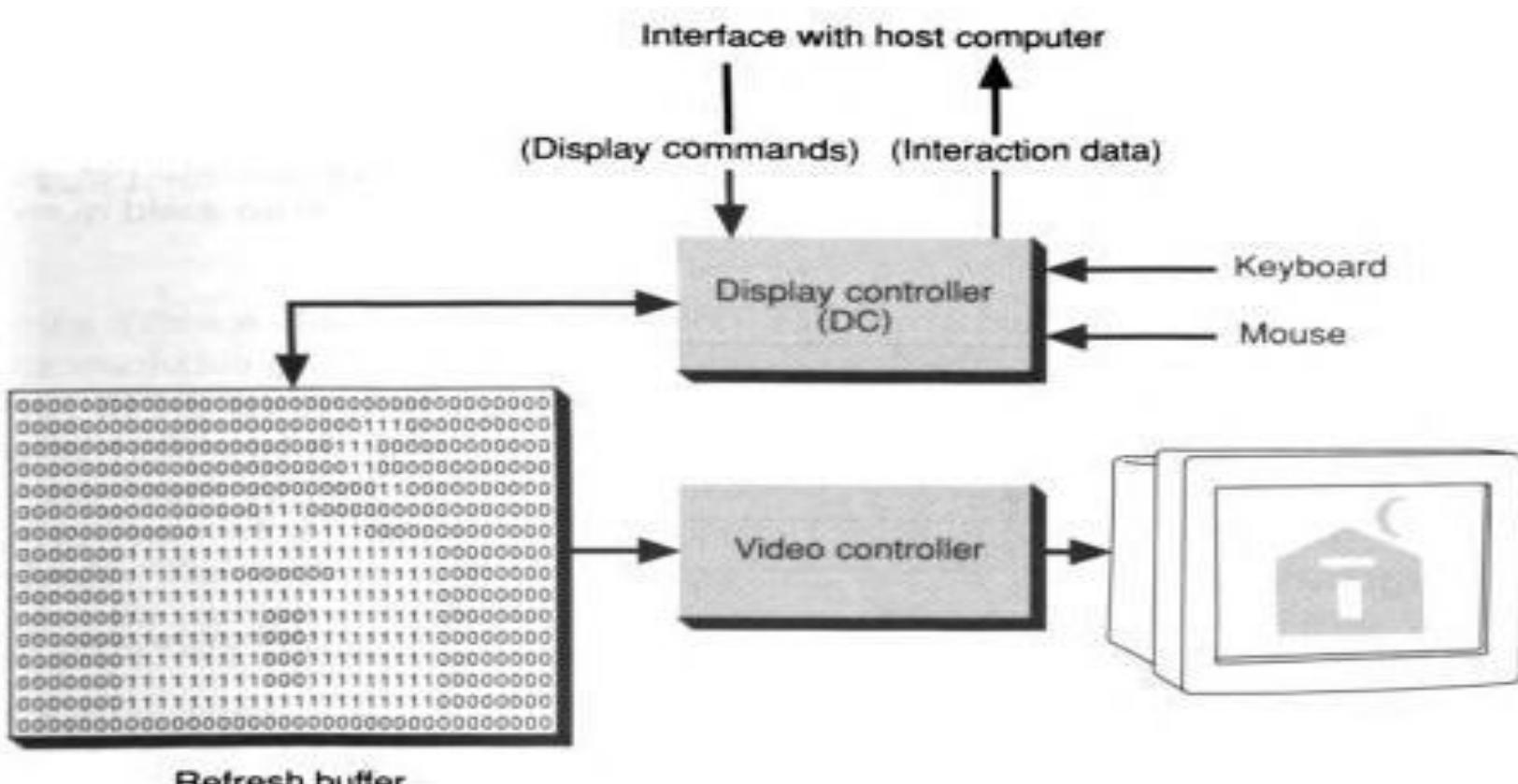
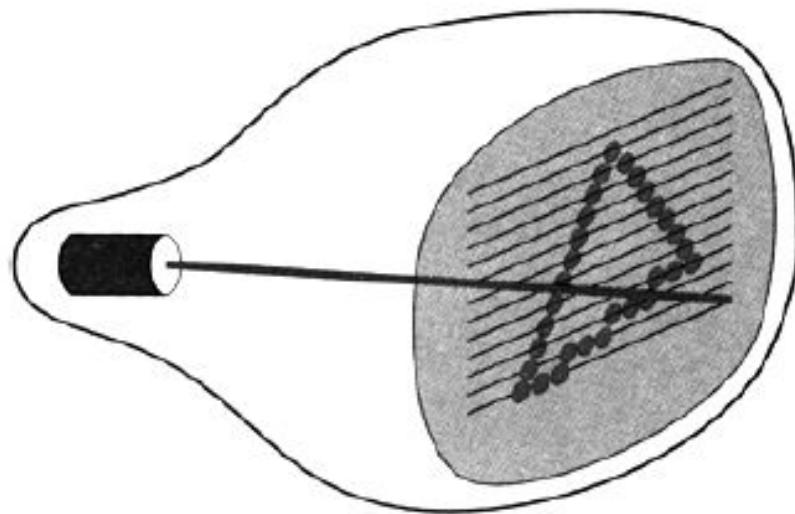


Fig. 1.2 Architecture of a raster display.

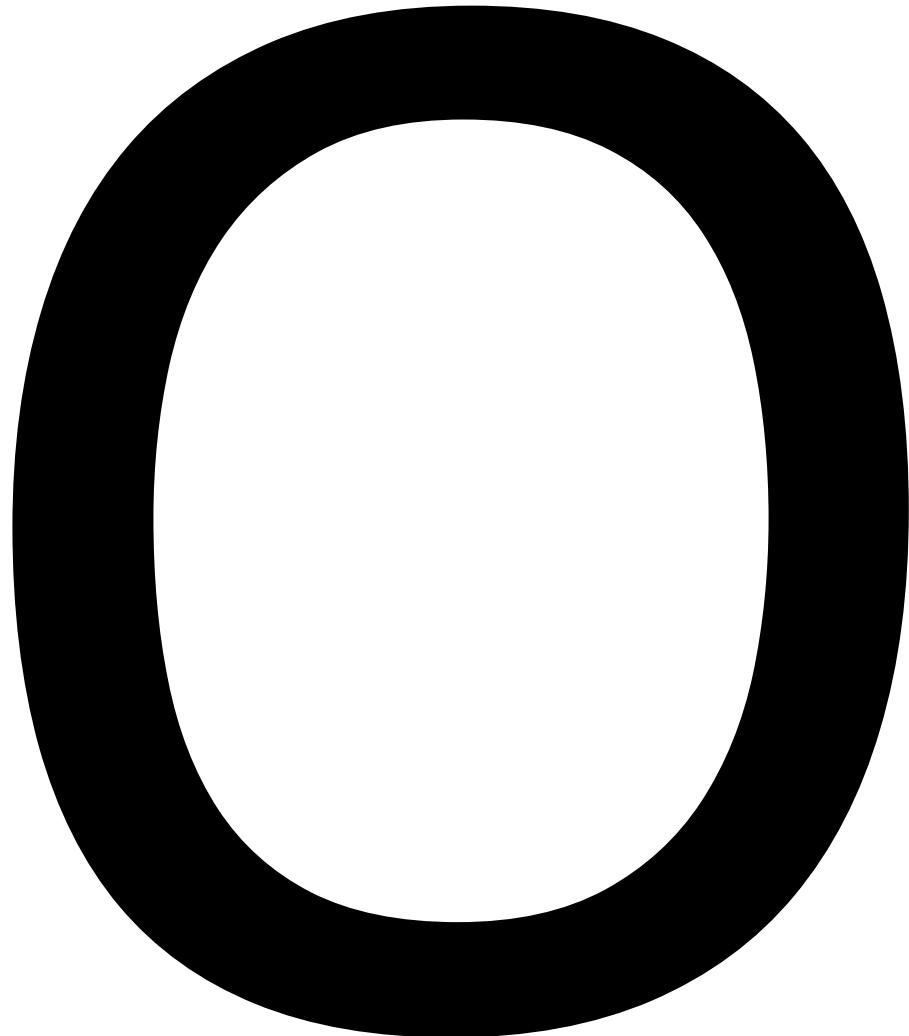
ii. Raster Display Technology



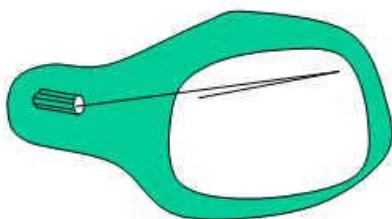
ii. Raster Display Technology

Disadvantages

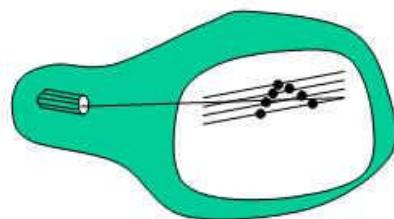
- i. For Real-Time dynamics not only the end points are required to move but all the pixels in between the moved end points have to be scan converted with appropriate algorithms which might slow down the dynamic process
- ii. Due to scan conversion “**jaggies**” or “**stair-casing**” are unavoidable



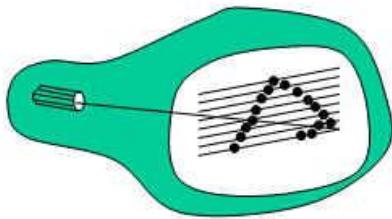
RDT/VDT



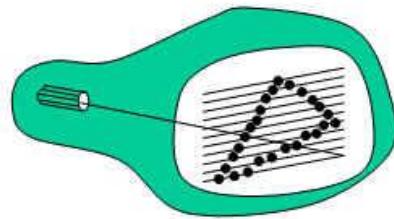
(a)



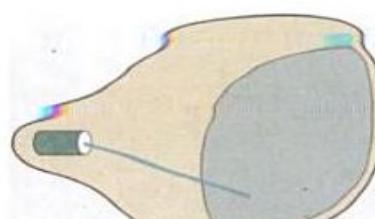
(b)



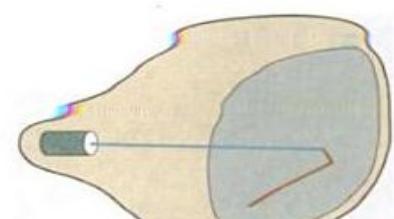
(c)



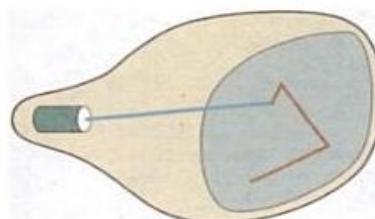
(d)



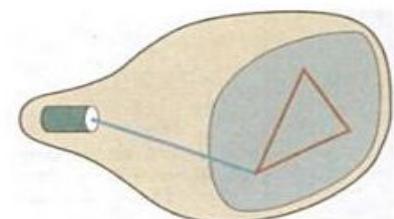
(a)



(b)

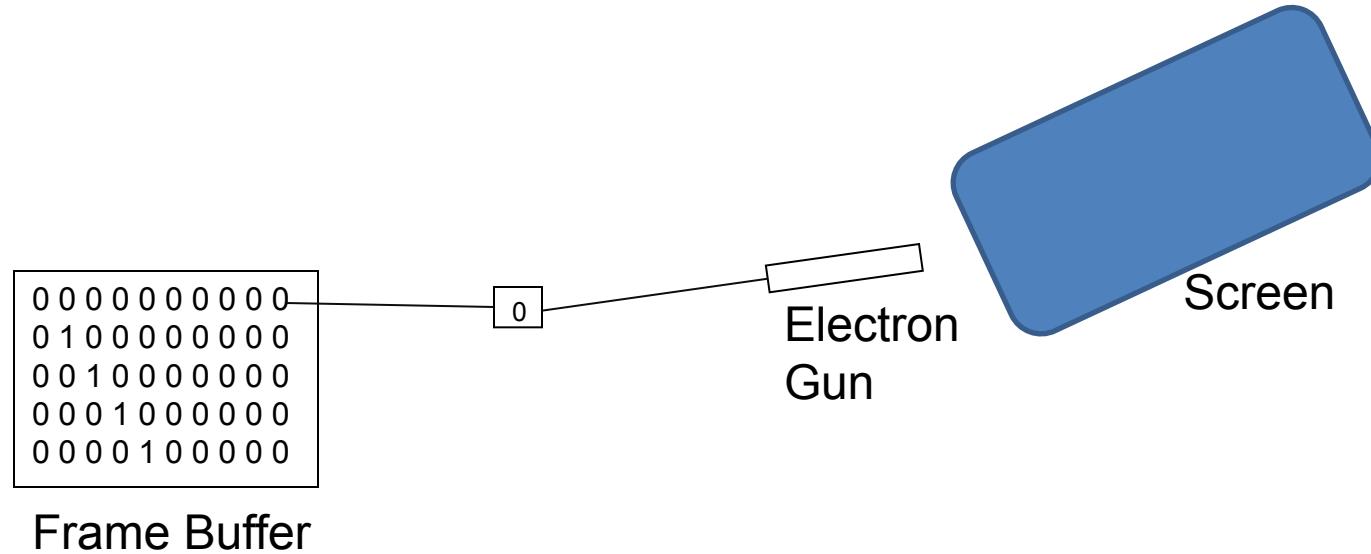


(c)

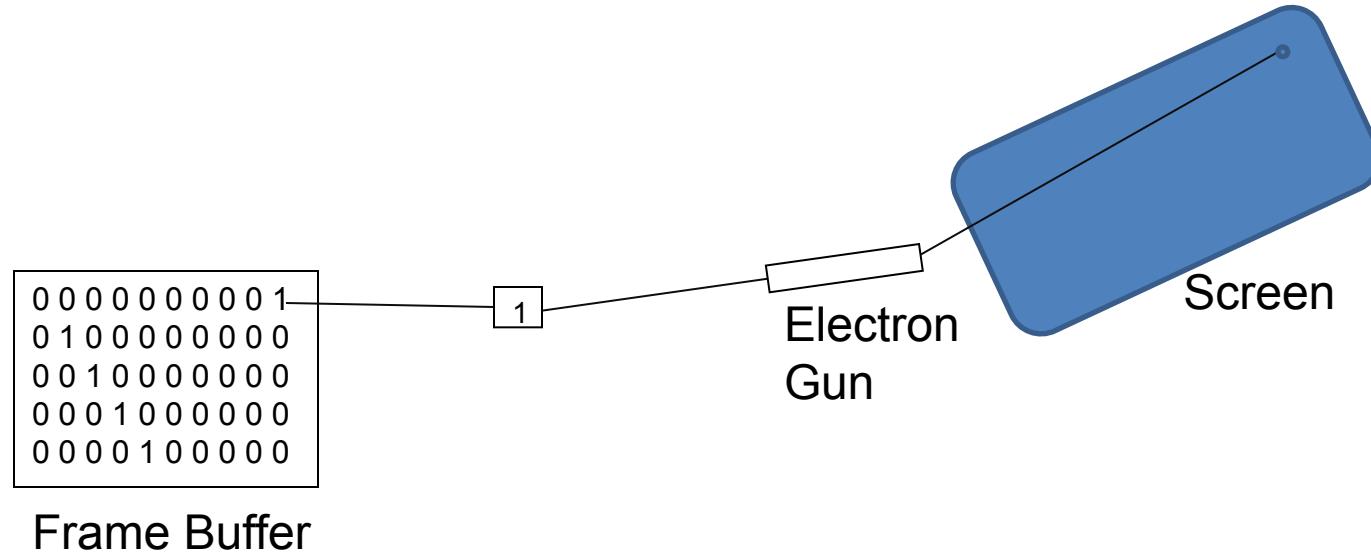


(d)

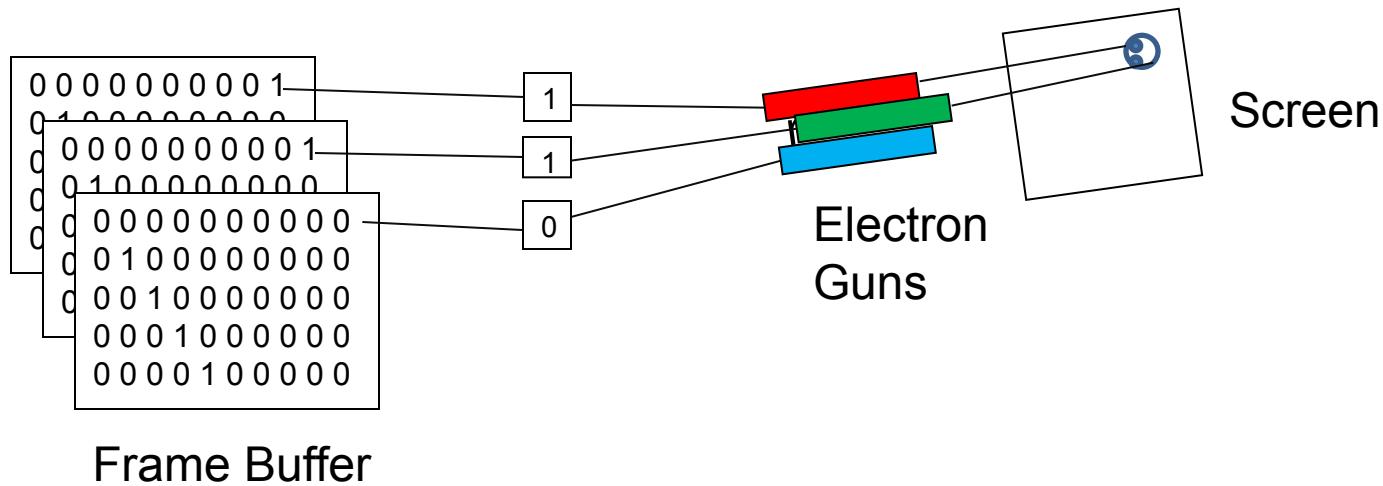
Frame Buffer



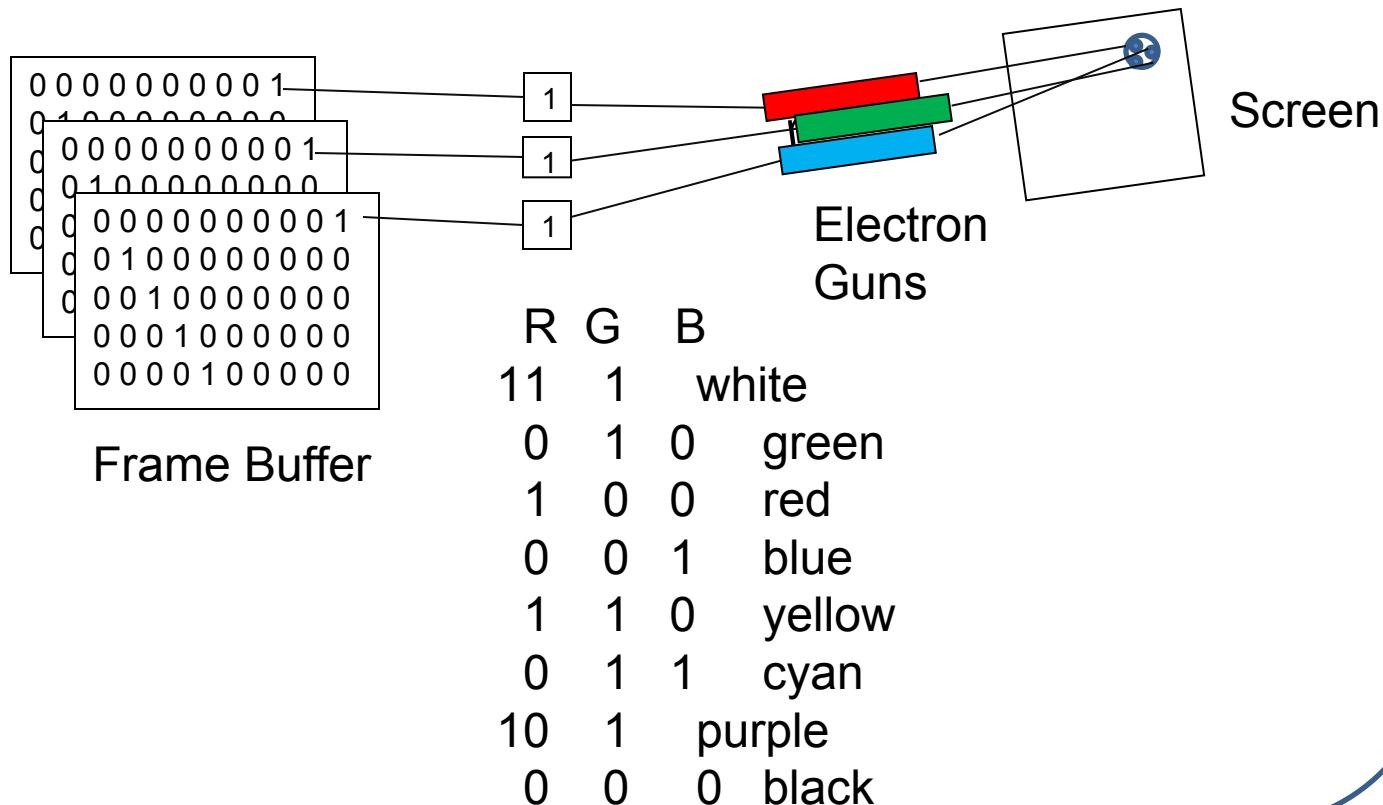
Frame Buffer



Frame Buffer



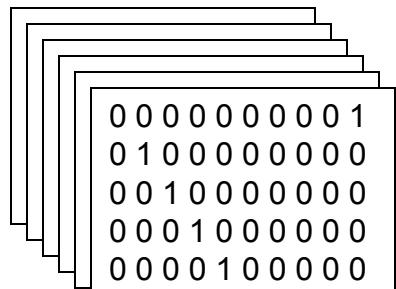
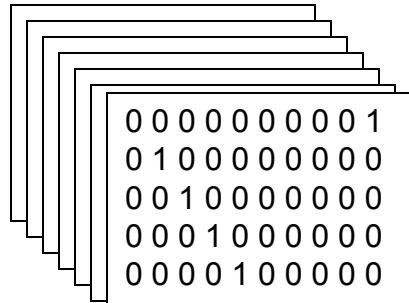
Frame Buffer



Frame Buffer

Total number of intensities achievable
out of a single pixel on the screen = 2^n

n = number of bits assigned to a single pixel



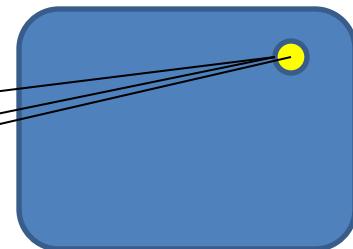
11111111

11111111

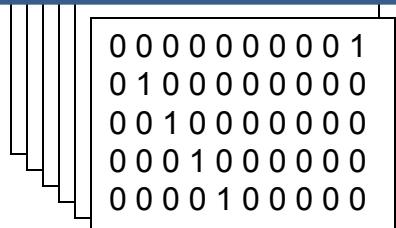
11111111

Screen

Electron
Guns



If n bits are assigned to a single pixel and the resolution of the screen is 1024×800 then the total size of the required frame buffer is $n \times 1024 \times 800$



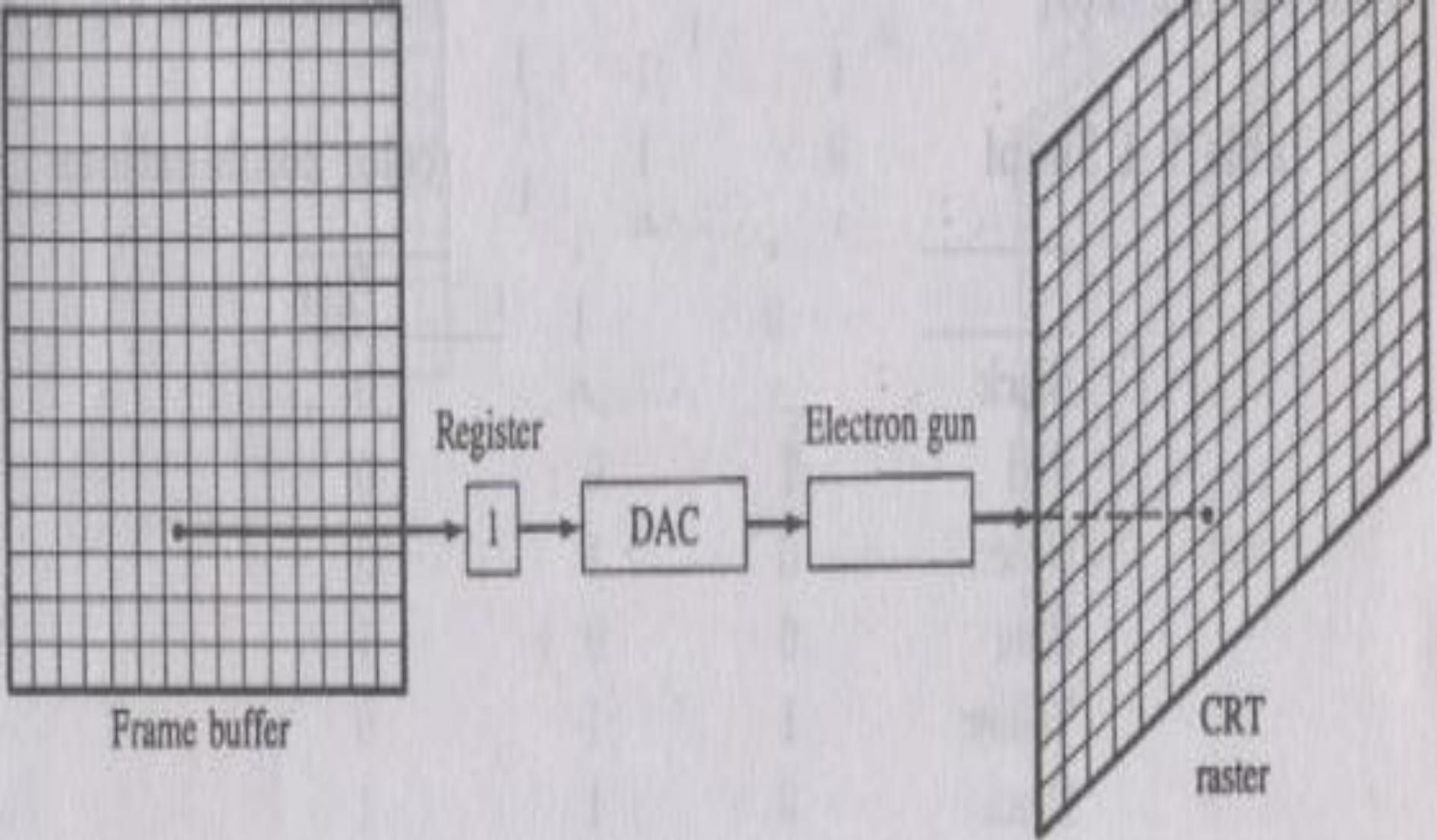
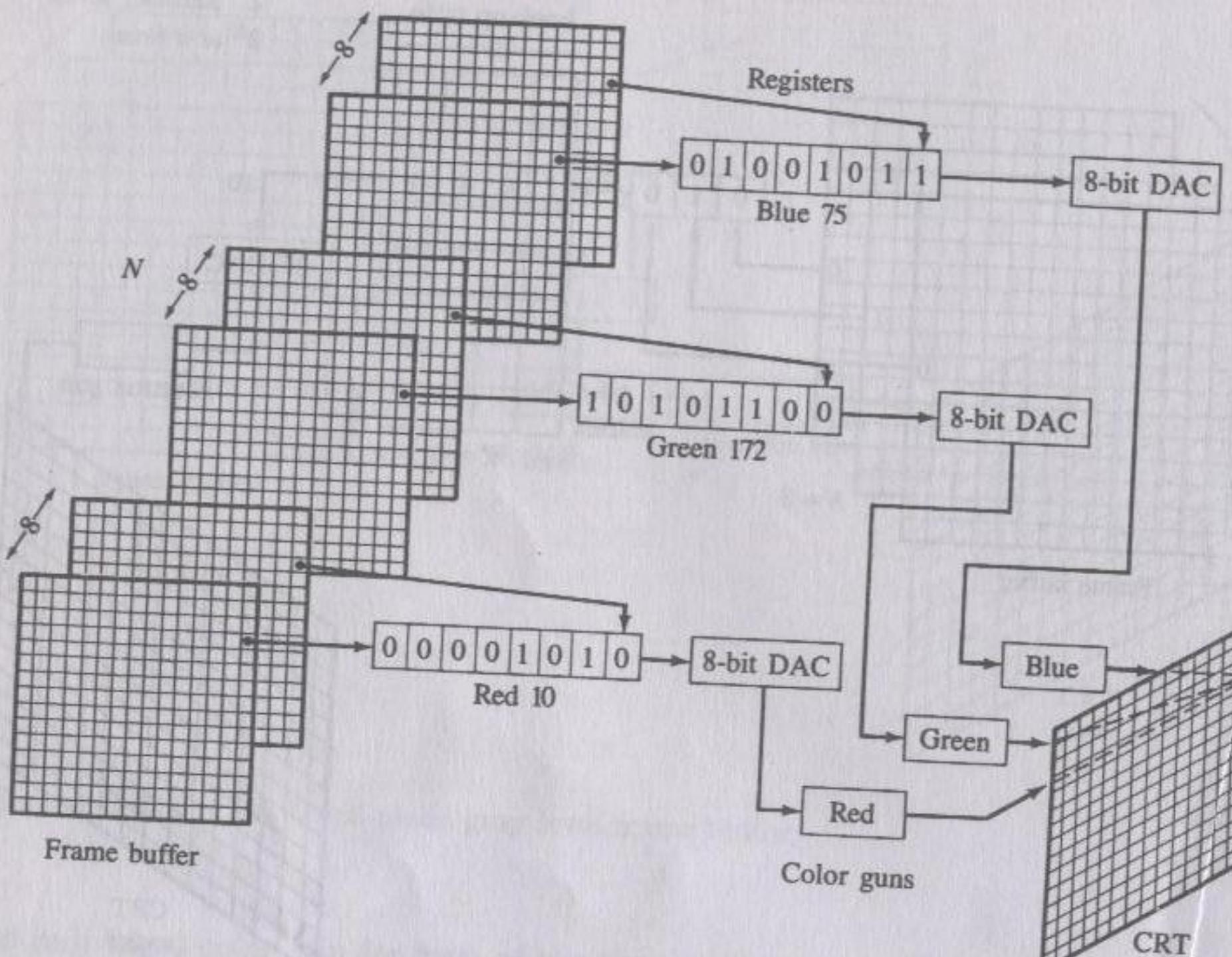
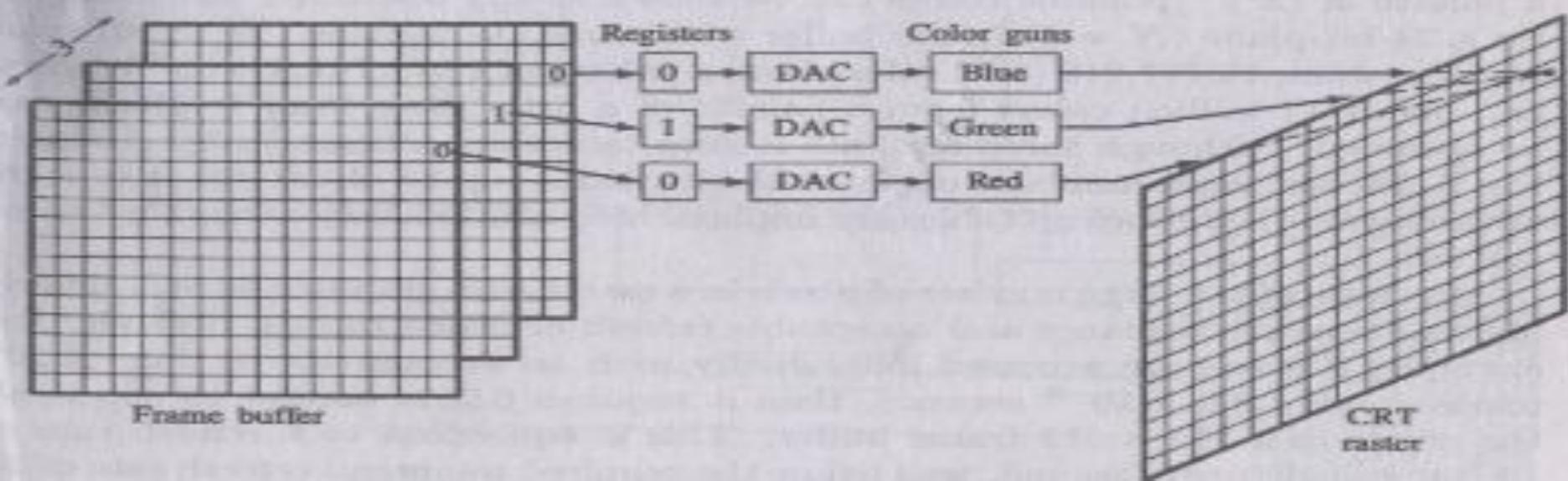
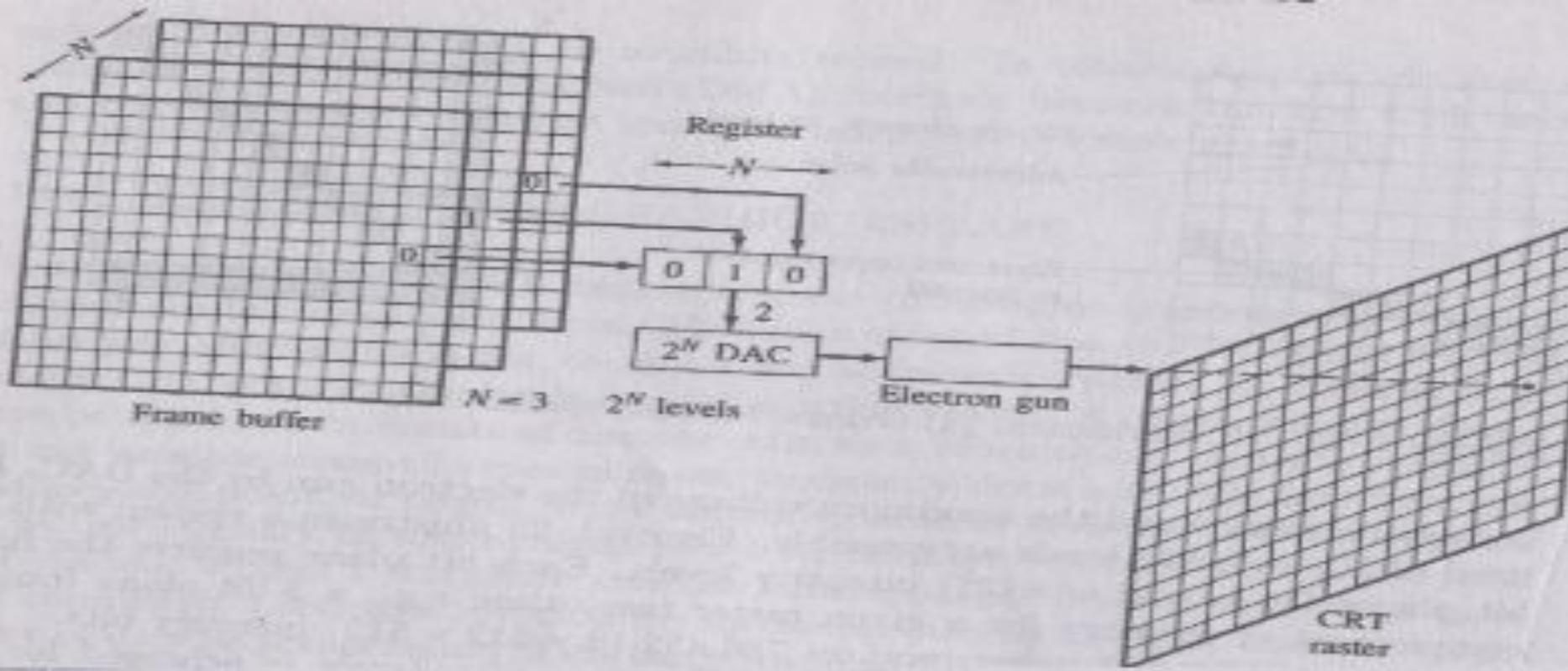


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

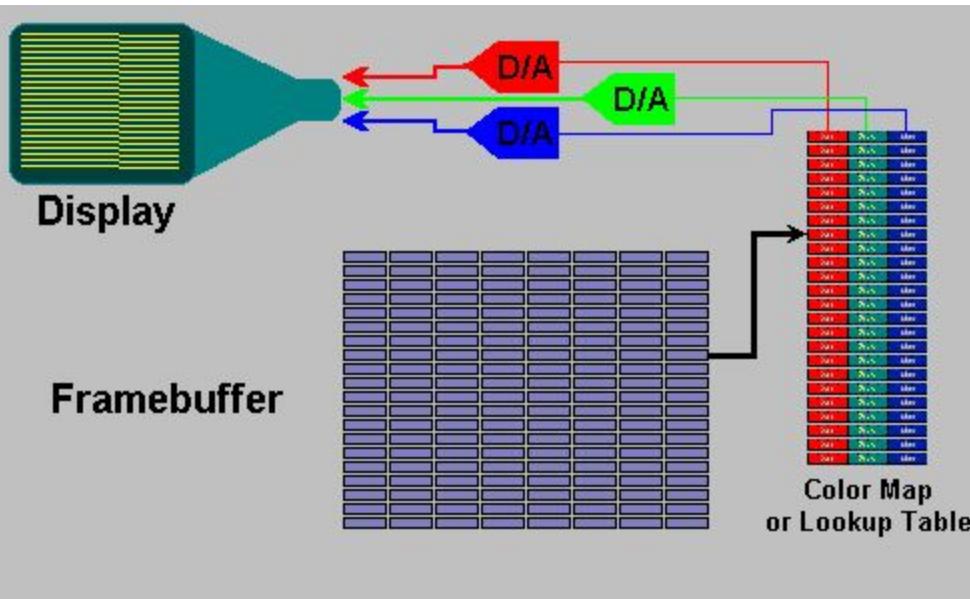
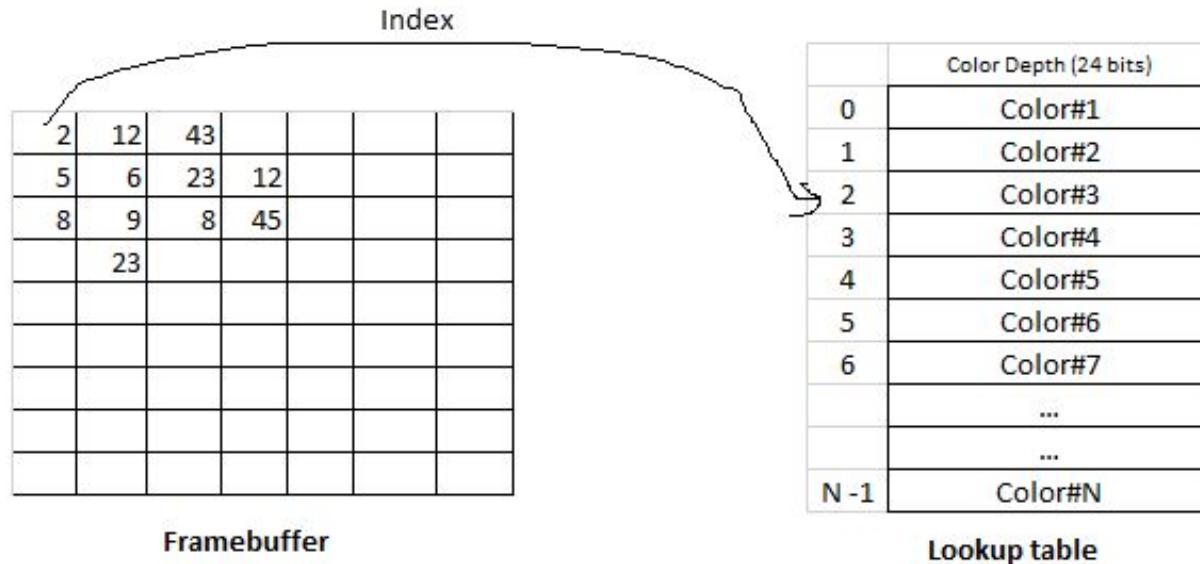




Video Cards and CRT Monitors

- Many CRT monitors use an analog signal to produce an image
- **Video card** converts digital output from the computer into an analog video signal and sends the signal through a cable to the monitor
 - Also called a **graphics card**

Look up Tables



The diagram illustrates the mapping of image pixels to colors. On the left, a 4x3 grid of pixels is shown with an 8-bit index number below it. The grid contains the following data:

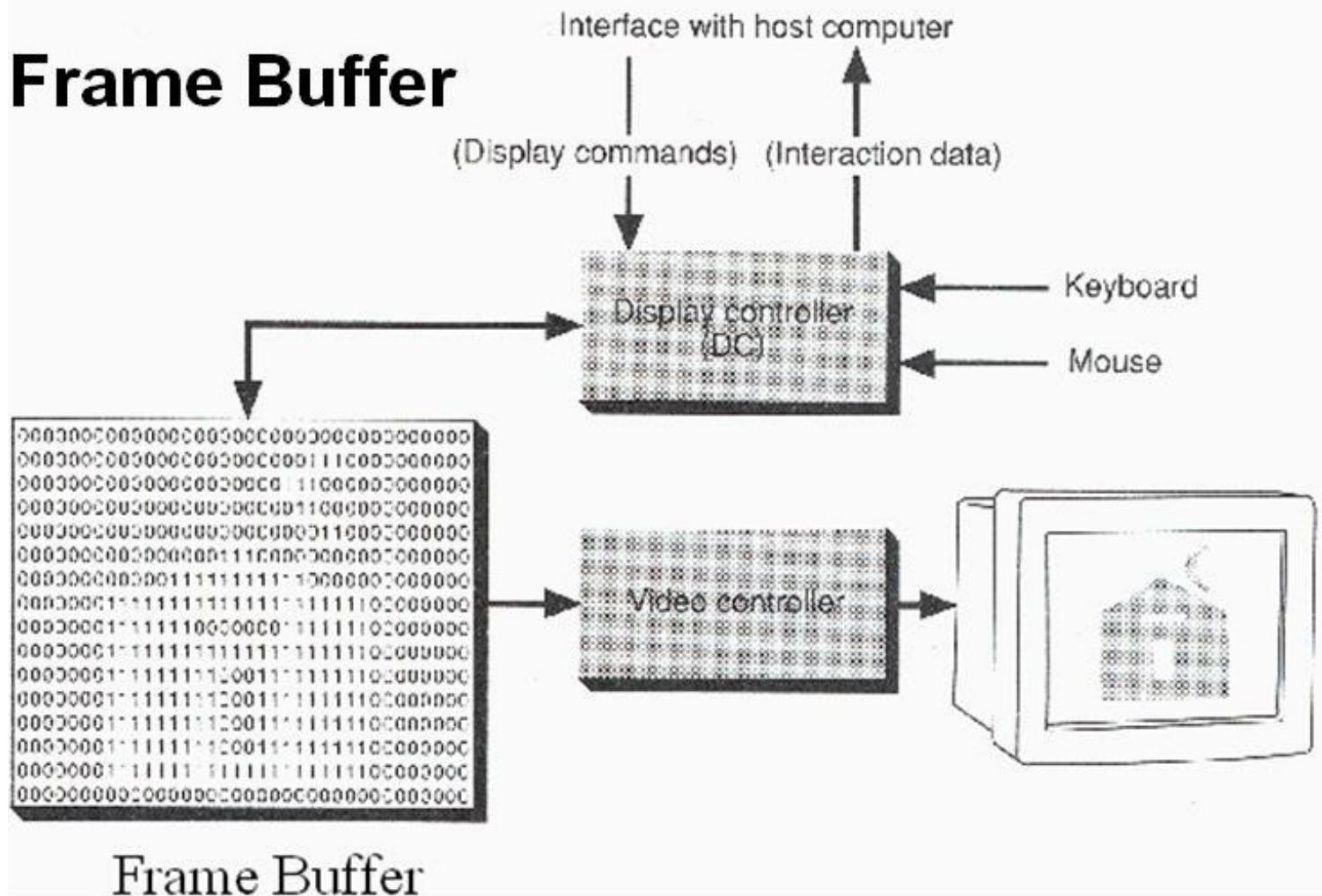
2	148	99
112	112	3
112	149	67
98	4	
254		

An 8-bit index number (Index #) is mapped to a row in the color lookup table (CLUT). The CLUT is a vertical list of 256 entries, each containing three 8-bit numbers representing an RGB value. The entries are:

0	12, 116, 0
1	255, 0, 20
2	120, 10, 15
3	43, 201, 101
4	155, 22, 233
251	112, 18, 23
252	54, 122, 0
253	87, 110, 115
254	2, 10, 254
255	90, 222, 32

A computer monitor icon on the right shows a small dot at the position corresponding to the pixel with index 3.

Frame Buffer



Video Cards and CRT Monitors

- The number of colors a video card displays is determined by its **bit depth**
- The video card's bit depth, also called the **color depth**, is the number of bits it uses to store information about each pixel
- i.e. 8-bit video card uses 8 bits to store information about each pixel; this video card can display 256 colors ($2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$)
- i.e. 24-bit video card uses 24 bits to store information about each pixel and can display 16.7 million colors
- The greater the number of bits, the better the resulting image

Video Cards and CRT Monitors



Video Display Standards Video Electronics Standards Association (VESA), which consists of video card and monitor manufacturers, develops video standards to define the resolution, number of colors, and other display properties.

Monochrome Display Adapter (MDA)

Hercules Graphics Card

Color Graphics Adapter (CGA)

Enhanced Graphics Adapter (EGA)

Video Graphics Adapter (VGA)

Super VGA (SVGA) and Other Standards Beyond VGA

Color Depth	Number of Displayed Colors	Bytes of Storage Per Pixel	Common Name for Color Depth
4-Bit	16	0.5	Standard VGA
8-Bit	256	1.0	256-Color Mode
16-Bit	65,536	2.0	High Color
24-Bit	16,777,216	3.0	True Color

Refresh rate

When electron beam strikes a dot in CRT, the surface of the CRT **only glows for a fraction of a second** and then fades.

Monitor **must redraw** the picture many times per second to avoid having the screen flicker

Refresh rate

The refresh rate is the number of times per second that monitor redraws the images on the screen.

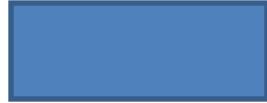
Very few people notice flicker at refresh rates above 72 Hz.

Higher refresh rates are preferred for better comfort in viewing the monitor

If the image dimension is 1366 x 768 then what is the size of the image in terms of Mega Pixels



If the image dimension is 2500 x 3192 then what is the size of the image in terms of Mega Pixels



Quality of a CRT Monitor

An RGB raster system is to be designed using an 8-inch by 10-inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for the frame buffer?

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

How long would it take to load a 640 by 480 frame buffer with 12 bits per pixel, if 10^5 bits can be transferred per second? How long would it take to load a 24-bit per pixel frame buffer with a resolution of 1280 by 1024 using this same transfer rate?

Total number of bits for the frame = $640 \times 480 \times 12$ bits = 3686400 bits

The time needed to load the frame buffer = $3686400 / 10^5$ sec = 36.864 sec

Total number of bits for the frame = $1280 \times 1024 \times 24$ bits = 31457280 bits

The time needed to load the frame buffer = $31457280 / 10^5$ sec = 314.5728 sec

Quality of a CRT Monitor

Assuming that a certain full-color (24-bit per pixel) RGB raster system has a 512-by-512 frame buffer, how many distinct color choices (intensity levels) would we have available? How many different colors could we display at any one time?

Total number of distinct color available is 2^{24}

Total number of colors we could display at one time is 512×512

Assuming that a certain RGB raster system has 512×512 frame buffer with 12 bit per pixel

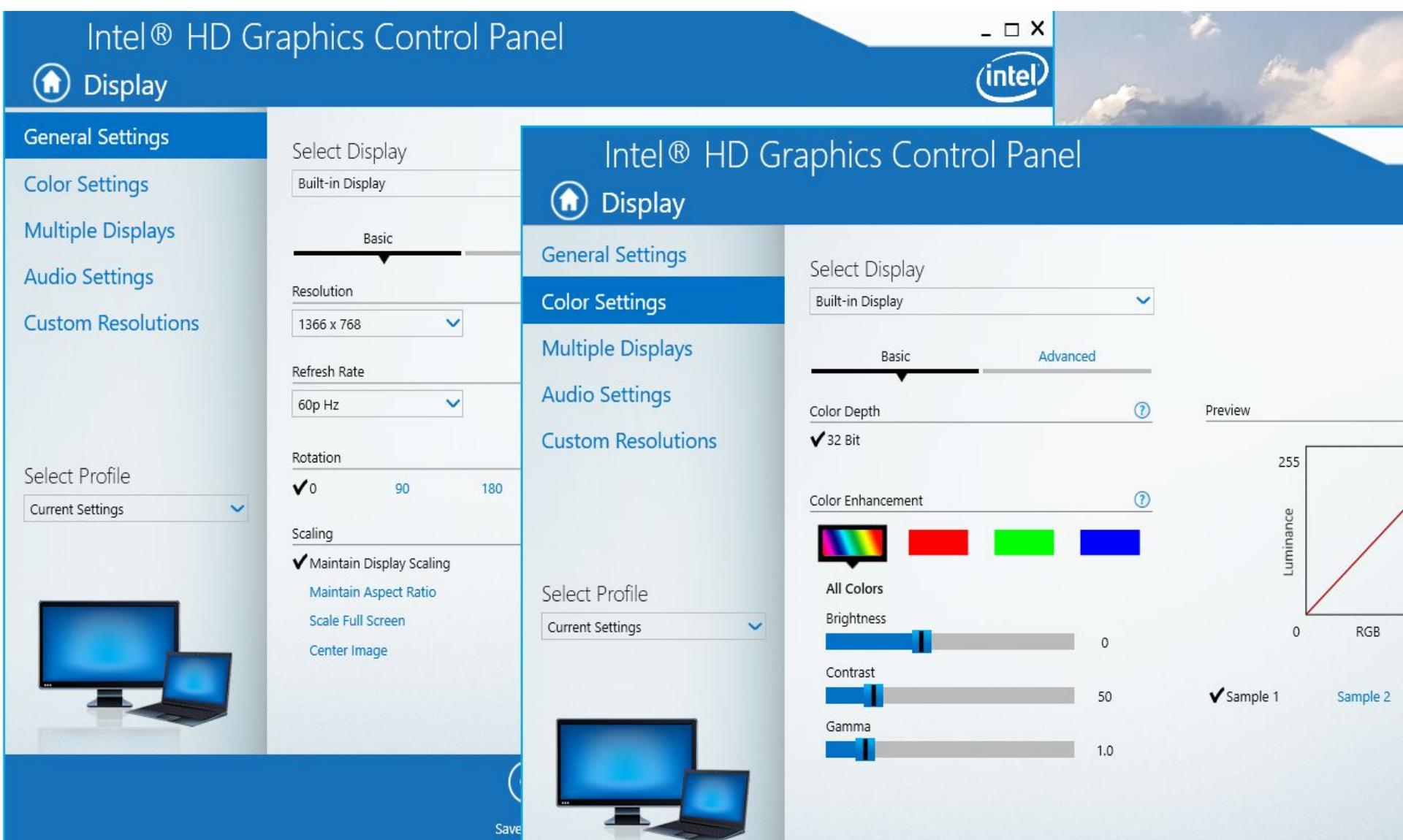
1. How many distinct color choice we have available
2. How many different color could we display at any one time?

Total number of distinct color available is 2^{24}

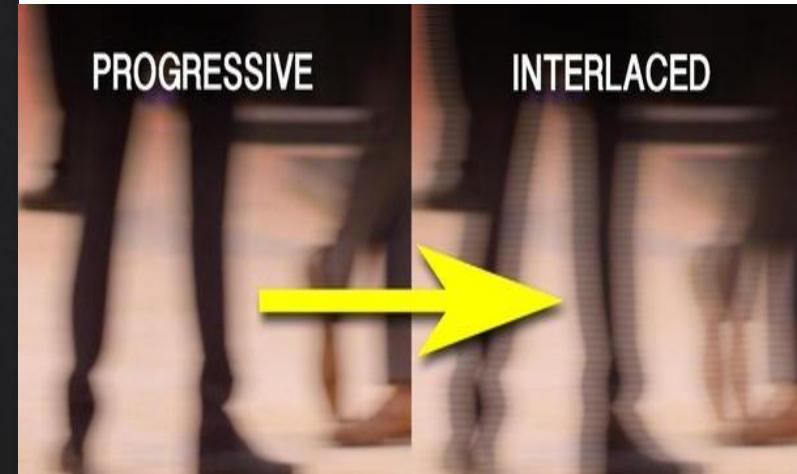
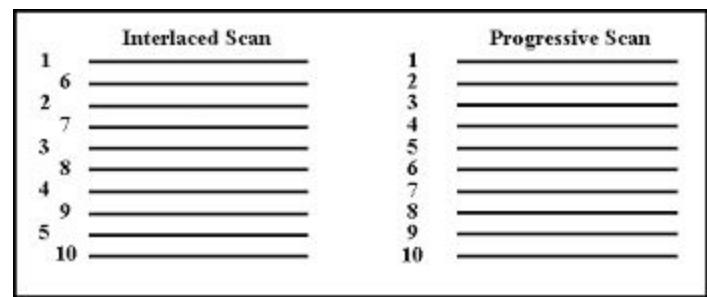
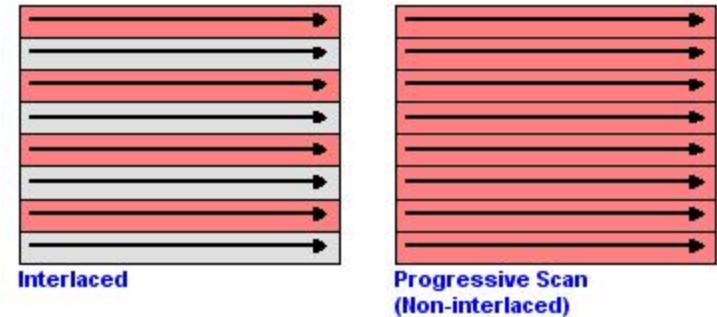
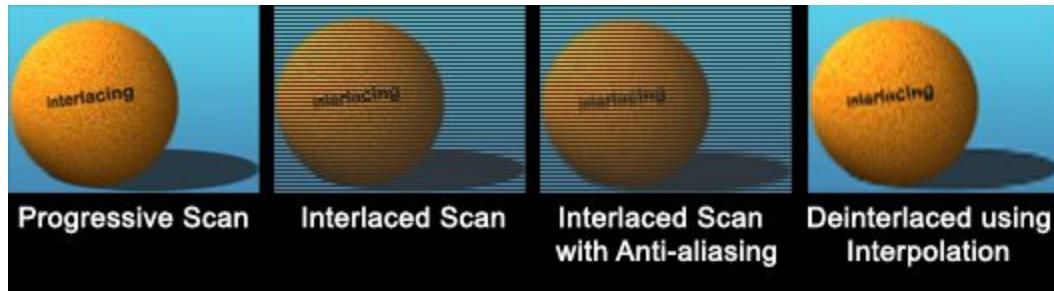
Total number of different color could display at any one time is 2^{12}

The storage spent for frame buffer is $512 \times 512 \times 12$ bit = 3145728 bit

Quality of a CRT Monitor

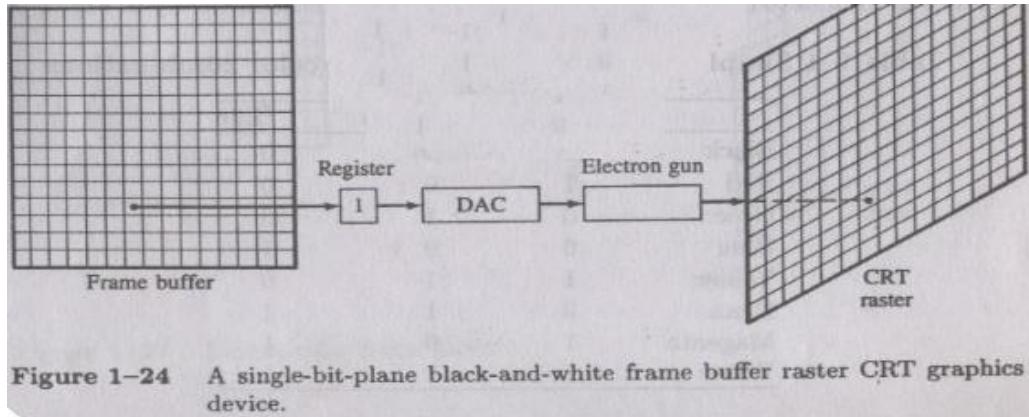


Refresh Rate 60i Hz or 60p Hz



Quality of a CRT Monitor

If on an average it takes 20 nanoseconds to glow a single pixel on the screen and the resolution of the screen is 1024 x 800, will there be a flickering effect seen on the screen?



To glow 1 pixel on screen it takes 20×10^{-9} s

T = To glow all pixels on screen it takes **$1024 \times 800 \times 20 \times 10^{-9}$ s**

$$F = 1 / T = 1 / (1024 \times 800 \times 20 \times 10^{-9})$$

System Refresh rate = 61 frames per sec

Standard refresh rate = 50 frames per sec

There will be no flickering effect seen on the screen as the refresh rate of the screen is 61 fps which is higher than the standard refresh rate!!!

Quality of a CRT Monitor

What is the fraction of total refresh time spent in retrace of electron beam for a non interlaced raster system with a resolution of $m \times n$ and refresh rate of r Hz , the horizontal retrace time is t_h microsecond and vertical retrace time of t_v microsecond?

Here ,

$$\text{The fraction of total refresh time} = \frac{n \times t_h + t_v}{1/r}$$

spent in retrace of electron beam

where t_h is horizontal retrace

t_v is vertical retrace and r is the refresh rate

$n \times m$ = 'm' scan lines and 'n' pixels in each scan line

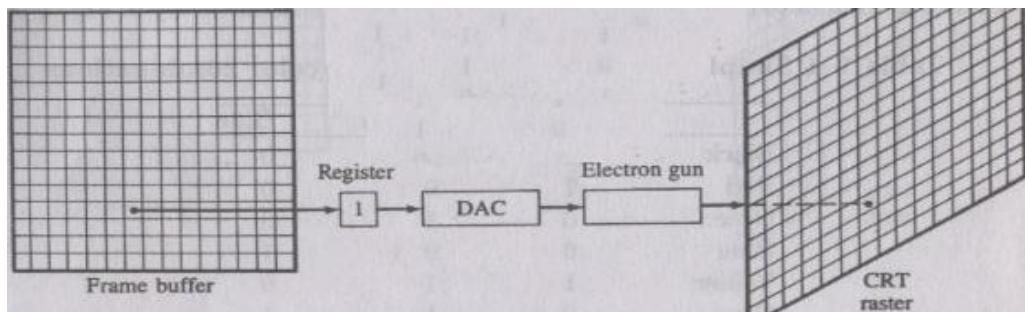


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

Quality of a CRT Monitor

Consider a raster system with resolution of 640 by 480. How many pixels could be accessed per second in this system by a display controller that refreshes the screen at a rate of 60 frames per second? What is the access time per pixel in this system?

The access time per pixel is $1 / (640 \times 480 \times 60)$ sec

The access time per pixel is $1 / (1280 \times 1024 \times 60)$ sec

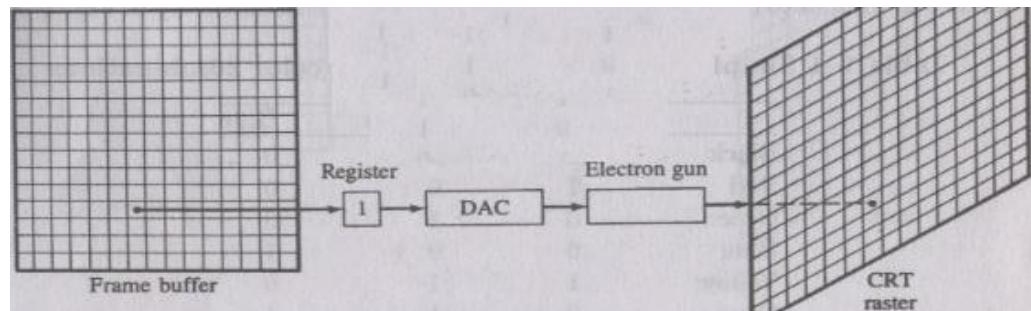
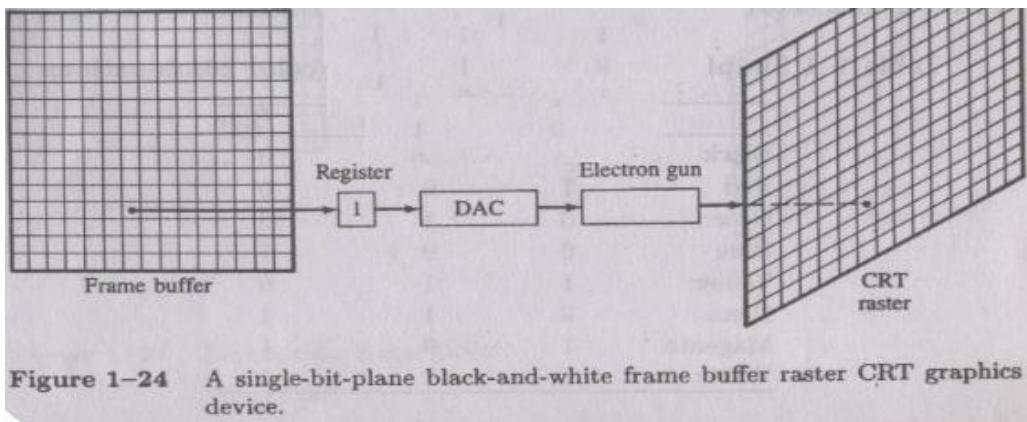


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

Quality of a CRT Monitor

A Raster system can produce a total number of 1024 different levels of intensities from a single pixel of a black and white monitor. If the total resolution of the screen is 640×480 what will be the required size of frame buffer for the display purpose?

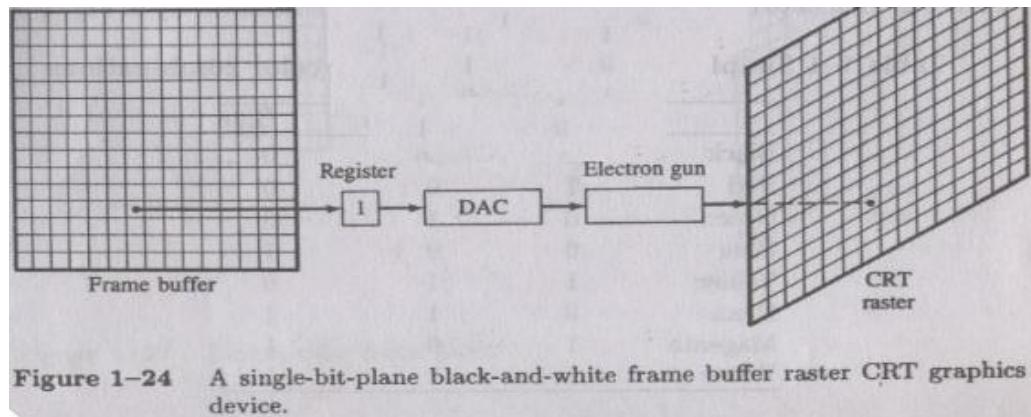


Your mobile phone has a total resolution of 1080×2160 with 401 PPI. What is the display size of your mobile?

Quality of a CRT Monitor

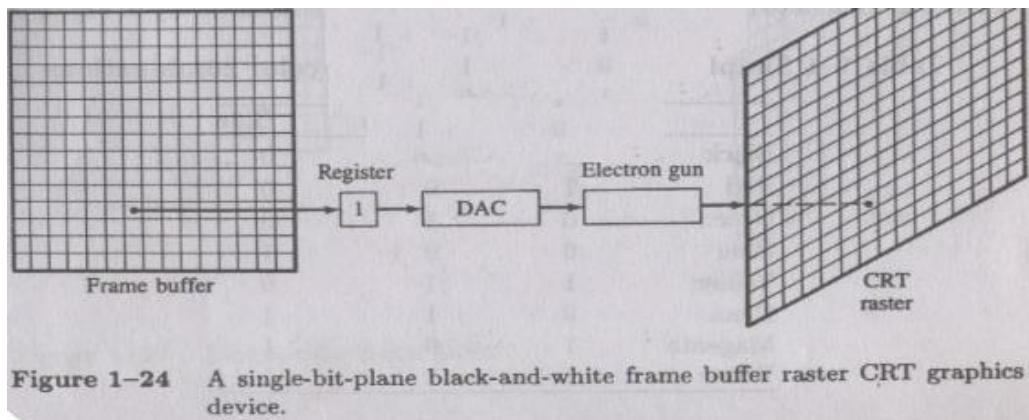
How much time is spent scanning across each row of pixels during screen refresh on a raster system with a resolution of 1280×1024 and refresh rate of 60 frames per second.

Here time required to refresh the screen (t) = $1/60$ seconds



Quality of a CRT Monitor

A Raster system can produce a total number of 1024 different levels of intensities from a single pixel composed of red, green and blue phosphor dots. If the total resolution of the screen is 640×480 what will be the required size of frame buffer for the display purpose?

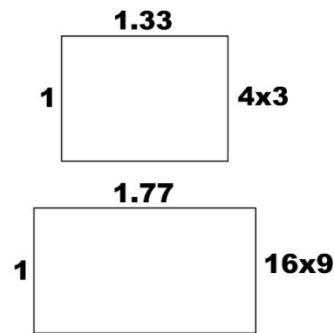
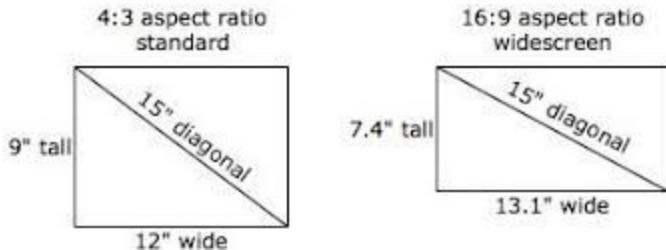
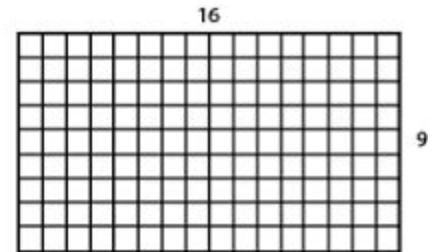
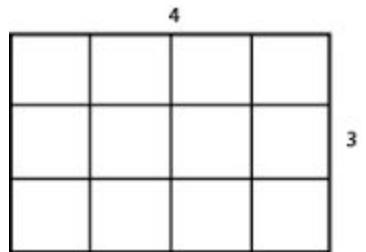
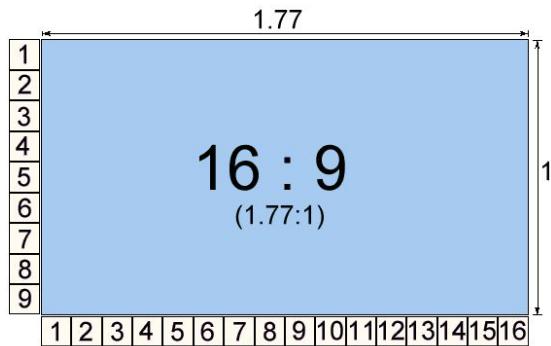


Your mobile phone has a total resolution of 640×480

with PPI. What is the

Aspect Ratio

The ratio of vertical points to the horizontal points necessary to produce length of lines in both directions of the screen is called Aspect Ratio.
Usually the aspect ratio is $\frac{3}{4}$.



Aspect Ratio

Aspect Ratio

4:3	1.33 : 1
16:9	1.77 : 1
21:9	2.35 : 1

TVs

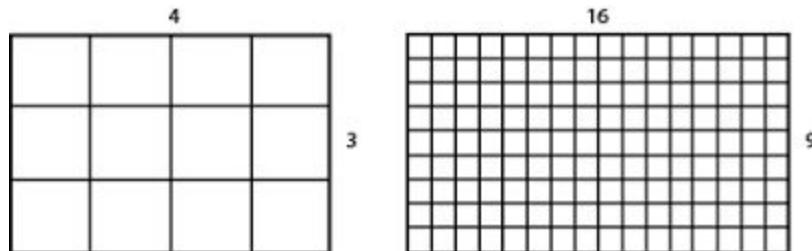
Old TV
HDTVs
Theaters

Aspect Ratio

Suppose we have a video monitor with display area that measures 12 inches across and 9.6 inches high. If resolution is 1280 by 1024 what is the aspect ratio and the diameter of each pixel?

$$\text{Aspect ratio} = w/h \quad \text{i.e.} \quad 1280/12 / 1024/9.6 = 1$$

$$\text{Diameter of each pixel} = 12/1280 \text{ or } 9.6/1024 = 0.0094 \text{ inches}$$



Computer Graphics

Computer Graphics, Saroj Shakya, Nepal
College of Information Technology,
Balkumari

Computer Graphics

A field related to the generation of graphics using computers

- It includes the **creation, storage and manipulation** of images of objects.

Until early 1980's it was a small specialized field

- **Hardware** was expensive
- Graphics based **application programs** that were easy to use and cost effective were few

PCs with built in raster graphics displays such as the **Xerox Star, Apple Macintosh** and the **IBM PC** – popularized the use of **bitmap graphics** for user computer interaction.

A bitmap is a 1 and 0s representation of the rectangular array of points on the screen.

- Each point is called a pixel, short for “**picture elements**”.

Computer Graphics

Once the bitmap graphics became **affordable** an explosion of easy to use, inexpensive graphics based user interfaces allowed millions of new users to control simple low cost application programs such as **word processors, spreadsheets and drawing programs**

The concept of a “**desktop**” now became popular metaphor for organizing screen space.

Even people who do not use computers encounter computer graphics in TV commercials and as cinematic special effects.

Computer Graphics

Thus computer graphics is an **integral part** of all computer user interfaces, and is indispensable for visualizing 2D, 3D objects in all most all areas such as **education , science , engineering, medicine, commerce the military advertizing and entertainment.**

The theme is that learning how to program and use computers now includes learning how to use simple **2D graphics**

Early History of Computer Graphics

Crude plotting of hardcopy devices such as **teletypes** and **line printers** dates from the early days of computing

The **whirlwind computer** developed in 1950 at the Massachusetts Institute of Technology (MIT) had **computer driven CRT displays** for output.



Computer Graphics, Saroj Shak
College of Information Techn
Balkumari



Early History of Computer Graphics

SAGE air-defense system developed in the middle 1950s was the first to **use command and control CRT display consoles** on which operators identified targets with light pens
(hand held pointing devices that sense light emitted by objects on the screen)

The **General Motors DAC** system for **automobile design** and the **Itek-Digitek system** for **lens design**



Computer Graphics, Saroj Shakya, Nepal
College of Information Technology,
Balkumari

Early History of Computer Graphics



Later on **Sketchpad system** by Ivan Sutherland came in light.

- The beginning of **modern interactive graphics**.
- **Keyboard and light pen** used for pointing, making choices drawing.

Prospects of the use of **(CAD)** and **(CAM)** in computer, automobile, and aerospace manufacturing Drafting and Drawing activities.

By the mid 60s, a number of **commercial products** using these systems had appeared

Early History

At that time only the **most technology intensive organizations** could use the interactive computer graphics where as others used **punch cards**, a non-interactive system .

Reasons:

The **high cost of graphics hardware** – at a time when automobiles cost a few thousand dollars, computers cost several millions of dollars, and the first commercial computer displays cost more than a hundred thousand dollars

The need for large scale **expensive computing resources** to support massive design database

The difficulty of writing large interactive programs using **batch oriented FORTRAN programming**

Early History of Computer Graphics

One of a kind , **non-portable software**, typically written for a particular manufacturer's display devices.

(When software is non-portable, moving to new display devices necessitates expensive and time consuming rewriting of working programs)

Thus **interactive computer graphics** had a limited use when it started in the early sixties but it became very common once the **Apple Macintosh** and **IBM PC** appeared in the market with affordable cost

Uses of Computer Graphics

User interfaces

Plotting

Office automation
and electronic
publishing

Art and commerce

Cartography

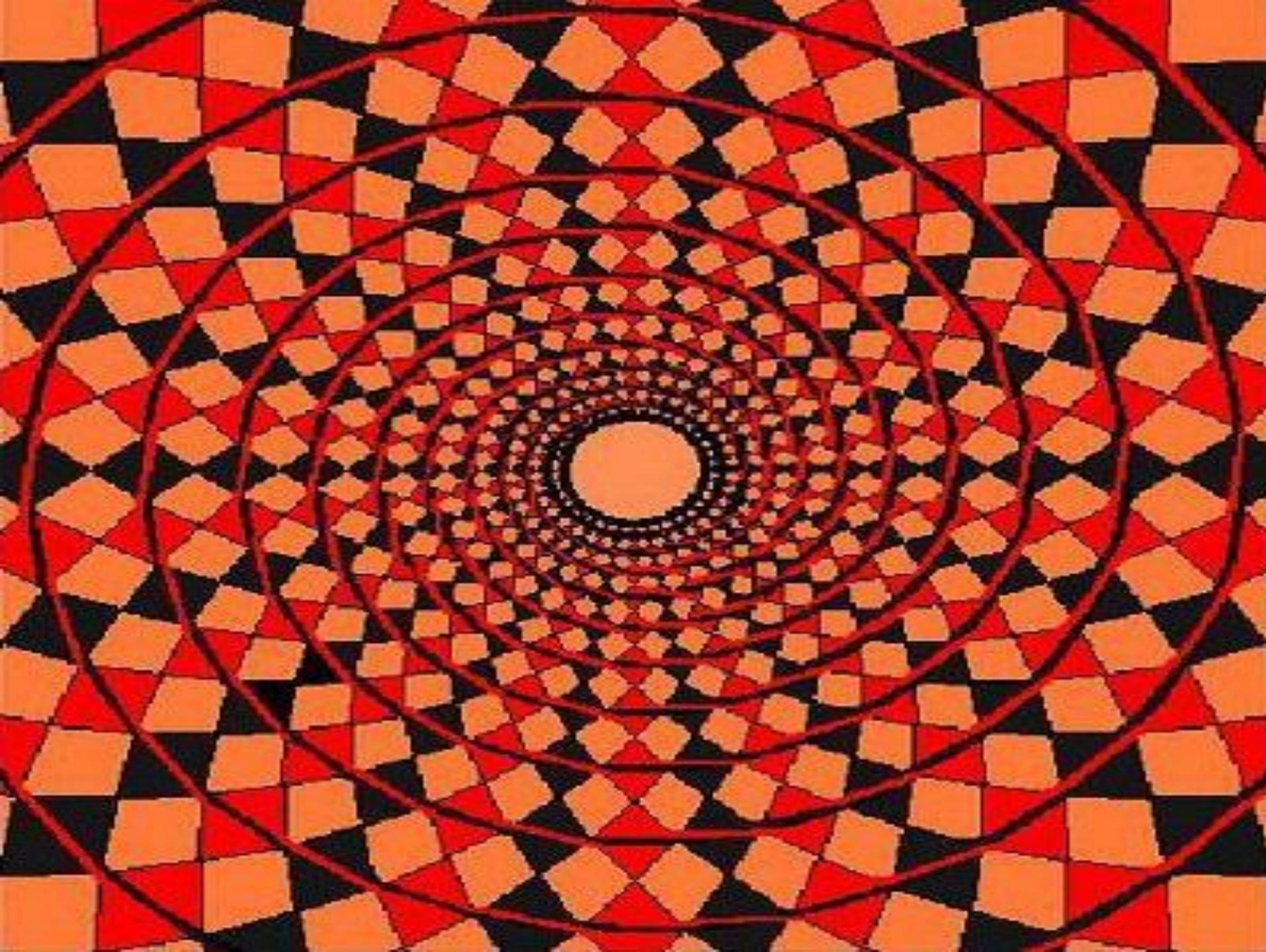
Computer aided
drafting and
design

Entertainment

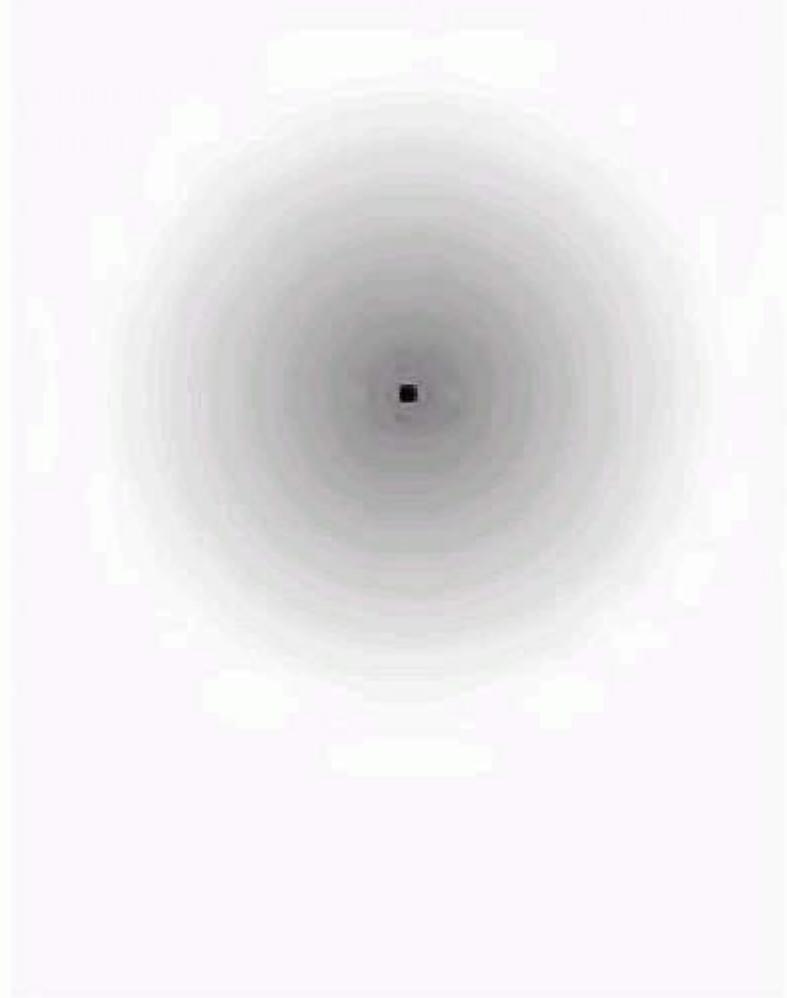
Simulation

Scientific and
business
visualization

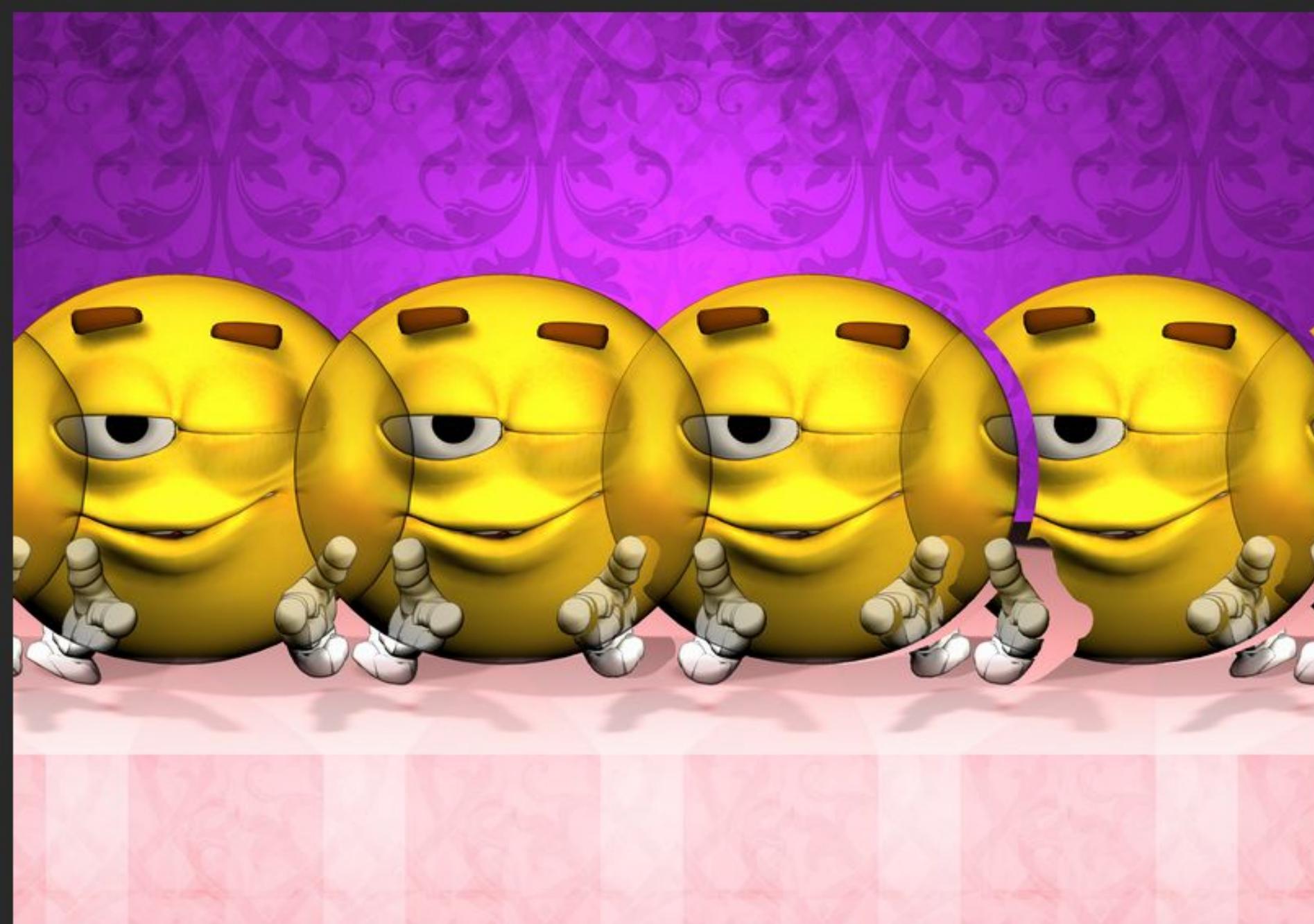




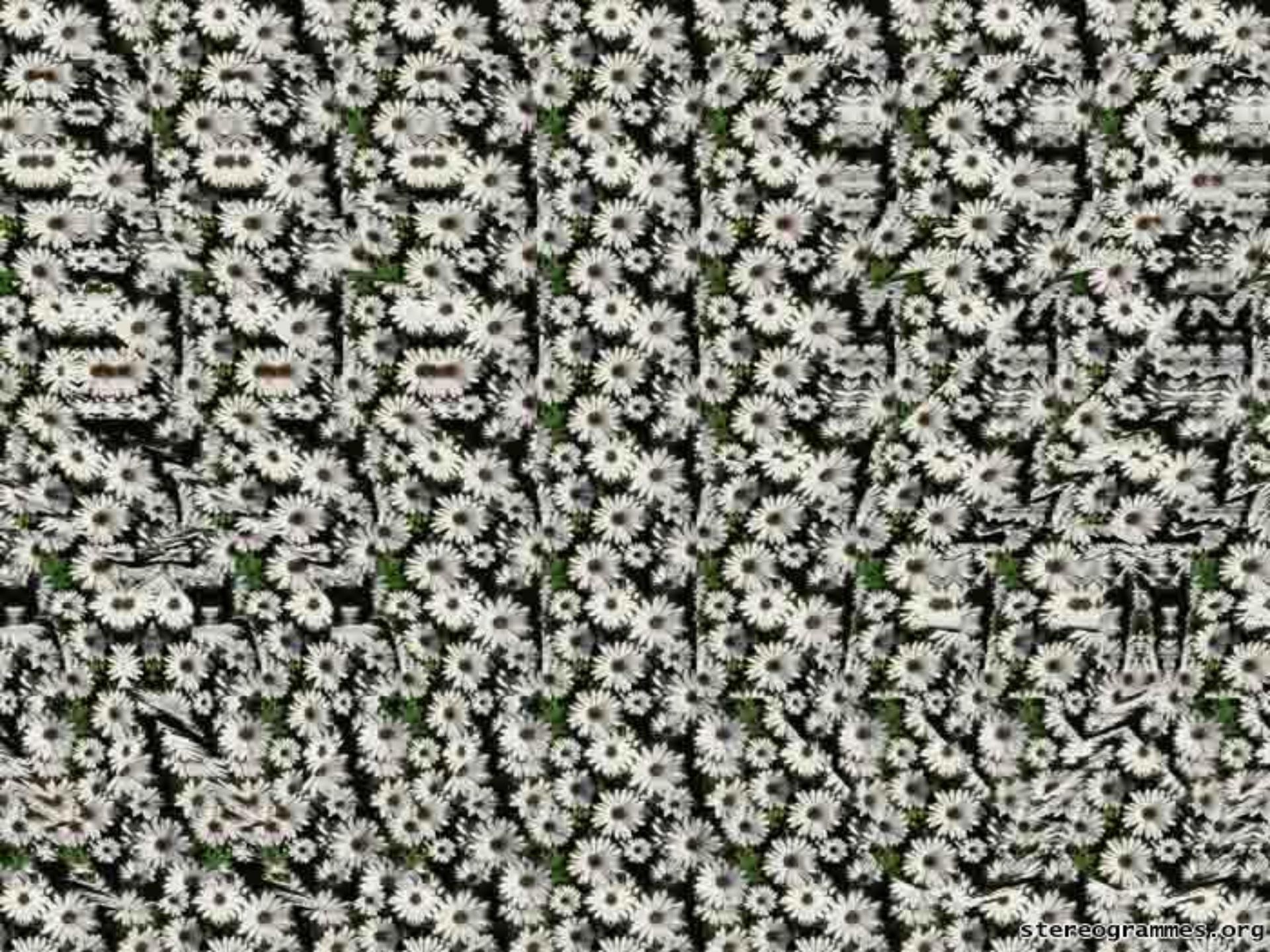
Keep staring at the black dot. After a while the gray haze around it will appear to shrink.







Created with Stereographic Suite
<http://www.indasoftware.com/stereo/>





Major components (HW/SW) for computer graphics

Hardware Components:
monitors, mouse and joy-sticker, and hard-copy devices, e.g. high-resolution color printer

For software Components:
Special purpose utilities (device-dependent and device independent)
needed for handling processing

Input Hardware

Tablet

Tablet a tablet is a **digitizer**.

- scan over an object and input a set of discrete coordinate positions.

These positions can then be joined with straight line segments to approximate the shape of the original object.

A tablet digitizes an object detecting the position of a movable stylus (a pencil shaped device) or a puck(a mouse like device with cross hairs for sighting positions) held in the user's hand

Input Hardware



Input Hardware

i. Electrical Tablet

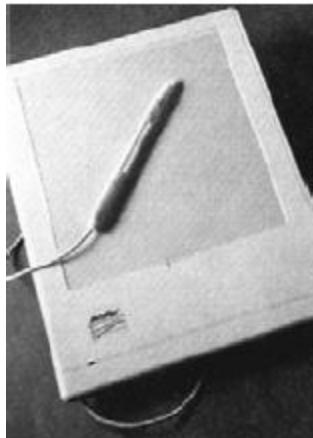
A grid of wires is embedded in the tablet surface

Electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induce an electrical signal in a wire coil in the stylus or puck

The strength of the signal induced by each pulse is used to determine the position of the stylus.

The signal strength is also used to determine roughly how far the stylus is from the tablet

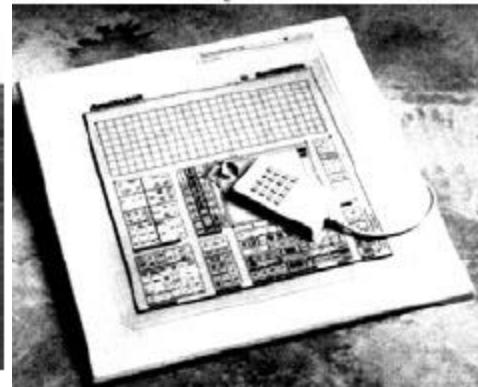
Input Devices: Tablets



desktop tablet
with stylus



artist's digitizer system
with cordless stylus



desktop tablet with a
16-button hand cursor



large tablet with
hand cursor

Input Devices: Tablets

Input Hardware

ii. Sonic Tablet

The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area

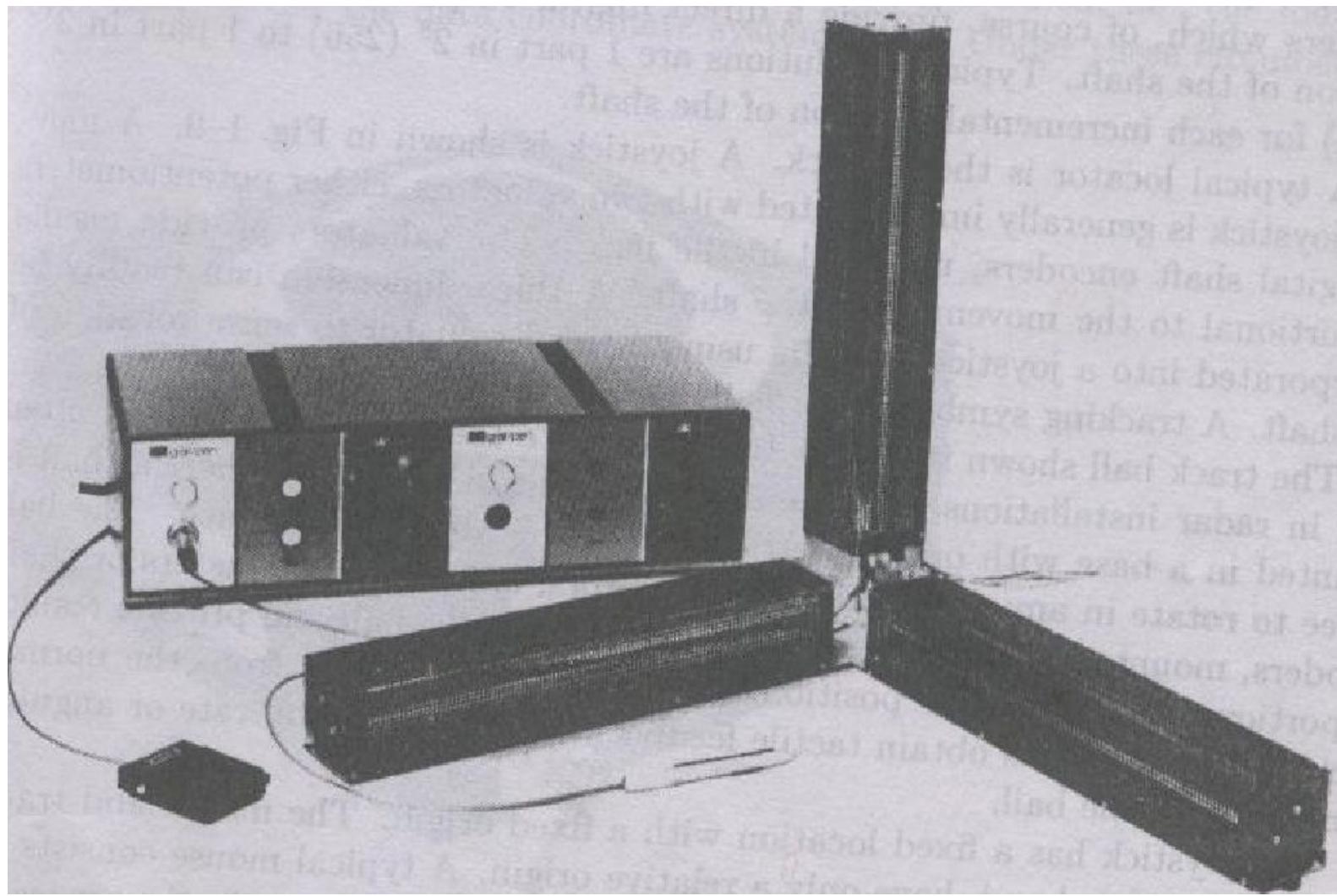
An electrical spark at the tip of the stylus creates **sound bursts**.

The position of the stylus or the coordinate values is calculated using the **delay between** when the spark occurs and when its sound arrives at each microphone

Input Hardware

The main advantage of sonic tablet is that it **doesn't require a dedicated working area** as the microphones can be placed on any surface to form the tablet work area

This facilitates digitizing drawing on thick books because in an electrical tablet this is not convenient for the stylus can not get closer to the tablet surface



3D Digitizers



manual digitizer
with stylus

3D Digitizers



MicroScribe G2
Desktop 3D Digitizer

Input Hardware

iii. Resistive Tablet

The tablet is just a piece of glass coated with a thin layer of conducting material

When a battery powered stylus is activated at certain position it emits high frequency radio signals which induces the radio signals on conducting layer.

The strength of the signal received at the edges of the tablet is used to calculate the position of the stylus

Input Hardware

Several types of tablets are transparent, and thus can be backlit for digitizing x-ray films and photographic negatives.

The Resistive tablet can be used to digitize the objects on CRT because it can be curved to the shape of the CRT.

The mechanism used in the electrical or sonic tablets can also be used to digitize the 3D objects

Input Hardware

Touch Panels

The touch panel allows the user to point at the screen directly with a finger to move the cursor around the screen or to select the icons.

i. Optical Touch Panel

It uses a series of infrared **light emitting diodes (LED)** along one vertical edge and along one horizontal edge of the panel

The opposite vertical and horizontal edges contain **photo detectors** to form a grid of invisible infrared light beams over the display area.

Input Hardware

Touching the screen breaks one or two vertical and horizontal light beams thereby indicating the fingers position

The cursor is then moved to this position or the icon at this position is selected

This is a low resolution panel which offers 10 to 50 positions in each direction

Input Devices: Touch Screens



plasma panels with touch screens

Input Hardware

ii. Sonic Touch Panel

Bursts of high frequency sound waves traveling alternately horizontally and vertically are generated at the edge of the panel .

Touching the screen causes part of each wave to be reflected back to its source

The screen position at the point of contact is then calculated using the time elapsed between when the wave is emitted and when it arrives back at the source

This is a high resolution touch panel having about 500 positions in each direction

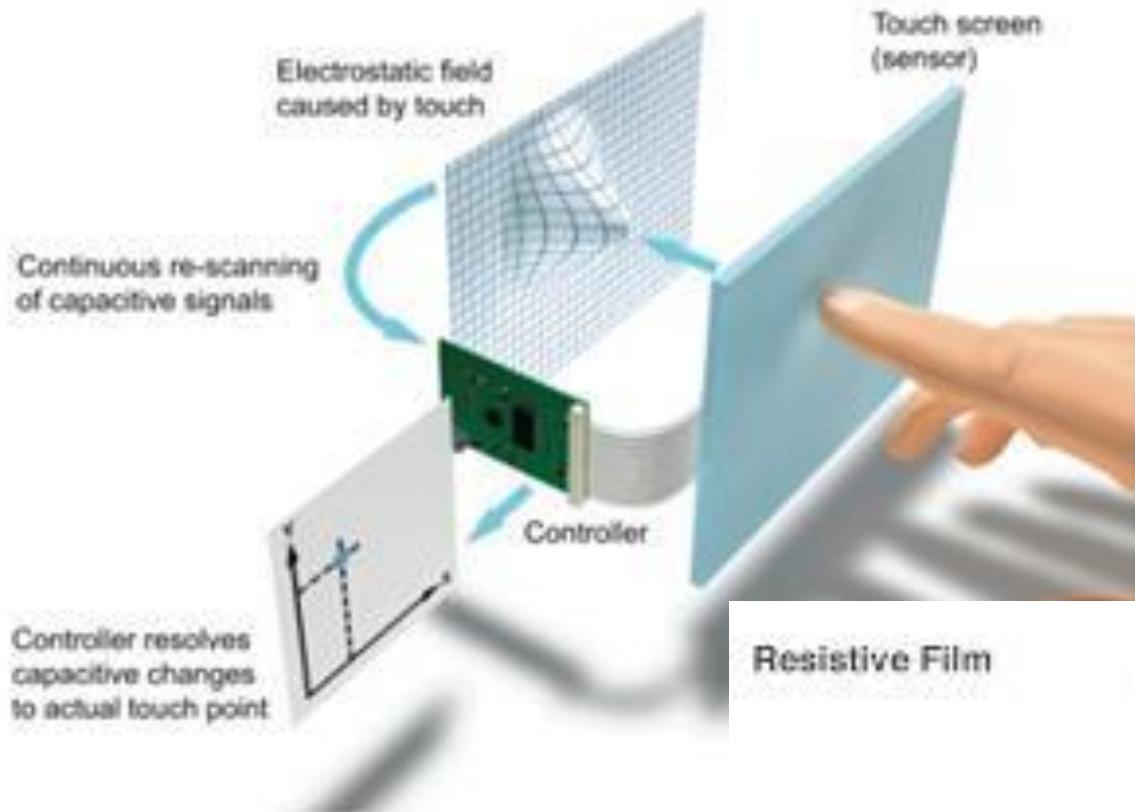
Input Hardware

iii. Electrical Touch Panel

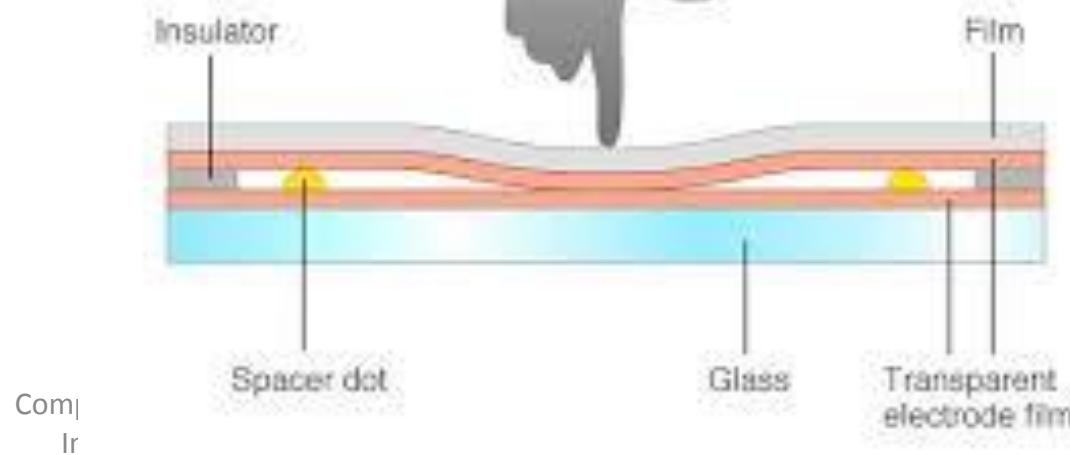
It consists of slightly separated two transparent panel one coated with a thin layer of conducting material and the other with resistive material

When the panel is touched with a finger the two plates are forced to touch at the point of contact thereby creating the voltage drop across the resistive plate which is then used to calculate the coordinate of the touched position

The resolution of the touch panel is similar to that of sonic touch panel



Resistive Film



Input Hardware

Light pen

It is a pencil shaped device to determine the coordinates of a point on the screen where it is activated such as pressing the button .

In raster display 'y' is set at y_{max} and 'x' changes from 0 to x_{max} the first scan line .

For the second line 'y' decreases by one and 'x' again changes from 0 to x_{max} and so on

When activated light pen sees a burst of light at certain position as the electron beam hits the phosphor coating at that position it generates an electric pulse

Input Hardware

This is used to save the video controller's 'x' and 'y' registers and interrupt the computer

By reading the saved values the graphics package can determine the coordinates of the position seen by the light pen

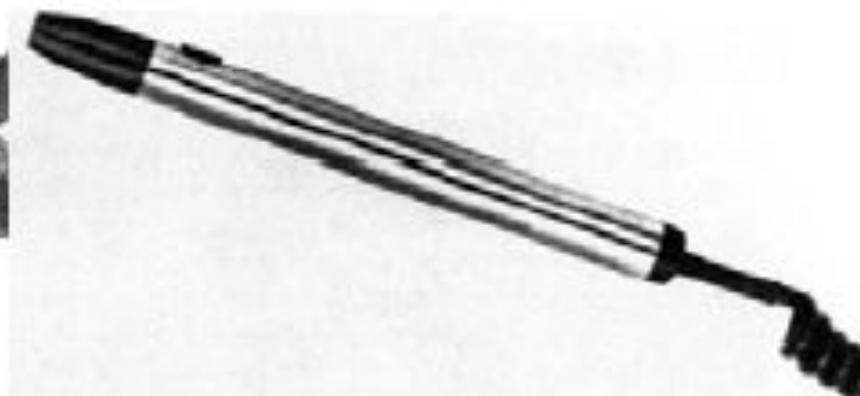
Drawbacks

- i. Light pen **obscures screen images** as it is pointed to required spot
- ii. Prolong use of it can cause **arm fatigue**
- iii. It cannot report the coordinates of a point that is **completely black** as a remedy one can display a dark blue field in place of the regular image for a single frame time
- iv. It gives sometimes **false reading** due to background lighting in a room

Input Devices: Light Pen



recognizes cathod ray
to calculate position



Types of Displays:

- Emissive Displays
 - The emissive display **converts electrical energy into light energy.**
 - The **image is produced directly on the screen** Phosphors convert electron beams or UV light into visible light
 - e.g.: Plasma, Cathode Ray Tube (CRT) Light-Emitting Diode Displays (LED), Plasma Display Panel (PDP)
- Non-Emissive Displays
 - **Light is produced behind the screen** and the image is formed by filtering this light e.g.: LC-Display (LCD)
 - The Non emissive display uses **optical effects to convert the sunlight or light from any other source to graphic form**

Light Emitting Diode Monitors

Light Emitting Diode (LED) is an improved version of LCD monitor and manufacturers have tried to eliminate the drawbacks of LCD monitors.

They differ in backlighting as LCD monitors use Cold Cathode Fluorescent Light and LED monitors are based on light emitting diode.

The backlighting impacts badly on the image and decreases its sharpness and brightness.

WLED and RGB LED are the two types of LED monitors, depending on the way LED placed in the panel.



Benefits of LED over CRT and LCD Monitors

LED monitors give a high-quality image with vibrant colors and viewing comfort.

LCD monitors are unable to display black and white image while LED monitors are capable of producing true black hues.

It consumes less electrical energy than CRT and LCD monitors as a cold cathode fluorescent lamp is embedded in the panel.

The absence of mercury makes it eco-friendly while zero percent flickering removes the chances of strain on the eyes

Video Display Standards Video Electronics Standards Association (VESA), which consists of video card and monitor manufacturers, develops video standards to define the resolution, number of colors, and other display properties.

Monochrome Display Adapter (MDA)

Hercules Graphics Card

Color Graphics Adapter (CGA)

Enhanced Graphics Adapter (EGA)

Video Graphics Adapter (VGA)

Super VGA (SVGA) and Other Standards Beyond VGA

Video Display Standards Video Electronics Standards Association (VESA), which consists of video card and monitor manufacturers, develops video standards to define the resolution, number of colors, and other display properties.

Monochrome Display Adapter (MDA)

Hercules Graphics Card

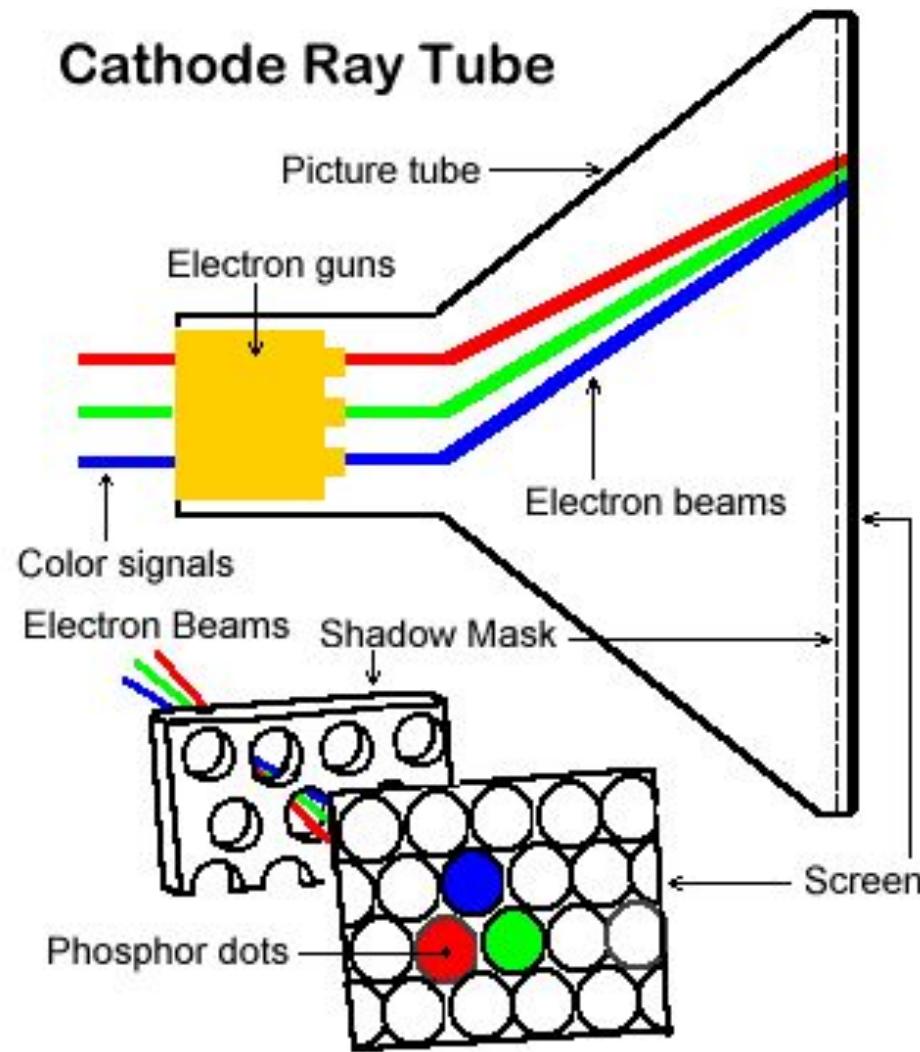
Color Graphics Adapter (CGA)

Enhanced Graphics Adapter (EGA)

Video Graphics Adapter (VGA)

Super VGA (SVGA) and Other Standards Beyond VGA

Cathode Ray Tube (CRT)

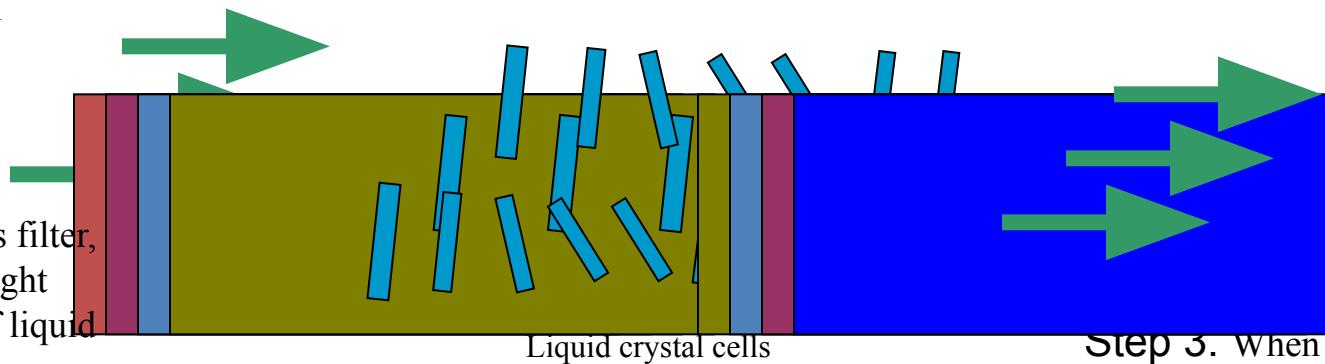


LCD (Liquid Crystal Display)

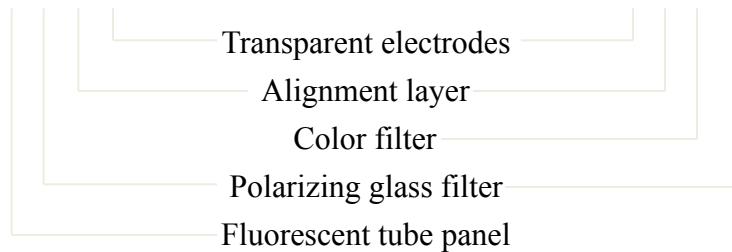
- LCD (Liquid Crystal Display)
 - A type of flat-panel display
 - Uses **liquid crystals** between two sheets of material to present information on a screen
 - An electric current passes through the liquid crystals, **they twist**
 - Depending on **how much they twist, some light waves are passed through** while other light waves are blocked. This creates the variety of color that appears on the screen

How LCD works?

Step 1. Panel of fluorescent tubes emits light waves through polarizing glass filter, which guides light toward layer of liquid crystal cells.



Step 2. As light passes through liquid crystal, electrical charge causes some of the cells to twist, making light waves bend as they pass through color filter.



Step 3. When light reaches second polarizing glass filter, light is allowed to pass through any cells that line up at the first polarizing glass filter. Absence and presence of colored light cause image to display on the screen.

LCD (Liquid Crystal Display)

- LCD monitors produce color using either passive-matrix or active-matrix technology
- **Active-matrix display**, also known as a TFT (thin-film transistor) display, uses a separate transistor to apply changes to each liquid crystal cell and thus display high-quality color that is viewable from all angles

LCD (Liquid Crystal Display)

- **Passive-matrix display** uses **fewer transistors** and requires less power than an active-matrix display
- The color on a passive-matrix display often is **not as bright** as an active-matrix display
- Users view images on a passive-matrix display best when working directly in front of it
- Passive-matrix displays are less expensive than active-matrix displays

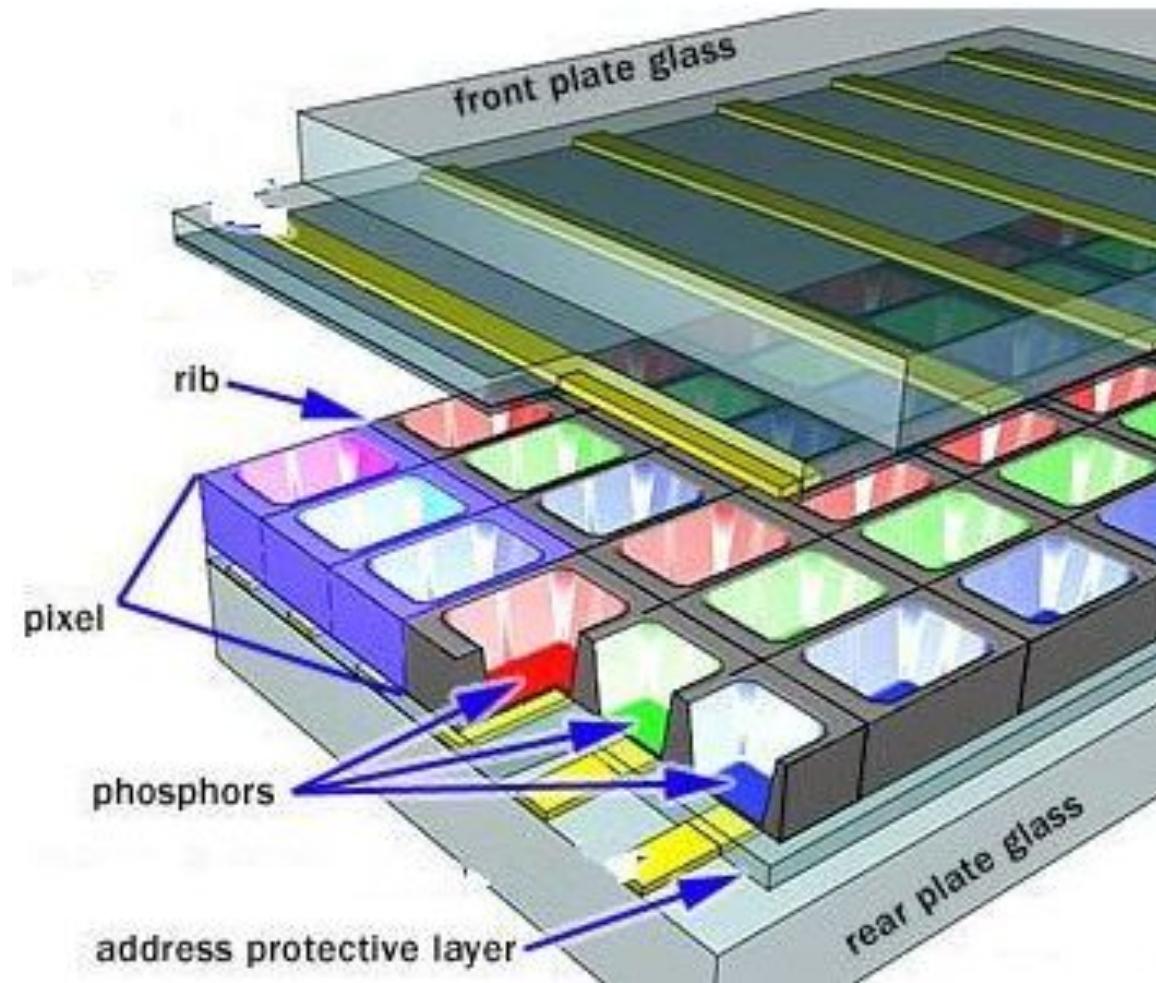
LCD (Liquid Crystal Display)

- An important measure of LCD monitors is the **response time**, which is the time in millisecond (ms) that it takes to turn a pixel on or off
- LCD monitors' response times average 25 ms
- The lower the number, the faster the response time
- **Resolution** and **dot pitch** determines quality of LCD monitor

Gas plasma monitor

- A flat-panel display that uses **gas plasma technology**
- A layer of **gas** between two sheets of material
- When voltage is applied, the **gas releases ultraviolet (UV) light** that causes the pixels on the screen to glow and form an image
- Larger screen sizes and higher display quality than LCD, but much more expensive

Gas plasma monitor



ADVANTAGES:

1. Refreshing is not required.
2. Produce a very steady image free of Flicker.
3. Less bulky than a CRT.

DISADVANTAGES:

1. Poor resolution of up to 60 d.p.i.
2. It requires complex addressing and wiring.
3. It is costlier than CRT.

Hard Copy Devices

Printers

Printed output is referred to as **hard copy** and do not require electric power as they are printed on papers to read after printing and provide permanent readable form information

According to how they print printers can be of different types:

Character printers prints one character of a text at a time

Line printer prints one line of the text at a time

A **page printer** prints one page of the text at a time

According to the technology used printers produce output by either **impact** or **non impact** methods

Impact printers

Impact printers **press the formed character faces** against an inked ribbon onto paper

Individual characters or graphics patterns are obtained by retracting certain pins so that the remaining pins form the pattern to be printed.

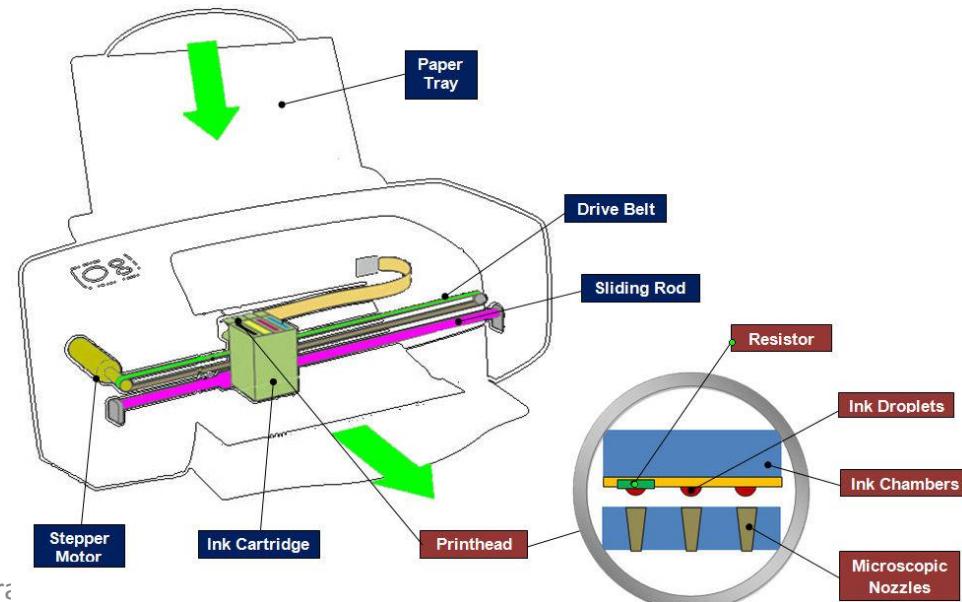
Non-Impact Printers

Non impact printers **use laser techniques, ink-jet sprays** etc to get images onto paper.

Ink-jet Devices

Ink-jet methods produce output by **squirting ink** in horizontal rows across a roll of paper wrapped on a drum.

When a heater is activated a drop of ink is exploded onto the paper

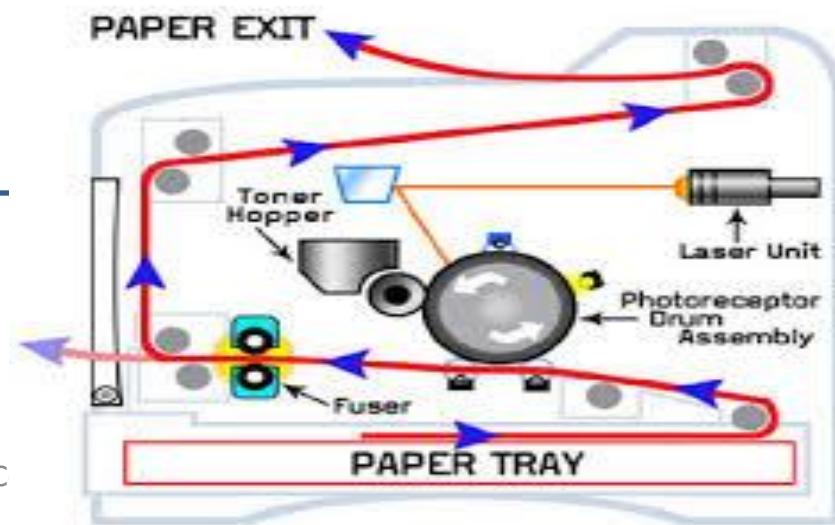


Laser Devices

These are **page printers**

They use **laser beam** to produce an image of the page containing text graphics on a photosensitive drum which is coated with negatively charged photo conductive material

In a laser device a laser beam creates a charge distribution on a rotating drum coated with a photo electric material such as selenium. Toner is applied to drum and then transferred to paper.



Potters

Plotter is a device that draws pictures on paper based on commands from a computer

They are used to produce precise and good quality graphics and drawing under computers control

They use motor driven ink pen or ink jet to draw graphic or drawings

Drawings can be prepared on paper, Velluym or Mylar (Polyester film)

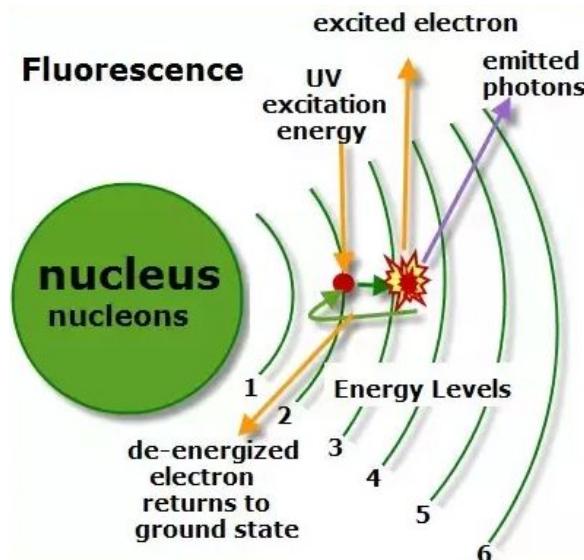


Display Devices

Fluorescence / Phosphorescence

A phosphors fluorescence is the light emitted as the very unstable electrons lose their excess energy whole the phosphor is being struck by electrons

Phosphorescence is the light given off by the return of the relatively more stable excited electrons to their unexcited state once the electron beam excitation is removed



Display Devices



Persistence

A phosphor's persistence is defined as the time from the removal of excitation to the moment when phosphorescence has decay to 10 percent of the initial light output

The range of persistence of different phosphors can reach many seconds

The phosphors used for graphics display devices usually have persistence of 10 to 60 micro seconds

A phosphor with low persistence is useful for animation and a high persistence phosphor is useful to highly complex static pictures

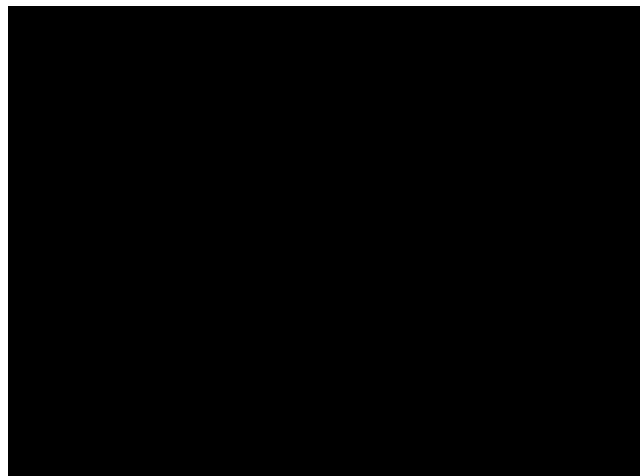
Display Devices

Refresh rate

The refresh rate is the number of times per second the image is redrawn to give a feeling of un-flickering pictures and it is usually 50 per second

As the refresh rate decreases flicker develops because the eye can no longer integrate the individual light impulses coming from a pixel

The refresh rate above which a picture stops flickering and fuses into a steady image is called the critical fusion frequency (CFF)



Display Devices

The factors affecting the CFF are:

- i. Persistence: longer the persistence the lower the CFF But the relation between the CFF and persistence is non linear
- ii. Image intensity: Increasing the image intensity increases the CFF with non linear relationship
- iii. Ambient room light Decreasing the ambient room light increases the CFF with nonlinear relationship
- iv. Wave lengths of emitted light
- v. Observer

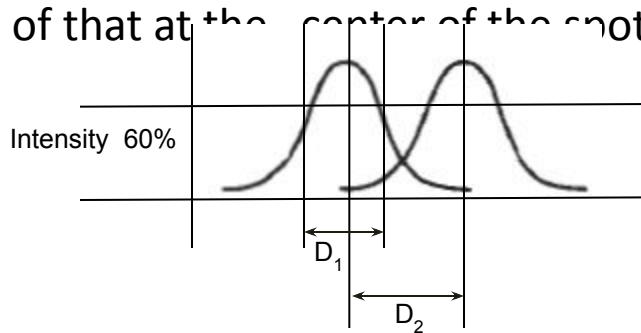
Display Devices

Resolution

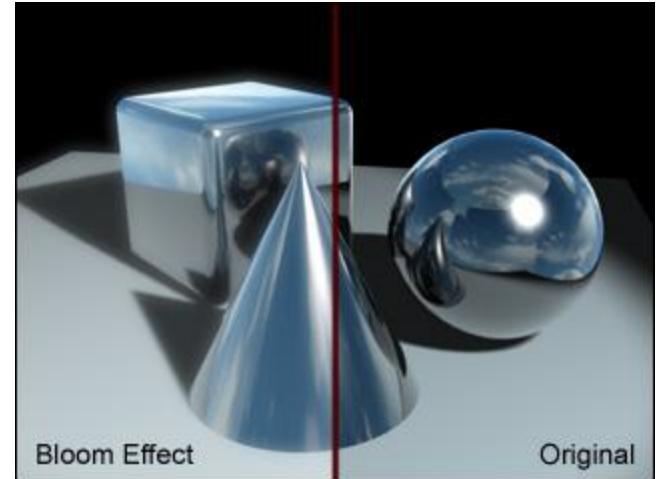
Resolution is defined as the maximum number of points that can be displayed horizontally and vertically without overlap on a display device. Factors affecting the resolution are as follows.

i. Spot profile:

The spot intensity has a Gaussian distribution. So two adjacent spots on the display device appear distinct as long as their separation D_2 is greater than the diameter of the spot D_1 at which each spot has an intensity of about 60 percent of that at the center of the spot.



Display Devices



ii. Intensity:

As the intensity of the electron beam increases the spot size on the display tends to increase because of spreading of energy beyond the point of bombardment

This phenomenon is called *blooming*. Consequently, the resolution decreases.

Thus it is noted that resolution is not necessarily a constant and it is not necessarily equal to the resolution of a pix-map, which is allocated in a buffer memory

Display Devices

Color CRTs

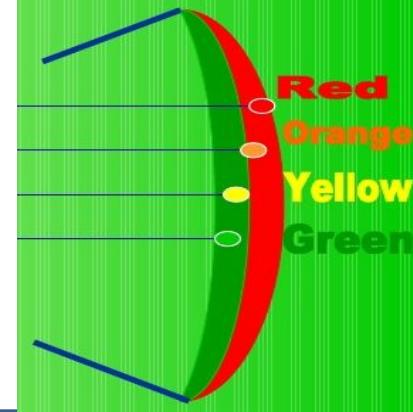
Color depends on the light emitted by phosphor.

Two type:

Beam Penetration Method

Shadow Mask Method

Display Devices



i. Beam Penetration Method:

Two different layers of phosphor coating used Red (outer) and Green (inner)

Display of color depends on the depth of penetration of the electron beam into the phosphor layers

- A beam of slow electrons excites only the outer red layer
- A beam of very fast electrons penetrates thru the red phosphor and excites the inner green layer
- When quantity of red is more than green then color appears as orange
- When quantity of green is more than red then color appears as yellow

Screen color is controlled by the beam acceleration voltage.

Only four colors possible, poor picture quality

Display Devices

ii. Shadow Mask Method

The inner side of the viewing surface of a color CRT consists of closely spaced groups of red, green and blue phosphor dots.

Each group is called a *triad*

A thin metal plate perforated with many small holes is mounted close to the inner side of the viewing surface. This plate is called *shadow mask*

The shadow mask is mounted in such a way that each hole is correctly aligned with a triad in color CRT

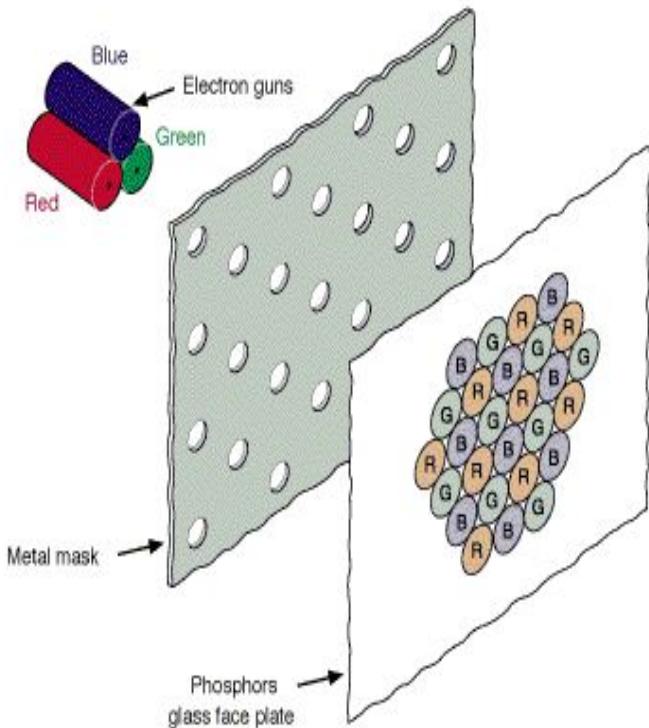
Display Devices

There are three electron guns one for each dot in a triad

The electron beam from each gun therefore hits only the corresponding dot of a triad as the three electron beams deflect

A triad is so small that light emanating from the individual dots is perceived by the viewer as a mixture of the three colors

Thus, a wide range of colors can be produced by each triad depending on how strongly each individual phosphor dot in a triad is excited.



Display Devices

a. A Delta –Delta CRT

A triad has a *triangular (delta) pattern* as are the three electron guns

Main drawback of this type of CRT is that a high precision display is very difficult to achieve because of technical difficulties involved in the alignment of shadow mask holes and the triad on one to one basis

b. Precision Inline CRT

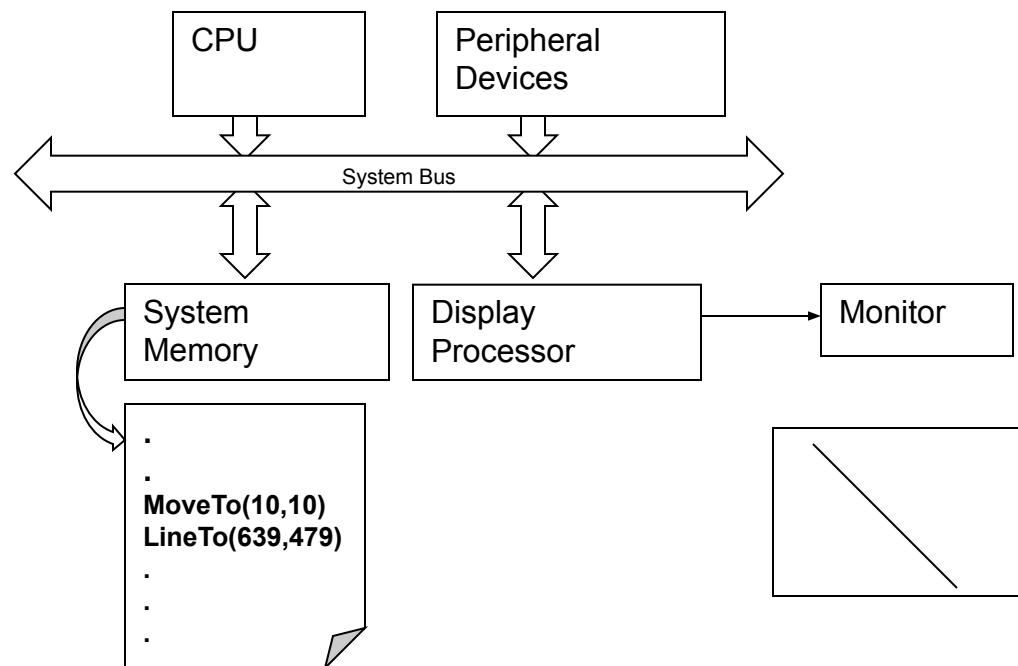
A triad has an *in-line pattern* as are the three electron guns

The introduction of this type of CRT has eliminated the main drawback of a Delta-Delta CRT

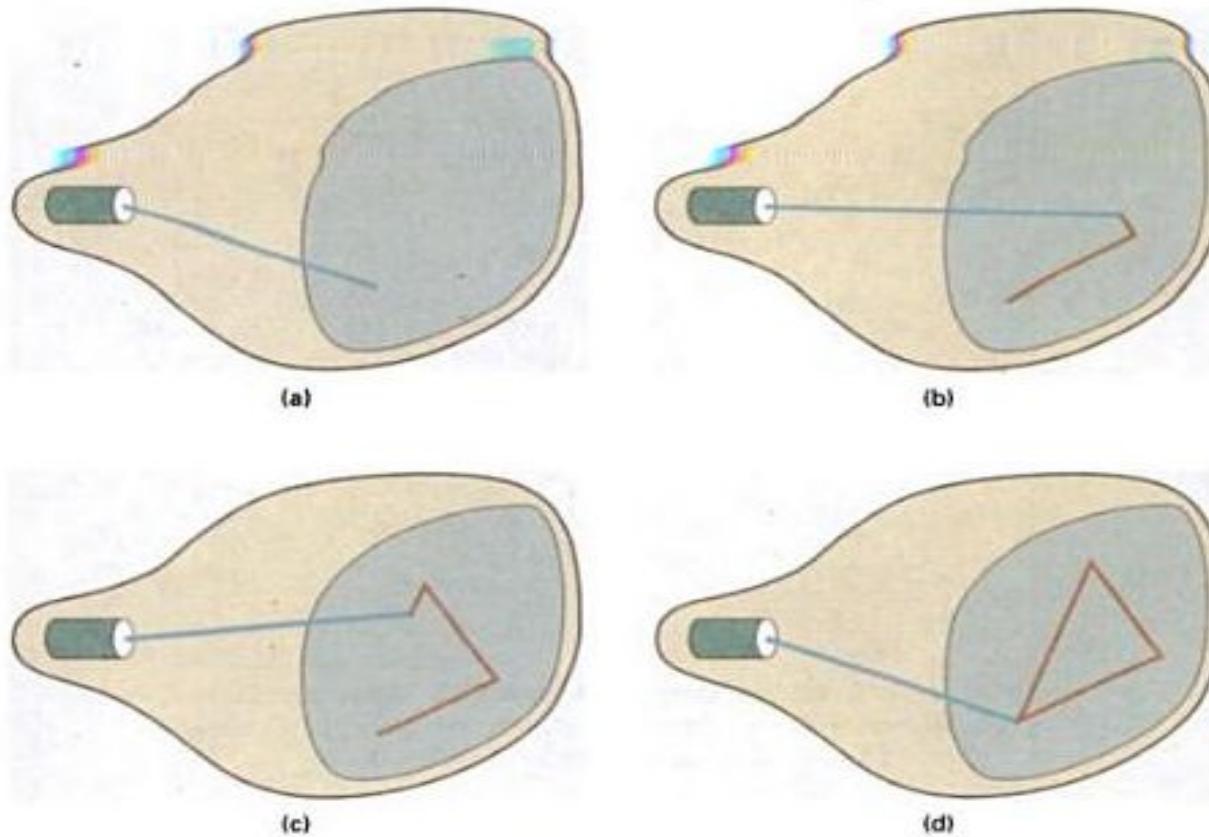
But a slight reduction of image sharpness at the edges of the tube has been noticed

Normally 1000 scan lines can be achieved

i. Vector Display Technology



i. Vector Display Technology



i. Vector Display Technology

It is also called **random scan, a stroke, a line drawing or calligraphic display**

Advantages:

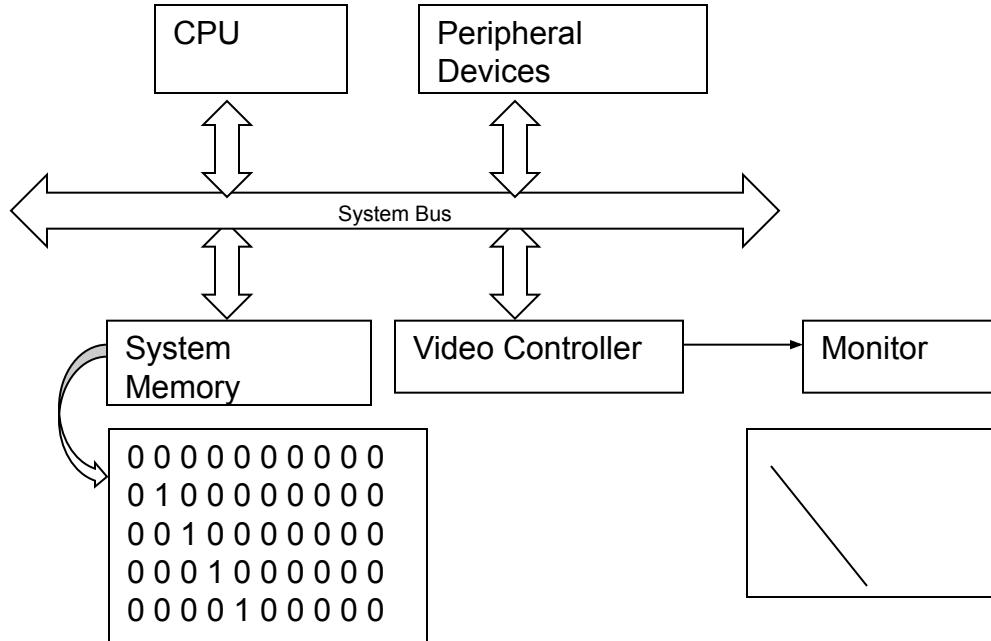
- i. It can produce a **smooth output primitives** with higher resolution unlike the raster display technology
- ii. It is better than raster display for real time dynamics such as **animation**
- iii. For transformation, only the end points has to be moved to the new position in vector display but in raster display it is necessary to move those end points and at the same time all the pixels between the end points must be scan converted using appropriate algorithm
No prior information on pixels can be reused

i. Vector Display Technology

Disadvantages:

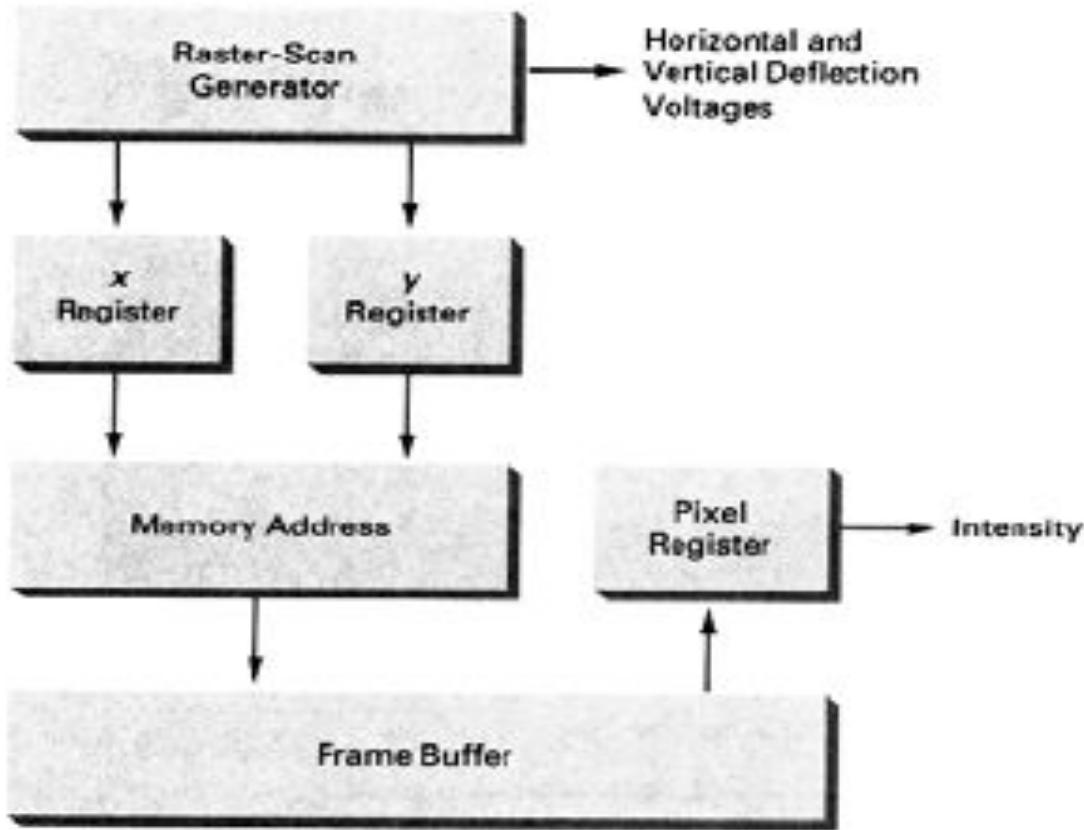
- i. A vector display **can not fill areas** with patterns and manipulate bits
- ii. Time required for refreshing an image depends upon its complexity (**more the lines, longer the time**) the flicker may therefore appear as the complexity of the image increases. The fastest vector display can draw about 100000 short vectors in a refresh cycle without flickering

ii. Raster Display Technology



ii. Raster Display Technology

Raster-Scan: Video Controller



basic video-
controller
refresh
operations

ii. Raster Display Technology

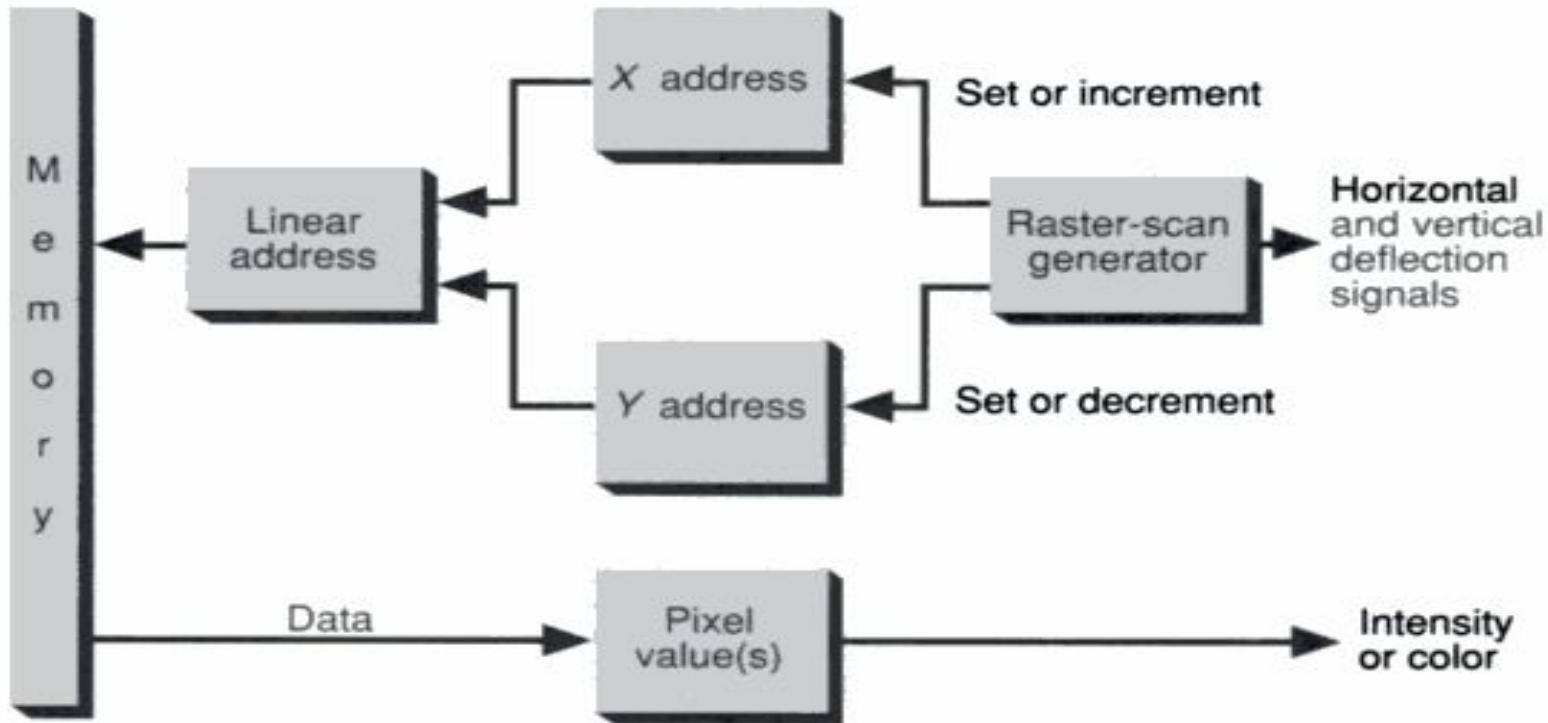


Fig. 4.20 Logical organization of the video controller.

ii. Raster Display Technology

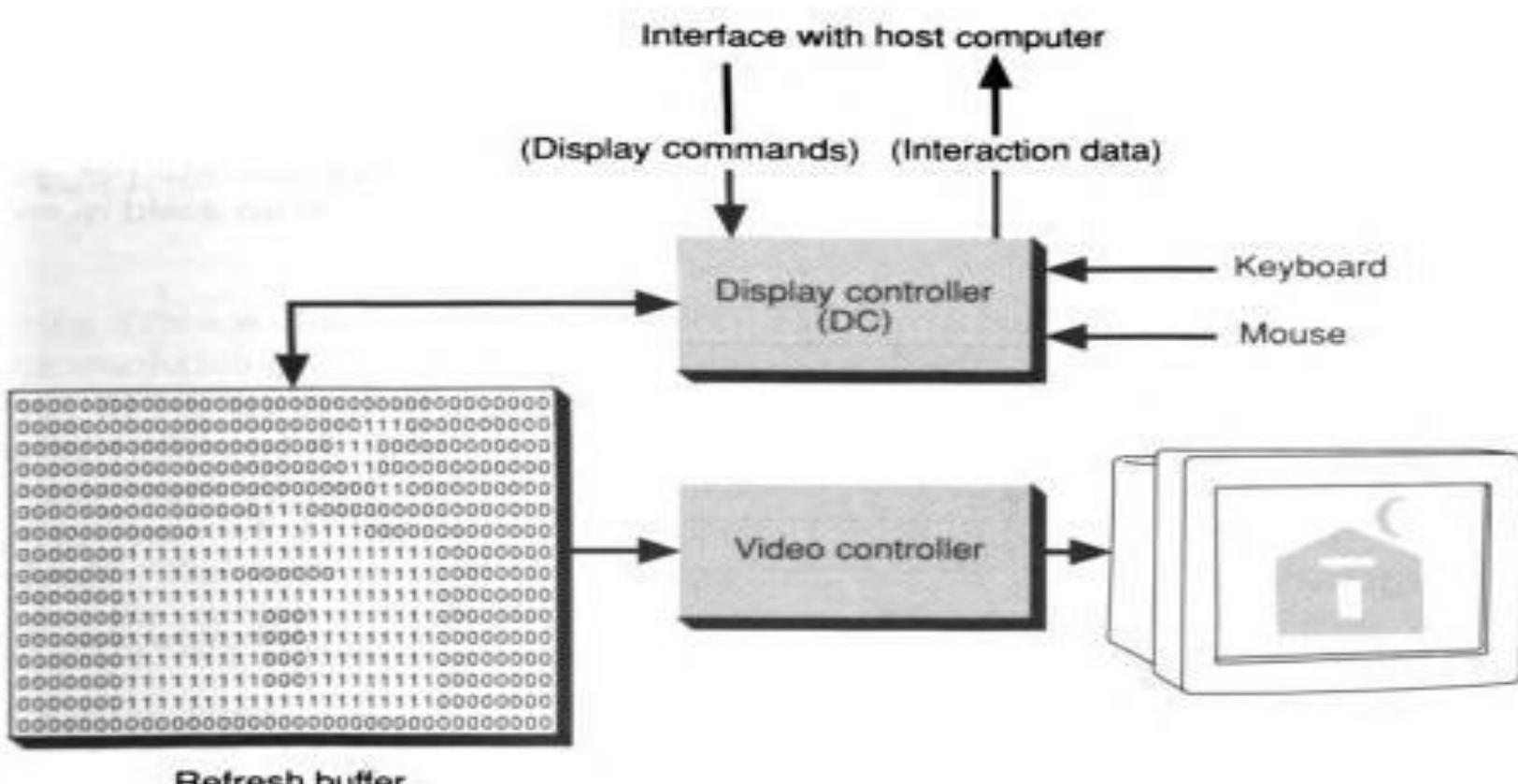
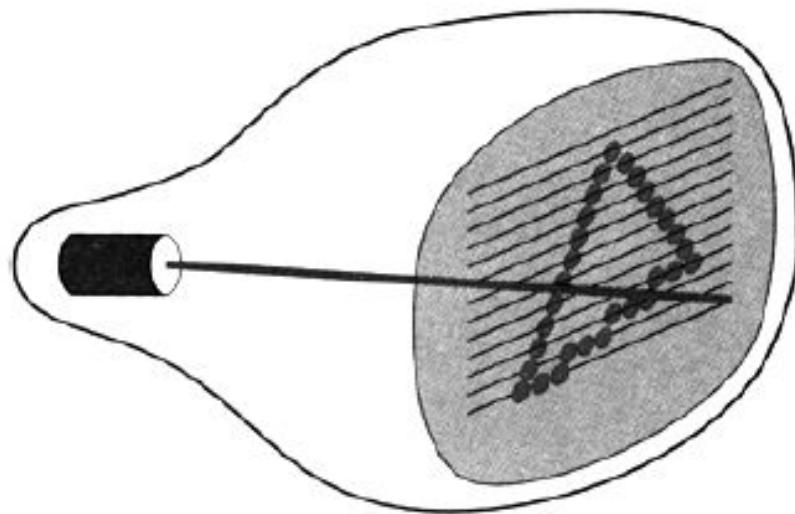


Fig. 1.2 Architecture of a raster display.

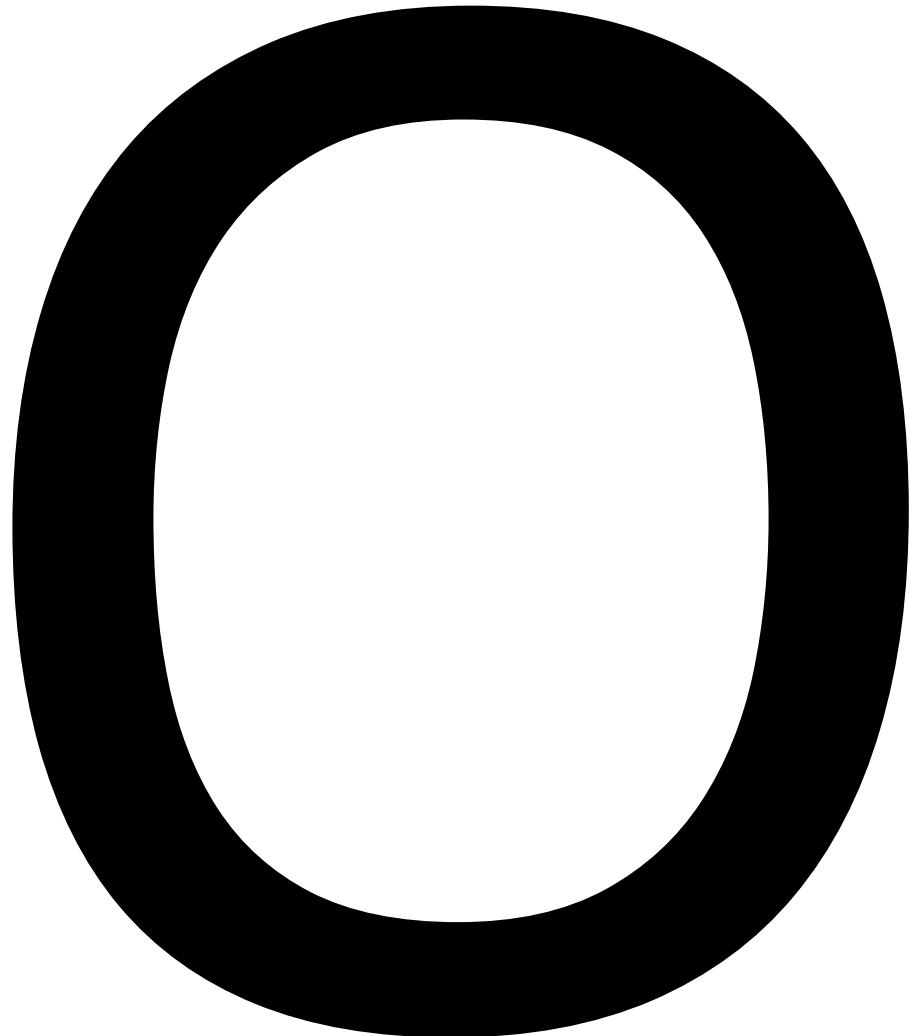
ii. Raster Display Technology



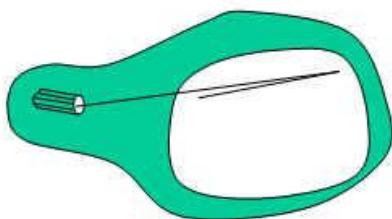
ii. Raster Display Technology

Disadvantages

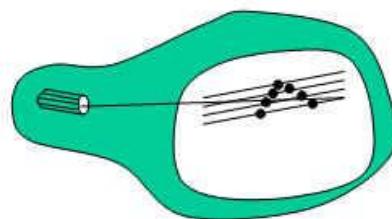
- i. For Real-Time dynamics not only the end points are required to move but all the pixels in between the moved end points have to be scan converted with appropriate algorithms which might slow down the dynamic process
- ii. Due to scan conversion “**jaggies**” or “**stair-casing**” are unavoidable



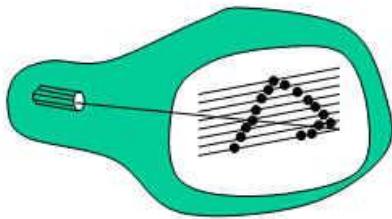
RDT/VDT



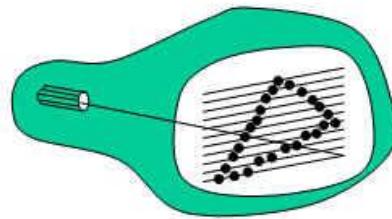
(a)



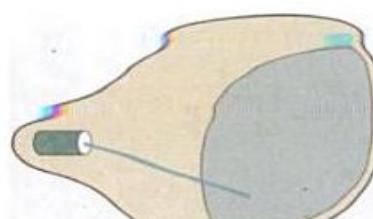
(b)



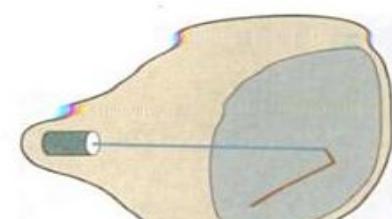
(c)



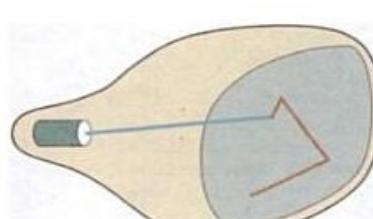
(d)



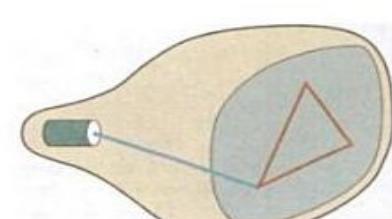
(a)



(b)

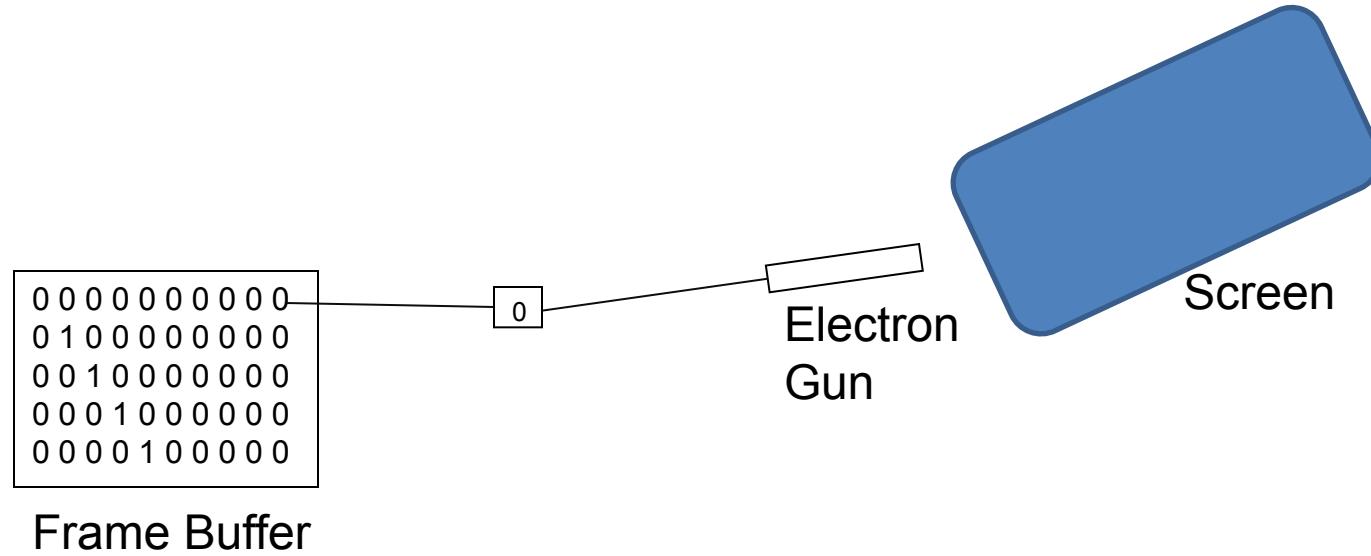


(c)

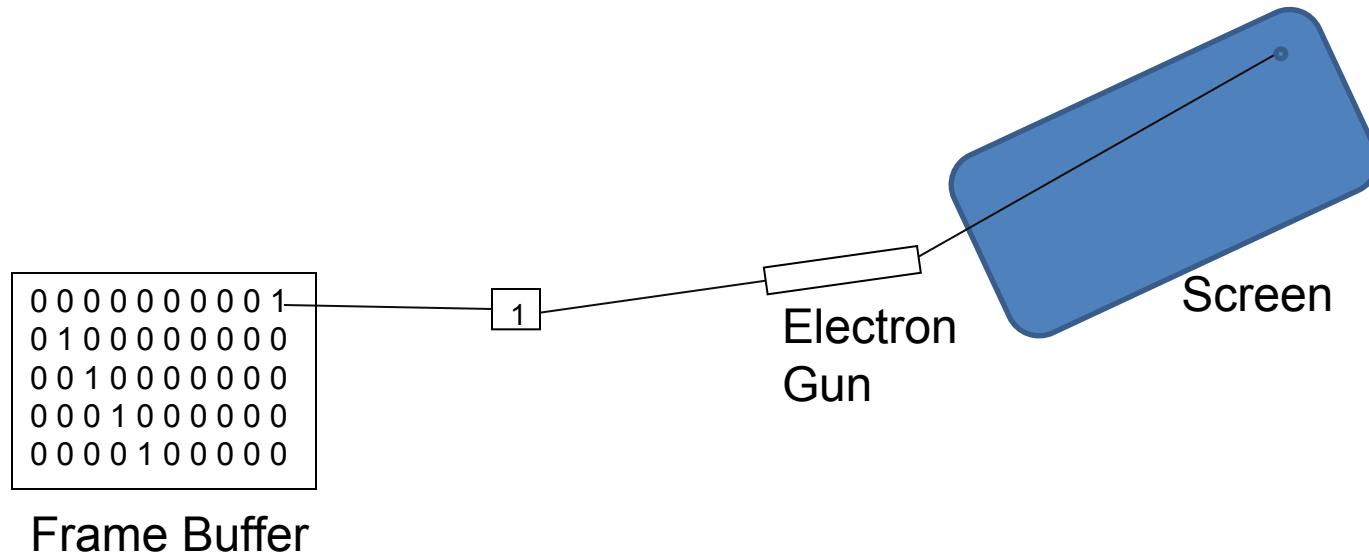


(d)

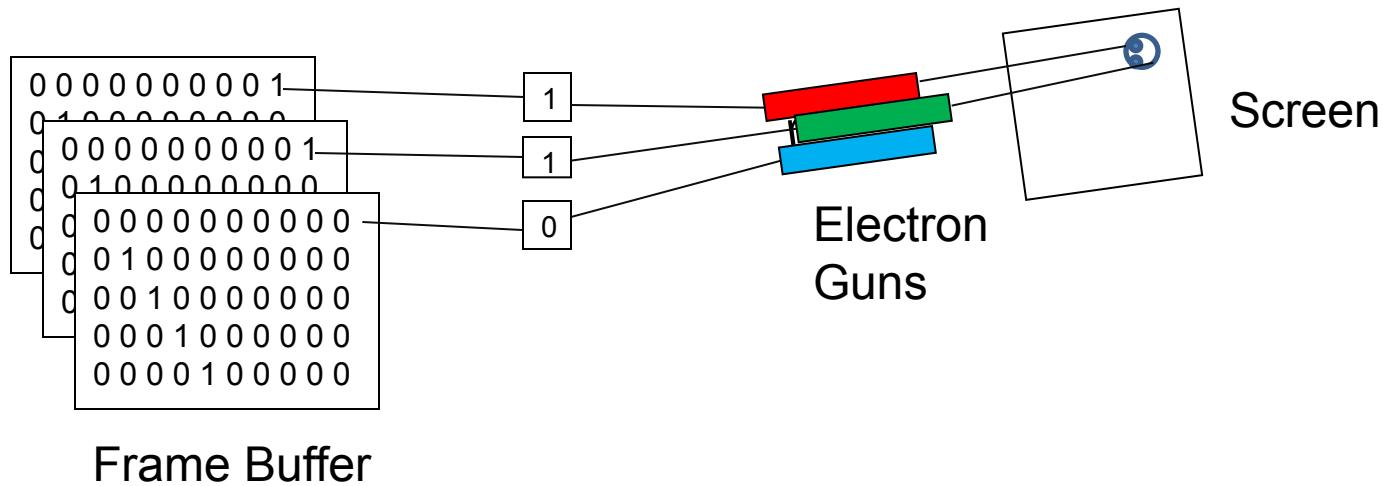
Frame Buffer



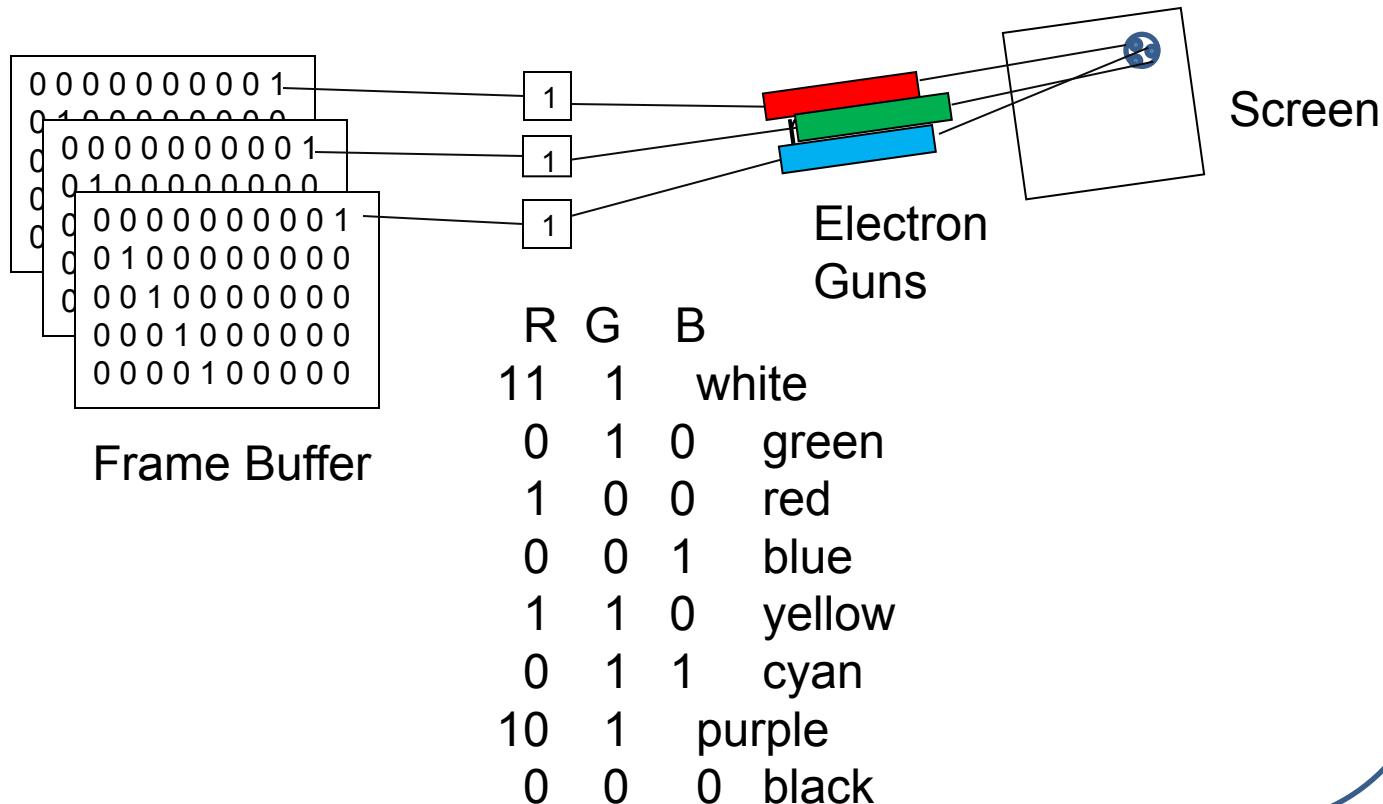
Frame Buffer



Frame Buffer



Frame Buffer

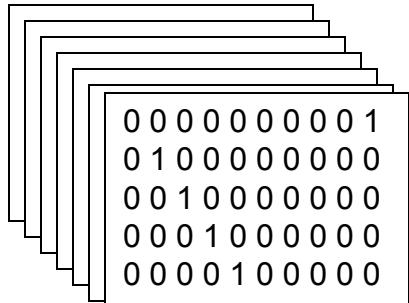


Frame Buffer

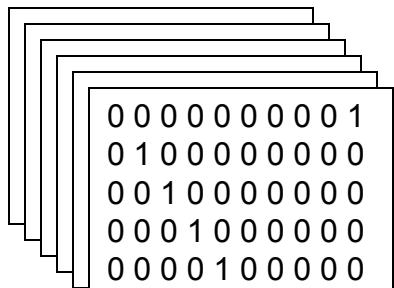
Total number of intensities achievable
out of a single pixel on the screen

$$\text{Total number of intensities achievable} = 2^n$$

n = number of bits assigned to a single pixel



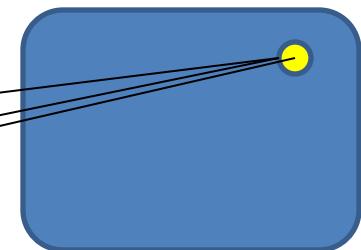
11111111



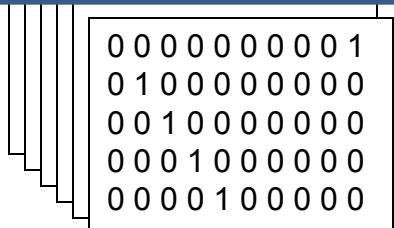
11111111

Electron
Guns

Screen



If n bits are assigned to a single pixel and the resolution of the screen is 1024×800 then the total size of the required frame buffer is $n \times 1024 \times 800$



11111111

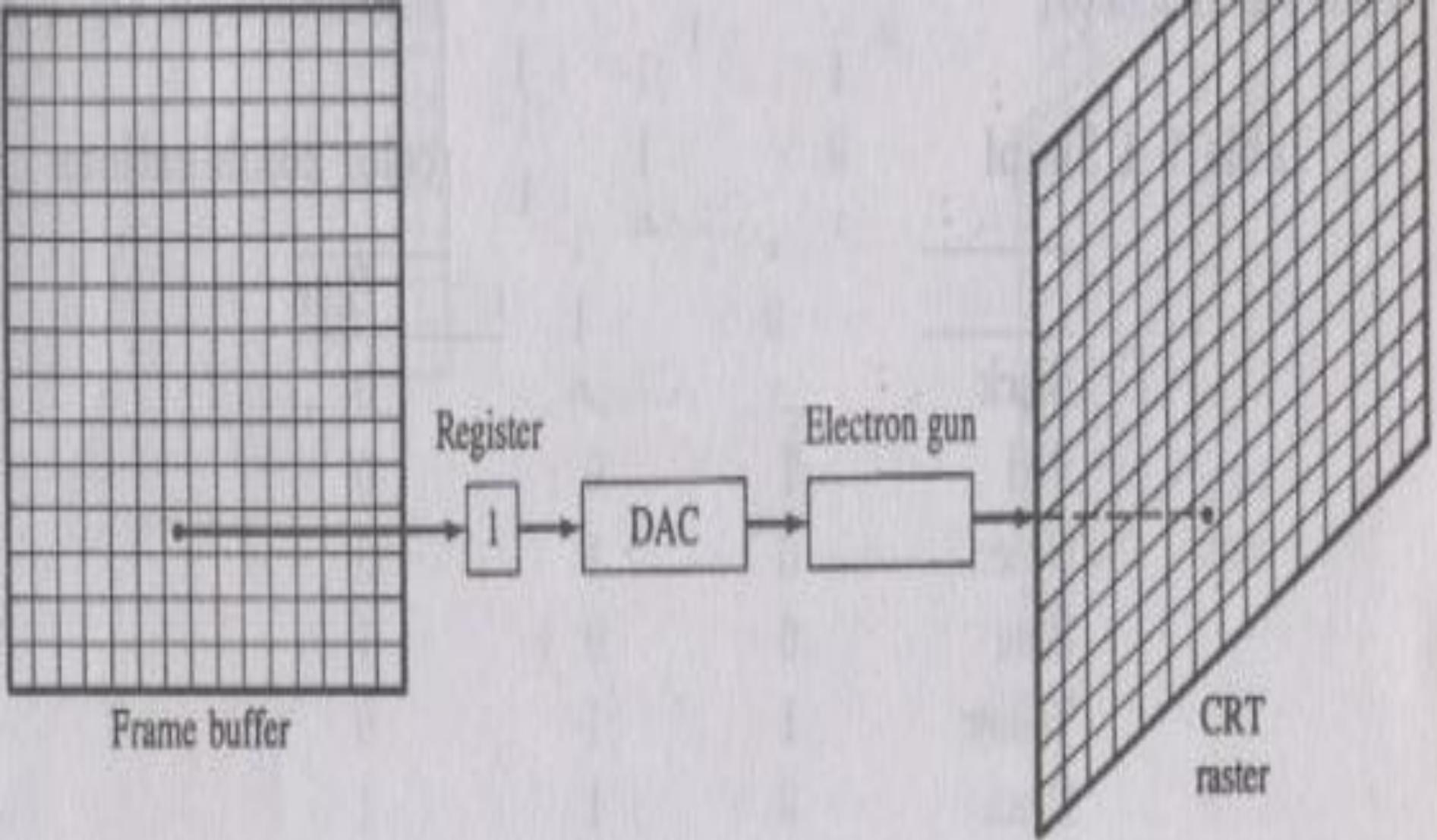
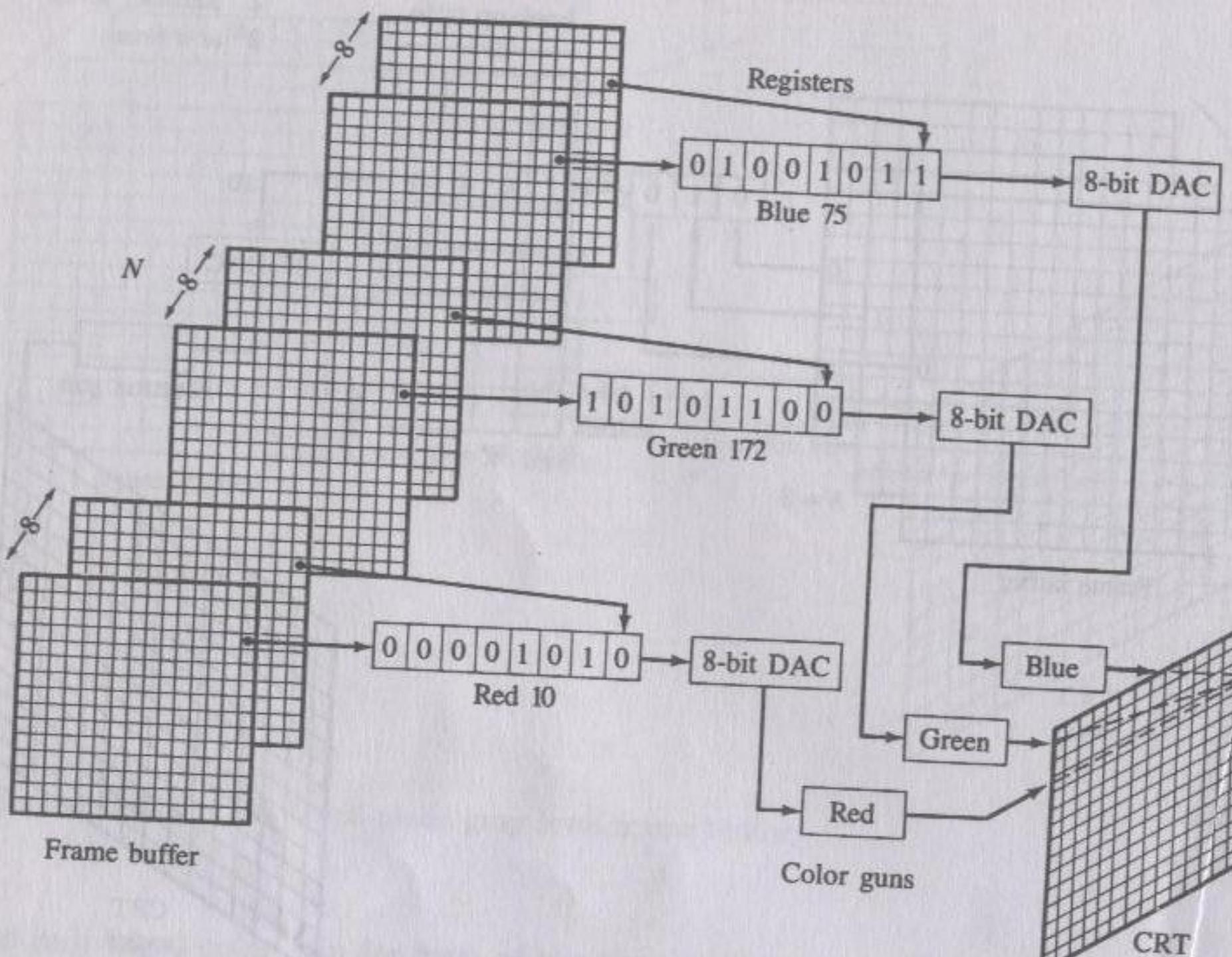
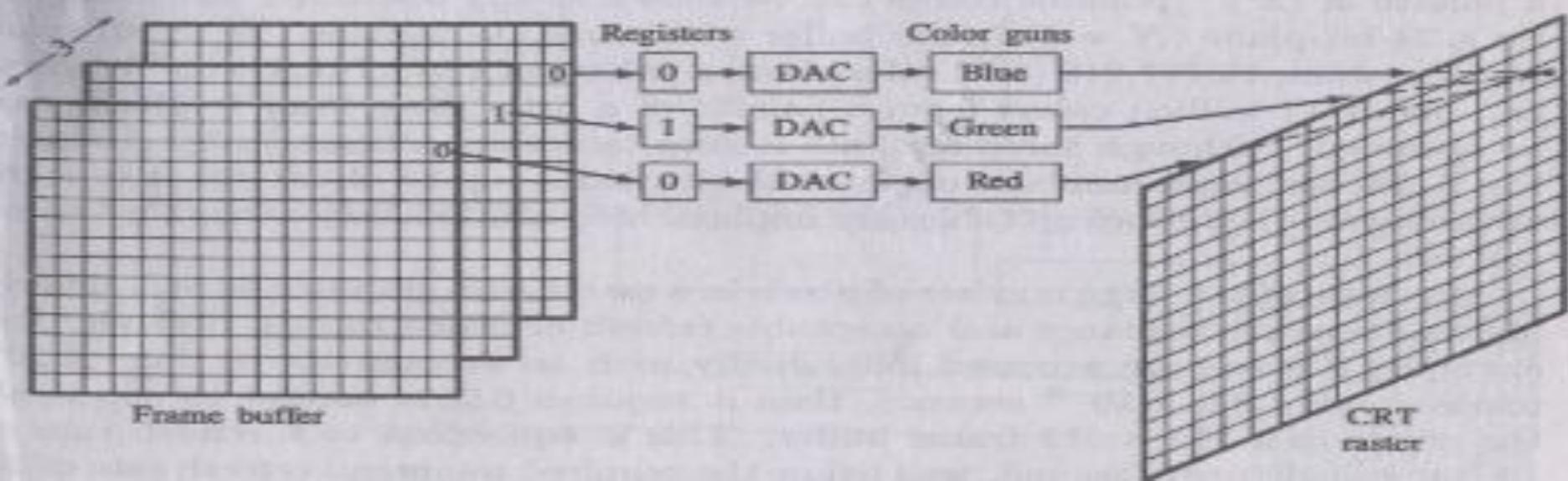
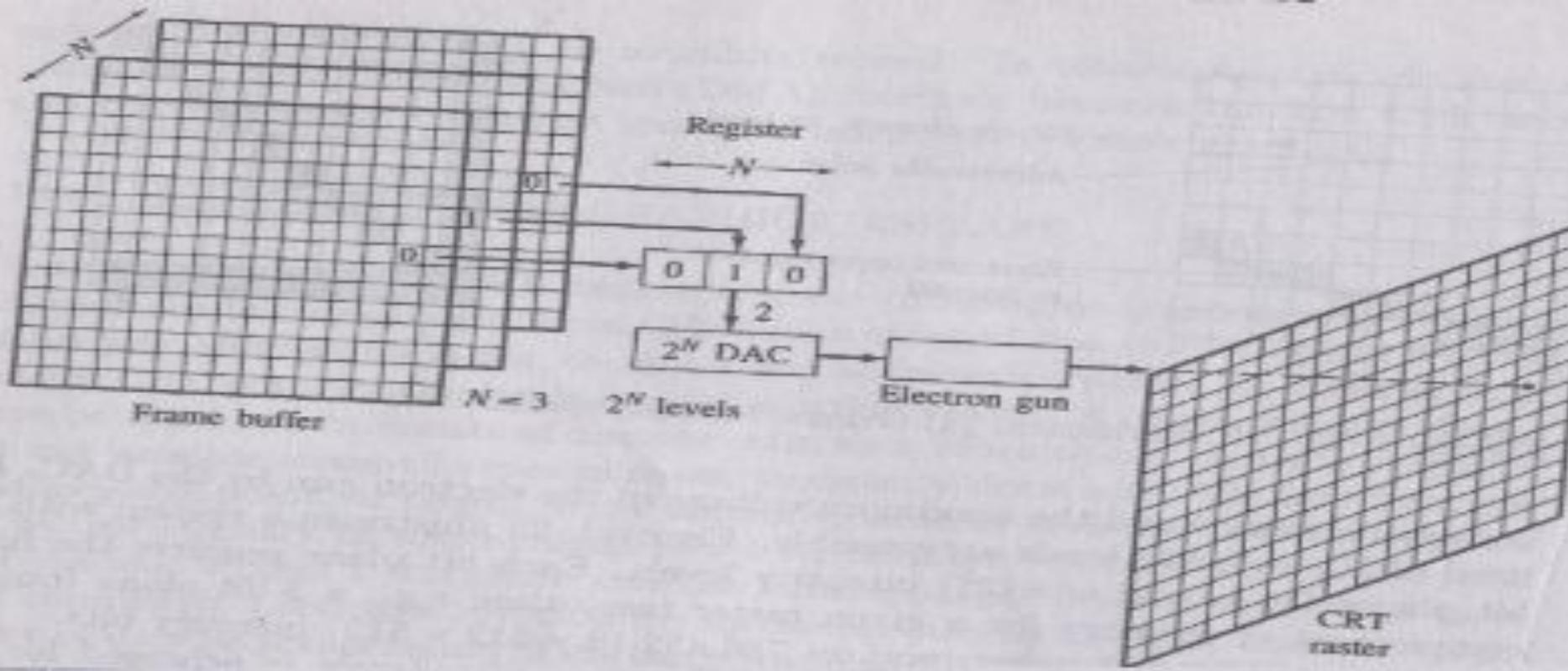
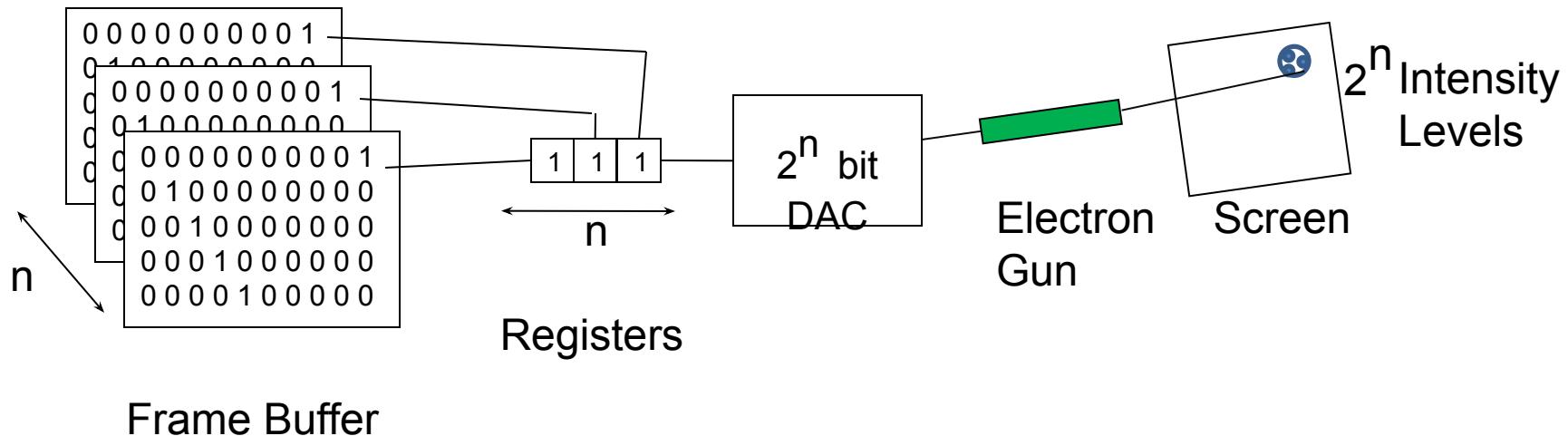


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.







The maximum number of intensity levels achievable from a single pixel on the screen



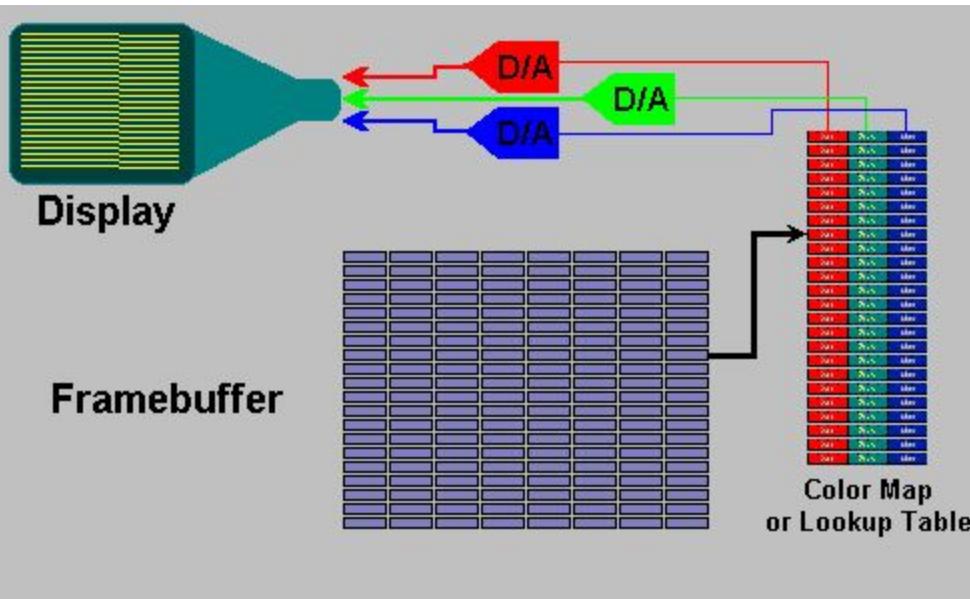
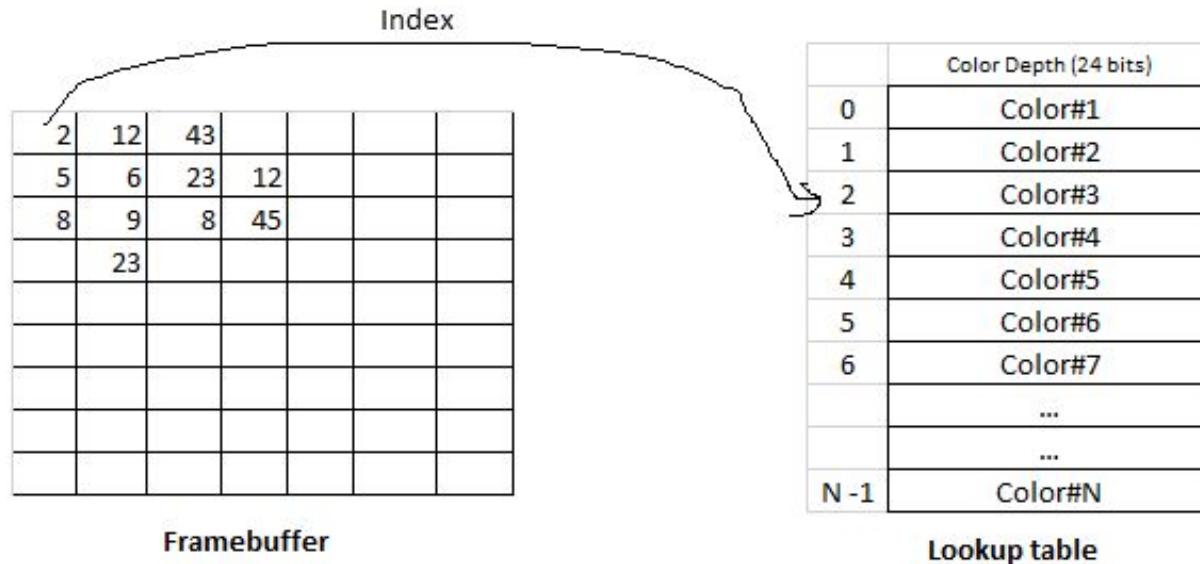
$$2^n$$

where n is the number of bits used to represent that single pixel

Video Cards and CRT Monitors

- Many CRT monitors use an analog signal to produce an image
- **Video card** converts digital output from the computer into an analog video signal and sends the signal through a cable to the monitor
 - Also called a **graphics card**

Look up Tables



The diagram illustrates the mapping of image pixels to colors. On the left, a 4x3 grid of pixels is shown with an 8-bit index number below it. The grid contains the following data:

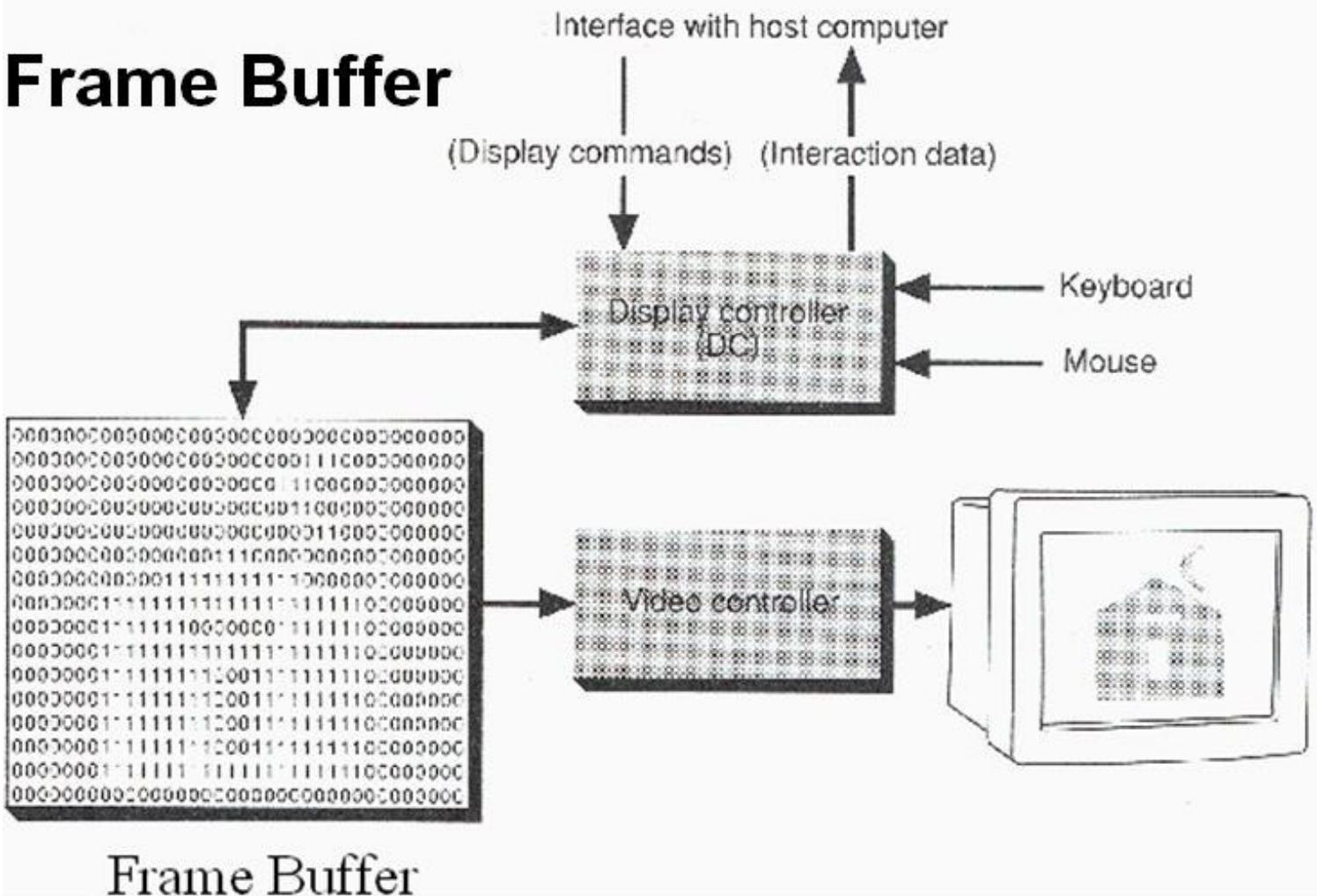
2	148	99
112	112	3
112	149	67
98	4	
254		

The index number 4 is highlighted in yellow. A blue arrow points from the index number 4 to the corresponding row in the color lookup table below. The color lookup table is titled "RGB Values Three 8-Bit Numbers" and lists 256 entries. The entry for index 4 is highlighted in red:

0	12, 116, 0
1	255, 0, 20
2	120, 10, 15
3	43, 201, 101
4	155, 22, 233
251	112, 18, 23
252	54, 122, 0
253	87, 110, 115
254	2, 10, 254
255	90, 222, 32

A computer monitor icon on the right displays a small red dot at its center, representing the color corresponding to index 4.

Frame Buffer



Video Cards and CRT Monitors

- The number of colors a video card displays is determined by its **bit depth**
- The video card's bit depth, also called the **color depth**, is the number of bits it uses to store information about each pixel
- i.e. 8-bit video card uses 8 bits to store information about each pixel; this video card can display 256 colors ($2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$)
- i.e. 24-bit video card uses 24 bits to store information about each pixel and can display 16.7 million colors
- The greater the number of bits, the better the resulting image

Video Cards and CRT Monitors



Video Display Standards Video Electronics Standards Association (VESA), which consists of video card and monitor manufacturers, develops video standards to define the resolution, number of colors, and other display properties.

Monochrome Display Adapter (MDA)

Hercules Graphics Card

Color Graphics Adapter (CGA)

Enhanced Graphics Adapter (EGA)

Video Graphics Adapter (VGA)

Super VGA (SVGA) and Other Standards Beyond VGA

Color Depth	Number of Displayed Colors	Bytes of Storage Per Pixel	Common Name for Color Depth
4-Bit	16	0.5	Standard VGA
8-Bit	256	1.0	256-Color Mode
16-Bit	65,536	2.0	High Color
24-Bit	16,777,216	3.0	True Color

Refresh rate

When electron beam strikes a dot in CRT, the surface of the CRT **only glows for a fraction of a second** and then fades.

Monitor **must redraw** the picture many times per second to avoid having the screen flicker

Refresh rate

The refresh rate is the number of times per second that monitor redraws the images on the screen.

Very few people notice flicker at refresh rates above 72 Hz.

Higher refresh rates are preferred for better comfort in viewing the monitor

If the image dimension is 1366 x 768 then what is the size of the image in terms of Mega Pixels



If the image dimension is 2500 x 3192 then what is the size of the image in terms of Mega Pixels



Quality of a CRT Monitor

An RGB raster system is to be designed using an 8-inch by 10-inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for the frame buffer?

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

How long would it take to load a 640 by 480 frame buffer with 12 bits per pixel, if 10^5 bits can be transferred per second? How long would it take to load a 24-bit per pixel frame buffer with a resolution of 1280 by 1024 using this same transfer rate?

Total number of bits for the frame = $640 \times 480 \times 12$ bits = 3686400 bits

The time needed to load the frame buffer = $3686400 / 10^5$ sec = 36.864 sec

Total number of bits for the frame = $1280 \times 1024 \times 24$ bits = 31457280 bits

The time needed to load the frame buffer = $31457280 / 10^5$ sec = 314.5728 sec

Quality of a CRT Monitor

Assuming that a certain full-color (24-bit per pixel) RGB raster system has a 512-by-512 frame buffer, how many distinct color choices (intensity levels) would we have available? How many different colors could we display at any one time?

Total number of distinct color available is 2^{24}

Total number of colors we could display at one time is 512×512

Assuming that a certain RGB raster system has 512×512 frame buffer with 12 bit per pixel

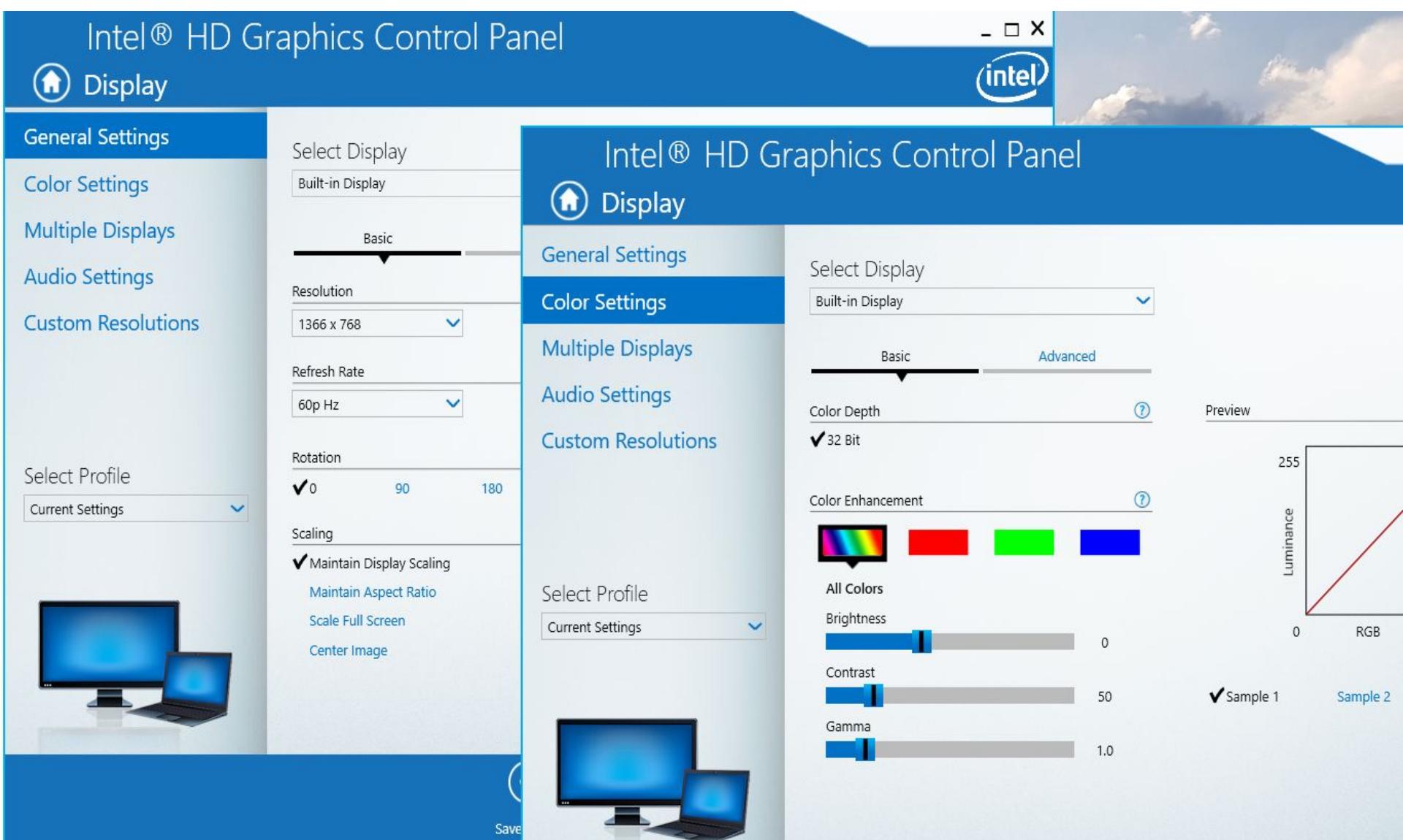
1. How many distinct color choice we have available
2. How many different color could we display at any one time?

Total number of distinct color available is 2^{24}

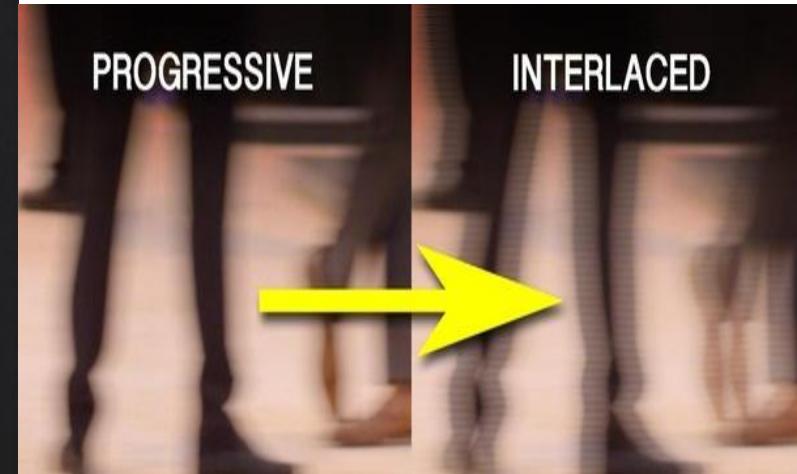
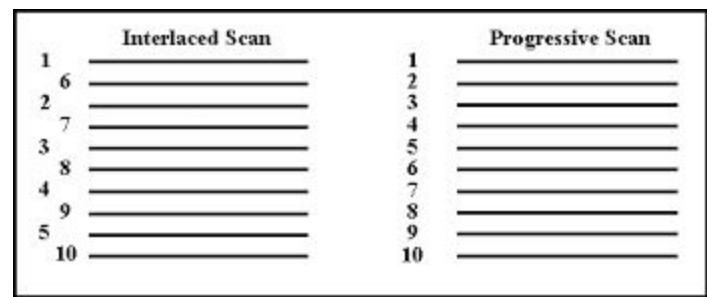
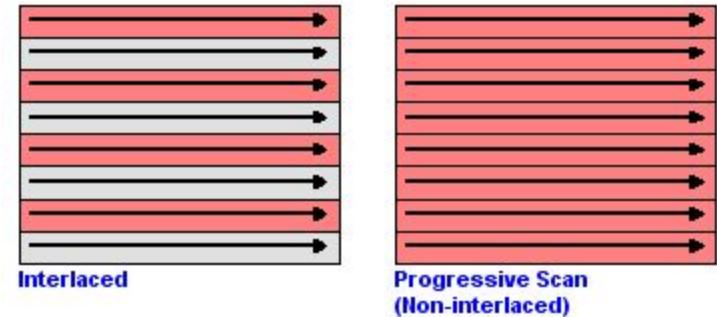
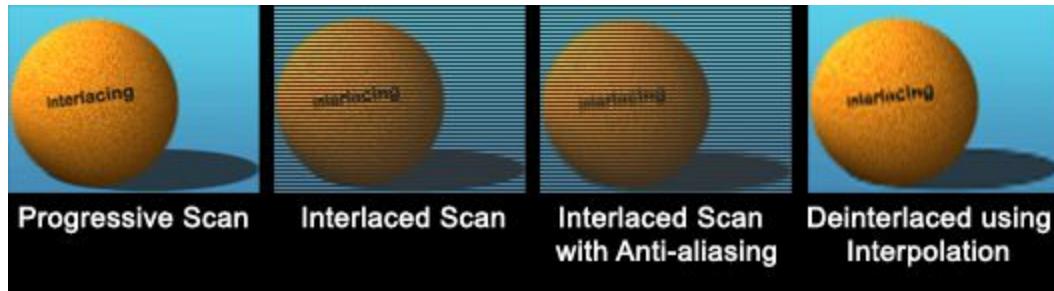
Total number of different color could display at any one time is 2^{12}

The storage spent for frame buffer is $512 \times 512 \times 12$ bit = 3145728 bit

Quality of a CRT Monitor

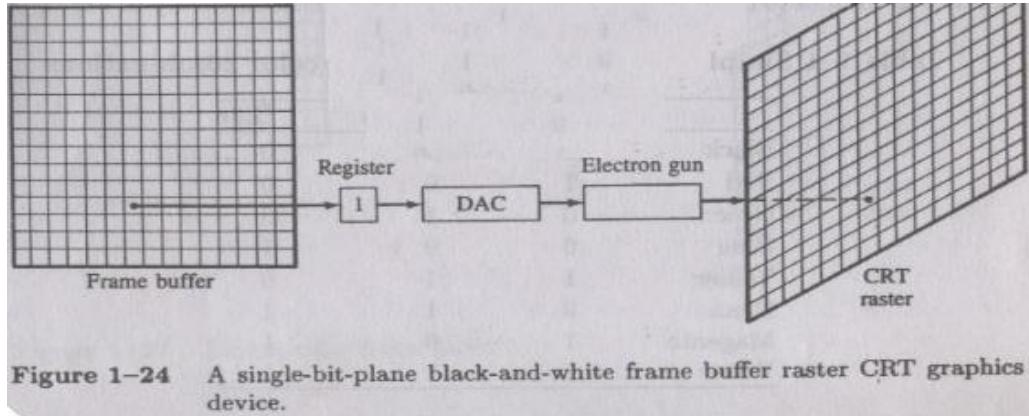


Refresh Rate 60i Hz or 60p Hz



Quality of a CRT Monitor

If on an average it takes 20 nanoseconds to glow a single pixel on the screen and the resolution of the screen is 1024 x 800, will there be a flickering effect seen on the screen?



To glow 1 pixel on screen it takes 20×10^{-9} s

T = To glow all pixels on screen it takes **$1024 \times 800 \times 20 \times 10^{-9}$ s**

$$F = 1 / T = 1 / (1024 \times 800 \times 20 \times 10^{-9})$$

System Refresh rate = 61 frames per sec

Standard refresh rate = 50 frames per sec

There will be no flickering effect seen on the screen as the refresh rate of the screen is 61 fps which is higher than the standard refresh rate!!!

Quality of a CRT Monitor

What is the fraction of total refresh time spent in retrace of electron beam for a non interlaced raster system with a resolution of $m \times n$ and refresh rate of r Hz , the horizontal retrace time is t_h microsecond and vertical retrace time of t_v microsecond?

Here ,

The fraction of total refresh time = $\frac{n \times t_h + t_v}{1/r}$
spent in retrace of electron beam

where t_h is horizontal retrace

t_v is vertical retrace and r is the refresh rate

$n \times m$ = 'm' scan lines and 'n' pixels in each scan line

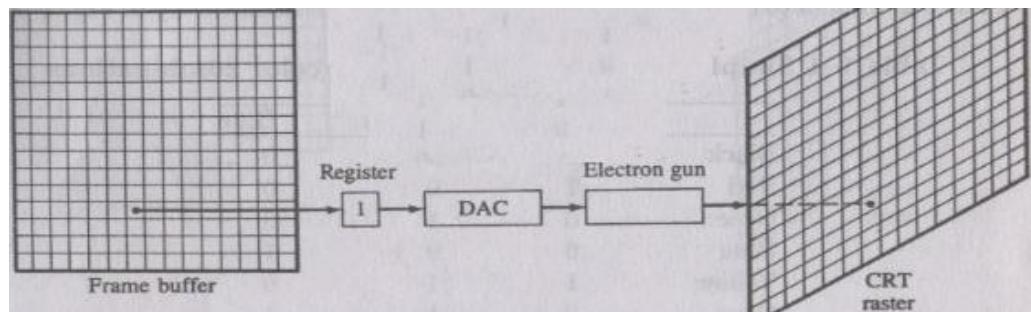


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

Quality of a CRT Monitor

Consider a raster system with resolution of 640 by 480. How many pixels could be accessed per second in this system by a display controller that refreshes the screen at a rate of 60 frames per second? What is the access time per pixel in this system?

The access time per pixel is $1 / (640 \times 480 \times 60)$ sec

The access time per pixel is $1 / (1280 \times 1024 \times 60)$ sec

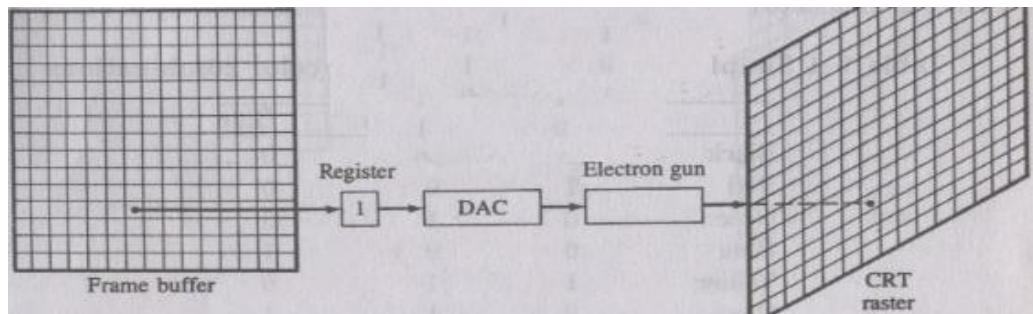
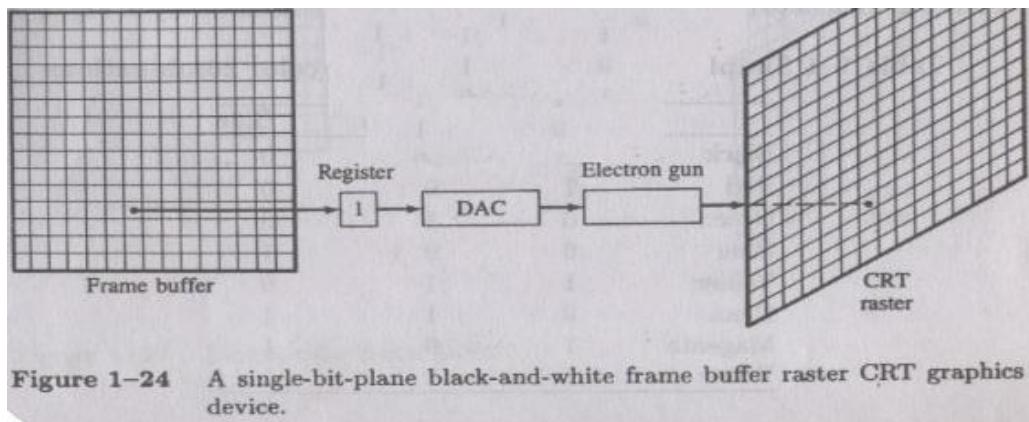


Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

Quality of a CRT Monitor

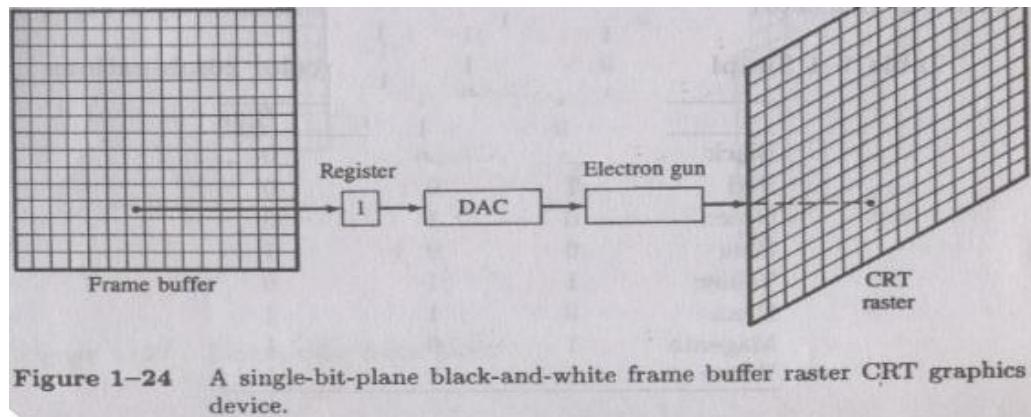
A Raster system can produce a total number of 1024 different levels of intensities from a single pixel of a black and white monitor. If the total resolution of the screen is 640×480 what will be the required size of frame buffer for the display purpose?



Quality of a CRT Monitor

How much time is spent scanning across each row of pixels during screen refresh on a raster system with a resolution of 1280×1024 and refresh rate of 60 frames per second.

Here time required to refresh the screen (t) = $1/60$ seconds



Quality of a CRT Monitor

A Raster system can produce a total number of 1024 different levels of intensities from a single pixel composed of red, green and blue phosphor dots. If the total resolution of the screen is 640×480 what will be the required size of frame buffer for the display purpose?

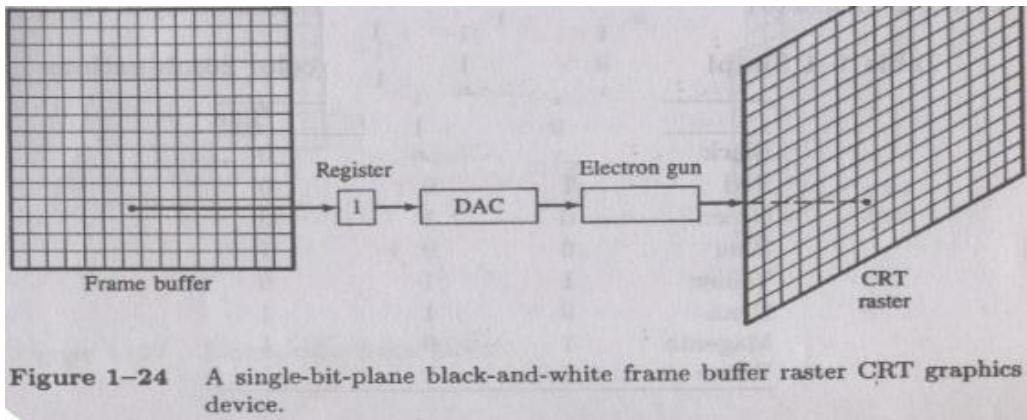
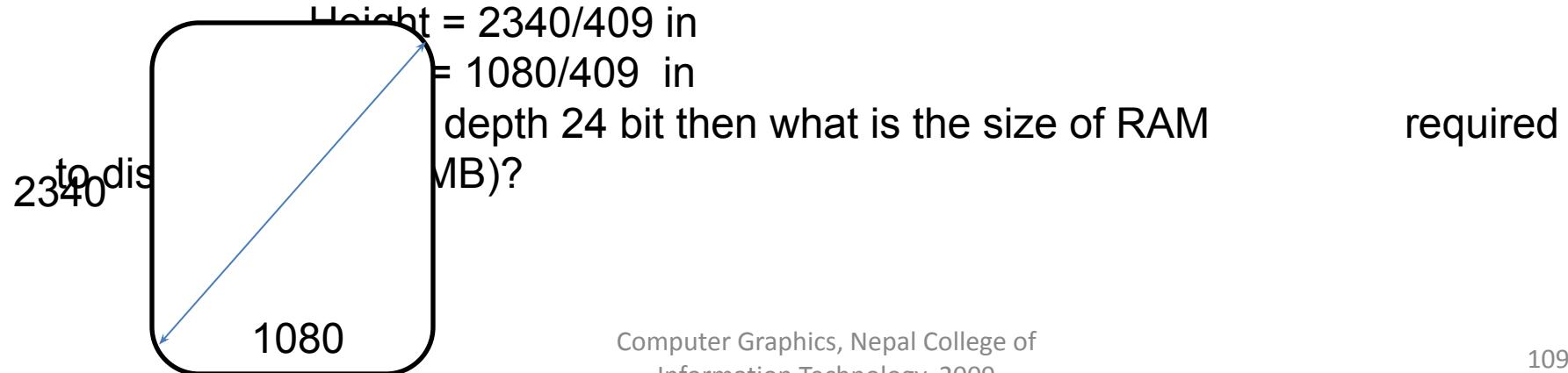


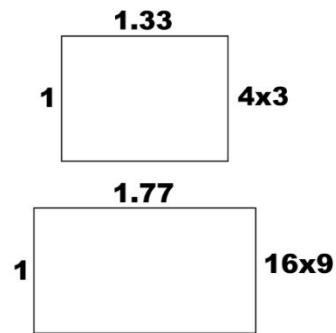
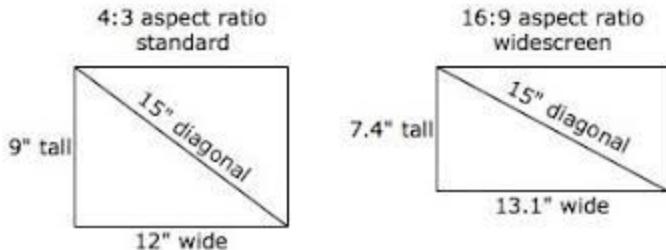
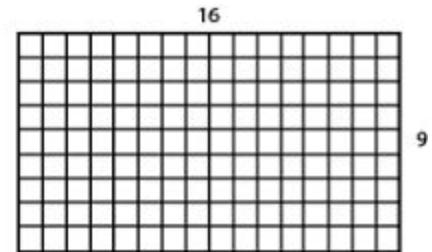
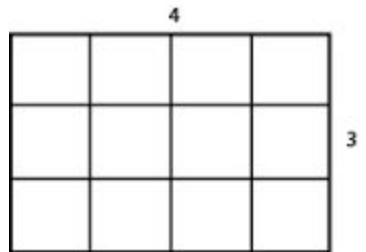
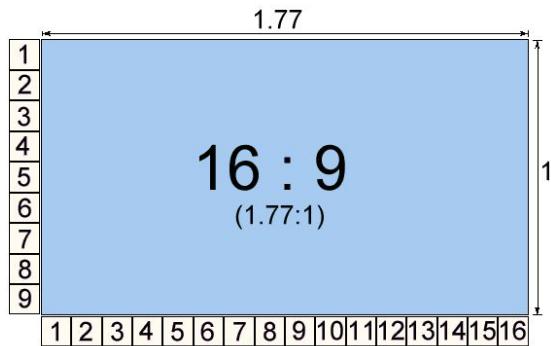
Figure 1-24 A single-bit-plane black-and-white frame buffer raster CRT graphics device.

Your mobile phone has a total resolution of 1080×2340 with 409 PPI. What is the display size of your mobile?



Aspect Ratio

The ratio of vertical points to the horizontal points necessary to produce length of lines in both directions of the screen is called Aspect Ratio.
Usually the aspect ratio is $\frac{3}{4}$.



Aspect Ratio

Aspect Ratio

4:3	1.33 : 1
16:9	1.77 : 1
21:9	2.35 : 1

TVs

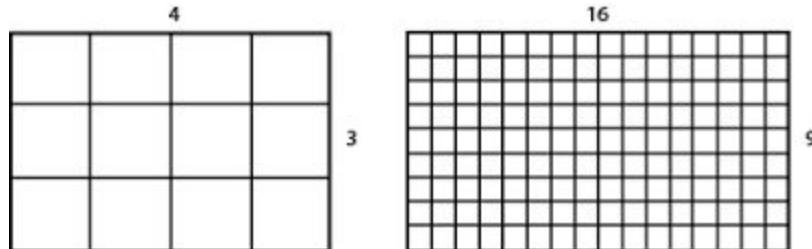
Old TV
HDTVs
Theaters

Aspect Ratio

Suppose we have a video monitor with display area that measures 12 inches across and 9.6 inches high. If resolution is 1280 by 1024 what is the aspect ratio and the diameter of each pixel?

$$\text{Aspect ratio} = w/h \quad \text{i.e.} \quad 1280/12 / 1024/9.6 = 1$$

$$\text{Diameter of each pixel} = 12/1280 \text{ or } 9.6/1024 = 0.0094 \text{ inches}$$



Quality of a CRT Monitor

Depends largely on its **resolution**, **dot pitch**, and **refresh rate**

Resolution describes the **sharpness** and **clearness** of an image

Manufacturers state the resolution of a monitor in pixel

Example: 800 X 600

A pixel (*picture element*) is a single point in an electronic image

A pixel is the smallest element in an electronic image

Line Drawing

Point plotting is accomplished by converting a single coordinate position by an application program into approximate operation for the output device in use.

CRT electron beam is **turned on** to illuminate the screen phosphor at selected location.

Line Drawing

In **random scan system**, point plotting commands are stored in display list and **coordinate values in these instructions are converted to deflection voltages** that position the electron beam at that screen location to be plotted during each refresh cycle.

In case of black and white **raster scan system** a point is plotted by **setting the bit value** corresponding to specified screen position **within *frame buffer*** to **1**.

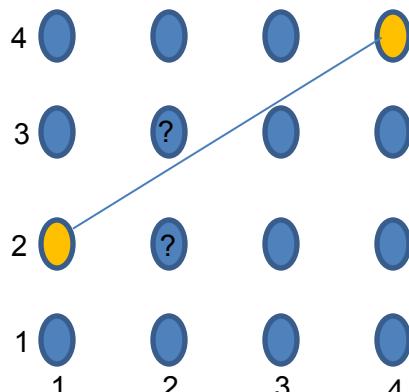
Line Drawing

For drawing lines, we need to calculate **intermediate positions** along the line path between two end points

e.g. 10.45 is rounded off to 10 (this causes **stair cases** or **jaggies** to be formed)

To *load* intensity value into frame buffer at position x, y we use function
setpixel (x, y, intensity)

To *retrieve* current frame buffer intensity value for specified location we use
getpixel(x,y)



DDA (Digital Differential Analyzer)

Slope intercept equation of a straight line is $y = m x + c$

For points (x_1, y_1) and (x_2, y_2) slope $m = y_2 - y_1 / x_2 - x_1$ or $\Delta y / \Delta x$

Algorithms for displaying lines depend on these equations for given interval x we can compute y interval

$$\Delta y = m \cdot \Delta x$$

' x ' interval Δx is obtained by $\Delta x = \Delta y / m$

These equations **form the basis for determining deflection voltages in analog devices**

DDA (Digital Differential Analyzer)

For lines with slope $m = 1$, Δy and Δx are equal

DDA scan conversion algorithm is **based on calculating Δx , Δy**

We **sample line at unit interval** in one coordinate and **determine corresponding integer values** nearest the line path for other coordinate

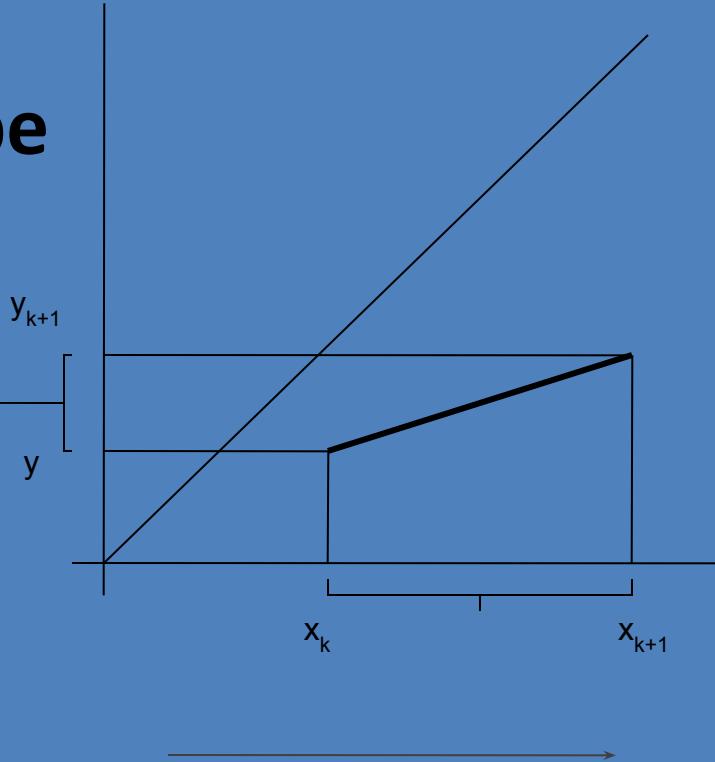
DDA (Digital Differential Analyzer)

Positive Slope
For slope < 1

$$\Delta x > \Delta y$$

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

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



Moving from left to right

DDA (Digital Differential Analyzer)

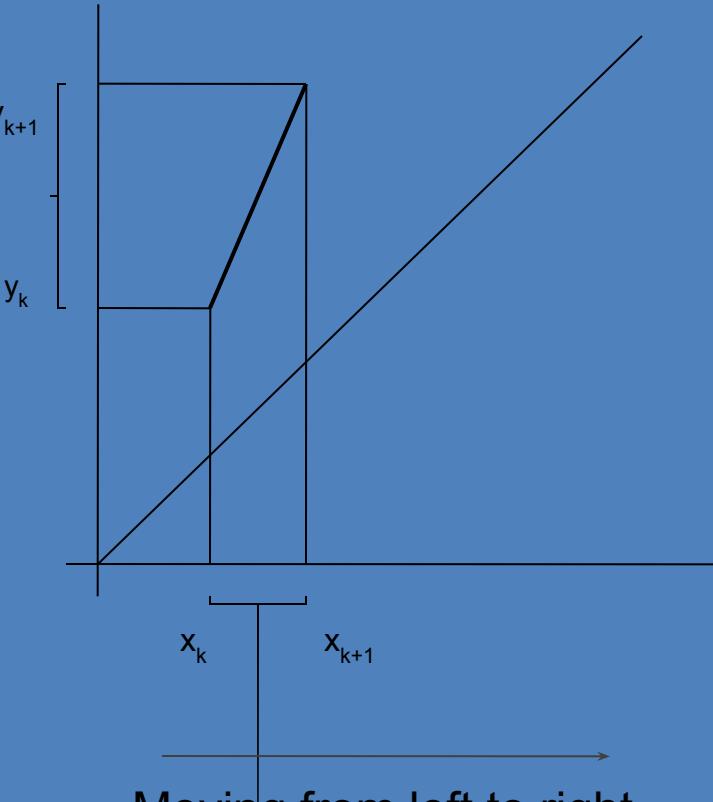
Positive Slope

For slope < 1

$$\Delta y > \Delta x$$

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

$$\text{or } x_{k+1} = x_k + 1/m$$



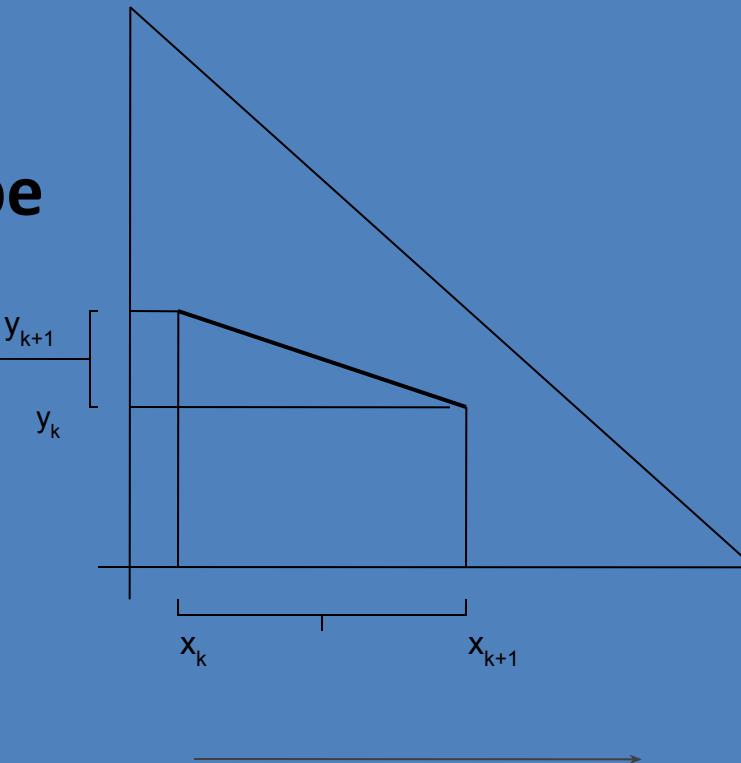
DDA (Digital Differential Analyzer)

Negative Slope
For slope < 1

$$\Delta y > \Delta x$$

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

$$\text{or } y_{k+1} = y_k + m$$



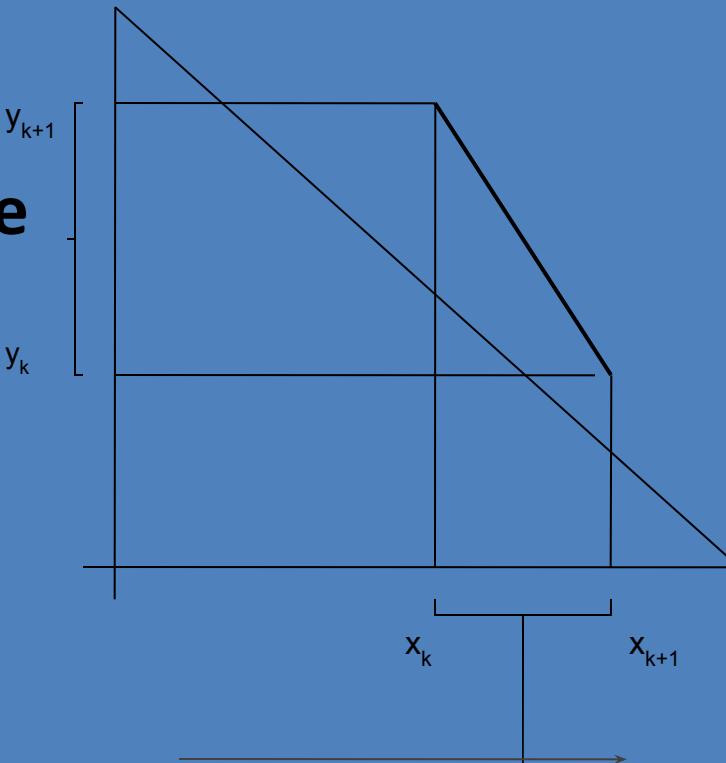
Moving from left to right

DDA (Digital Differential Analyzer)

Negative Slope
For slope > 1

$$\Delta y > \Delta x$$

$$\text{or } x_{k+1} = x_k - m ?$$



DDA (Digital Differential Analyzer)

Advantages:

- i.Accurate

Disadvantages :

- i.Slower as it uses floating point arithmetic

Bresenham's Line Drawing Algorithm

Accurate and easier line drawing algorithm by Bresenham

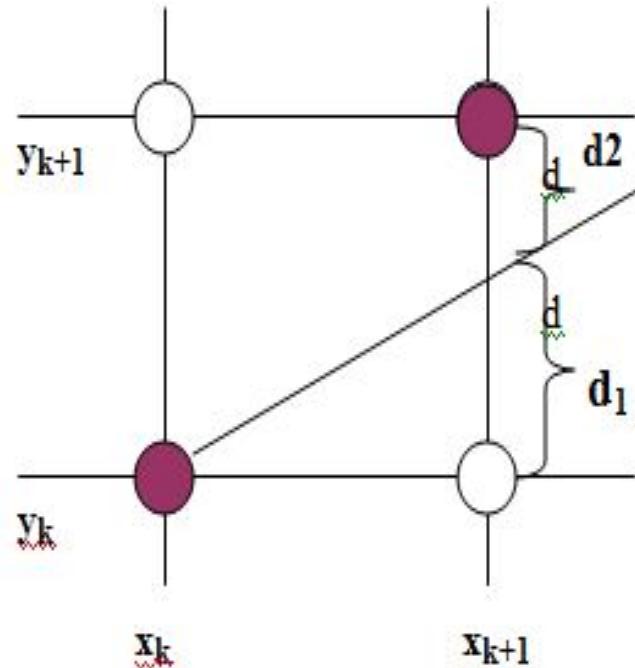
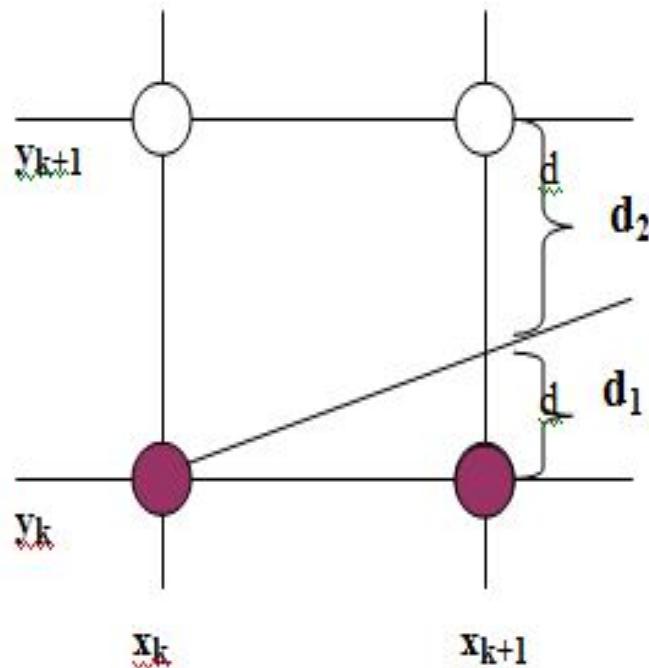
Vertical axis: *scan line*

Horizontal axis: *pixel column*

For slope < 1 sample at unit 'x' interval in 'x' direction, determine which of the two possible pixel positions is closer to the line path at each sample step.

Bresenham's algorithm gives the solution to this problem by testing the sign of the integer parameter whose value is proportional to difference between the separation of two pixel positions from actual line path.

Line Drawing



(x_k, y_k) plotted next to plot (x_{k+1}, y_k) or (x_{k+1}, y_{k+1})

Line Drawing

Algorithm for $0 < m < 1$

Input the two line endpoints and store the left end point in (x_0, y_0)

Load (x_0, y_0) into the frame buffer; that is, plot the first point

Calculate constants Δx , Δy , $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

At each x_k along the line, start at $k = 0$, perform the following test:

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

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

Repeat step 4 Δx times

Dot Pitch

The distance between adjacent sets of red, green and blue dots.

The dot pitch of the monitor indicates how fine the dots are that make up the picture.

The smaller the dot pitch, the more sharp and detailed the image.

Filled Area Primitives

Standard output primitive in graphics packages is solid color ,patterned polygon area

Polygons are easier to process due to linear boundaries

Two basic approaches to area filling on a raster system

- 1. Determine the overlap intervals for scan lines that cross the area.
Typically useful for filling polygons, circles, ellipses**

- 2. Start from a given interior position and paint outwards from
this point until we encounter the specified boundary conditions
useful for filling more complex boundaries, interactive painting
system.**

Filling rectangles

Two things to consider

- i. which pixels **to fill**
- ii. **with what value to fill**

Move along scan line (from left to right) that intersect the primitive and fill in pixels that lay inside

To fill rectangle with solid color

Set each pixel lying on scan line running from left edge to right with same pixel value, each span from x_{max} to x_{min}

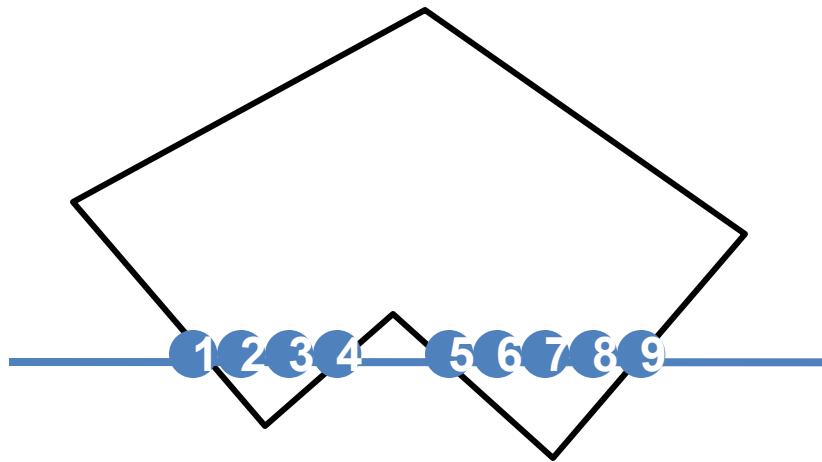
```
for( y from  $y_{min}$  to  $y_{max}$  of rectangle)          /*scan line*/  
    for( x from  $x_{min}$  to  $x_{max}$  of rectangle)      /*by pixel*/  
        writePixel(x, y, value);
```

Filling Polygons

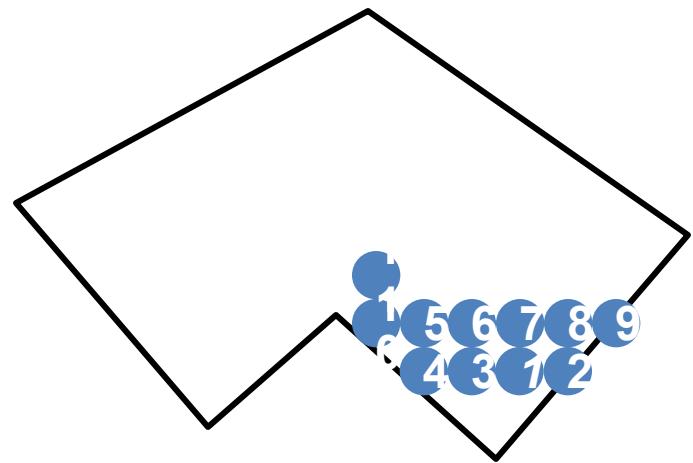
Filling Polygons

Scan-line fill algorithm

Inside-Outside tests



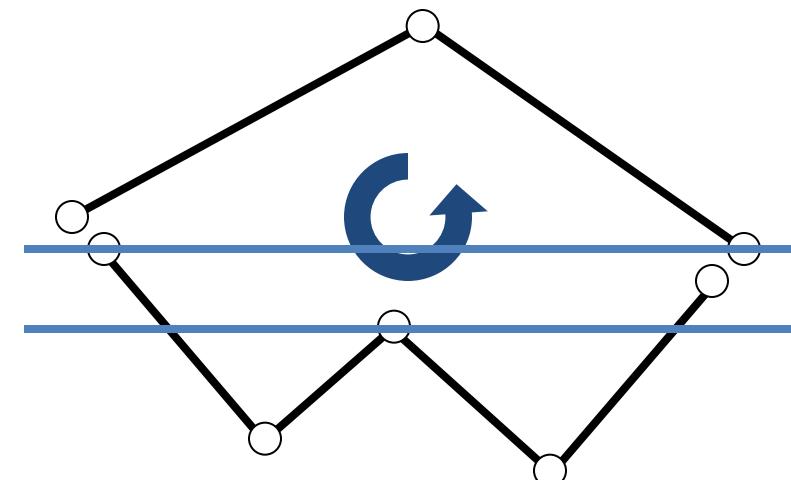
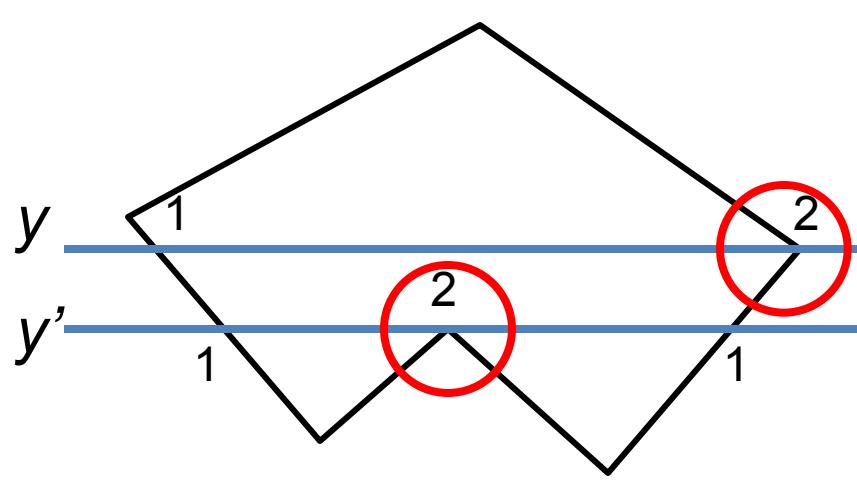
Boundary fill algorithm



Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

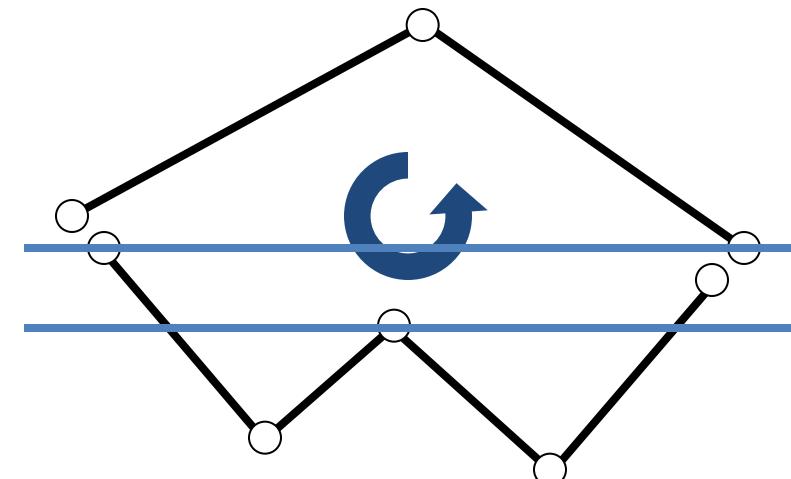
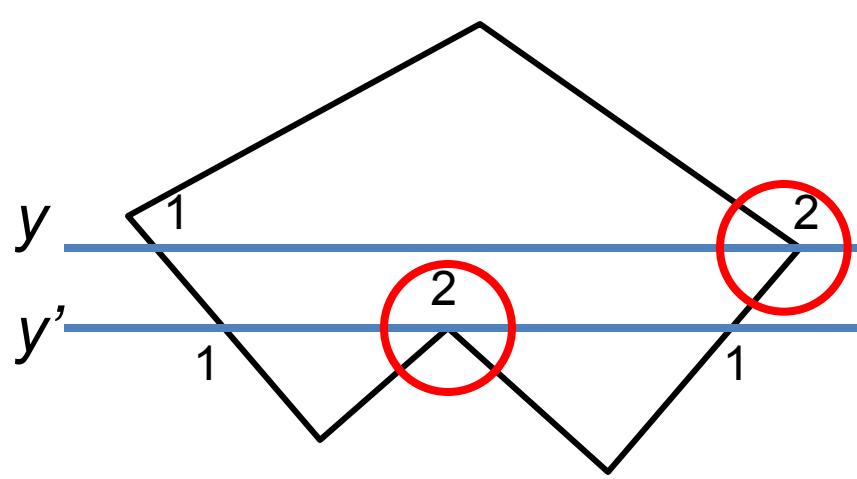
y' : intersection edges are same side



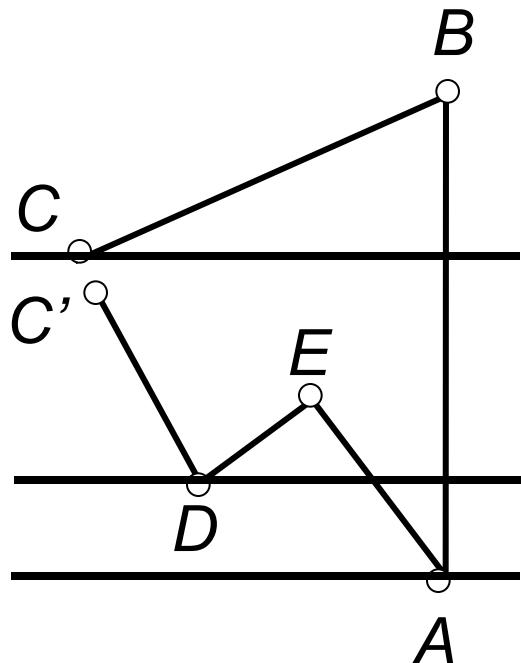
Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

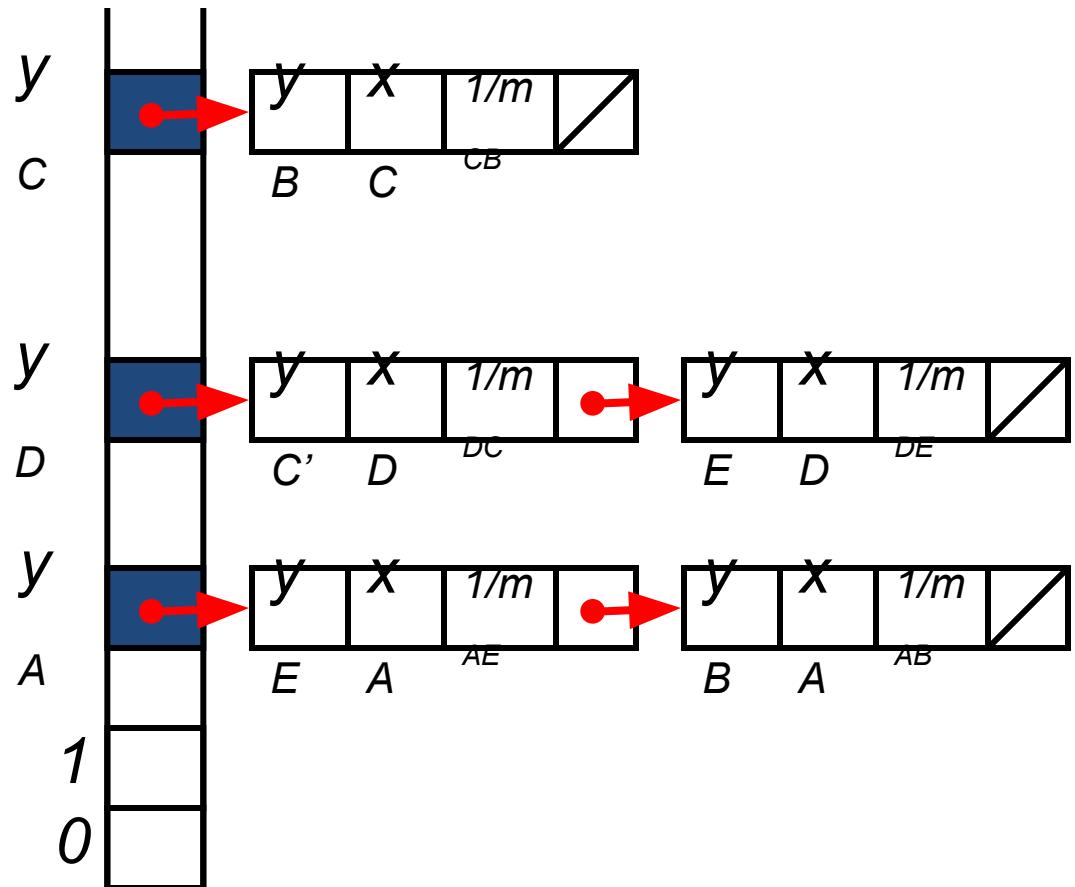
y' : intersection edges are same side



Edge Sorted Table



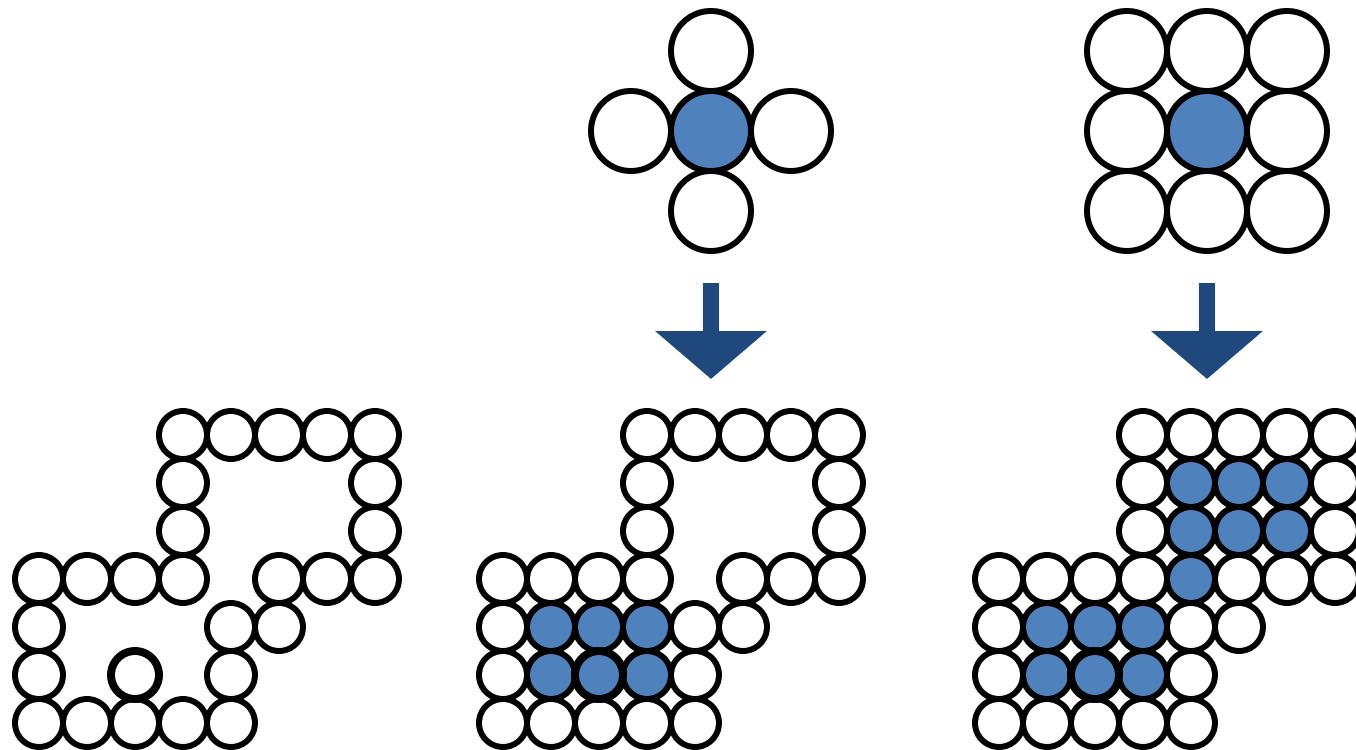
*Scan-Line
Number*



Proceed to Neighboring Pixels

4-Connected

8-Connected



Boundary Fill

Start at a point inside a region **and paint the interior outward toward the boundary.**

If boundary is specified in a single color the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered

Accepts as input coordinates of an interior point (x, y) a fill color and boundary color.

Starting from (x, y) the procedure tests neighboring position to determine whether they are of the boundary color.

If not they are painted with fill color and their neighbors are tested

Process continues until all pixels up to boundary color for the area have been tested

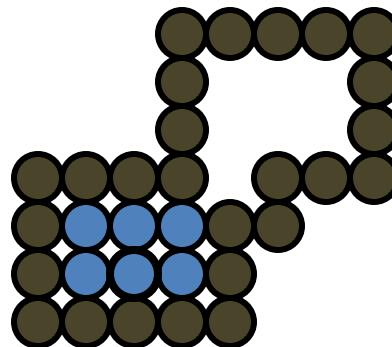
boundaryFill ()

Retrive intensity of the pixel inside the boundary

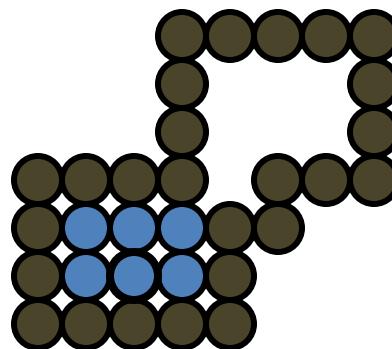
If the retreived intensity is not that of the boundary pixels and AND it is not

the fill color

use recursion to plot pixel in all four directions



```
boundaryFill(int x,y,fill_color, boundary_color){
    int color;
    getpixel(x,y,color)
    if(color != boundary_color AND color != fill_color ){
        setpixel( x,y,fill_color)
        boundaryFill( x+1,y,fill_color, boundary_color)
        boundaryFill( x,y+1,fill_color, boundary_color)
        boundaryFill( x-1,y,fill_color, boundary_color)
        boundaryFill( x,y-1,fill_color, boundary_color)
    }
}
```



Flood Fill

Filling an area that is not defined within a single color boundary.

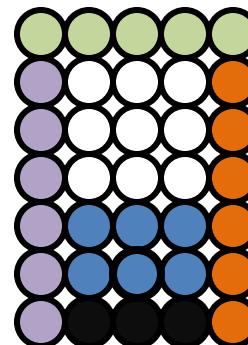
In this case replace a specified interior color instead of searching for boundary color value.

Start from specified interior point (x , y) and reassign all pixel values that are currently set to a given interior color with the desired color.

If the area we want to paint has more than 1 interior color , first assign pixel values so that all interior points have same color.

We can then use 8 or 4 connected approach to move on until all interior points have been repainted.

```
floodFill(int x,y,fill_color, original_color){  
    int color;  
    getpixel(x,y,color)  
    if(color == original_color ){  
        setpixel(x,y,fill_color)  
        floodFill(x+1,y,fill_color, original_color)  
        floodFill(x,y+1,fill_color, original_color)  
        floodFill(x-1,y,fill_color, original_color)  
        floodFill(x,y-1,fill_color, original_color)  
    }  
}
```



Line Drawing

Point plotting is accomplished by converting a single coordinate position by an application program into approximate operation for the output device in use.

CRT electron beam is **turned on** to illuminate the screen phosphor at selected location.

Line Drawing

In **random scan system**, point plotting commands are stored in display list and **coordinate values in these instructions are converted to deflection voltages** that position the electron beam at that screen location to be plotted during each refresh cycle.

In case of black and white **raster scan system** a point is plotted by **setting the bit value corresponding to specified screen position within *frame buffer* to 1.**

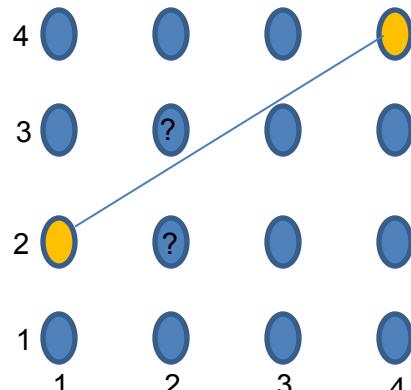
Line Drawing

For drawing lines, we need to calculate **intermediate positions** along the line path between two end points

e.g. 10.45 is rounded off to 10 (this causes **stair cases** or **jaggies** to be formed)

To *load* intensity value into frame buffer at position x, y we use function
setpixel(x, y, intensity)

To *retrieve* current frame buffer intensity value for specified location we use
getpixel(x,y)



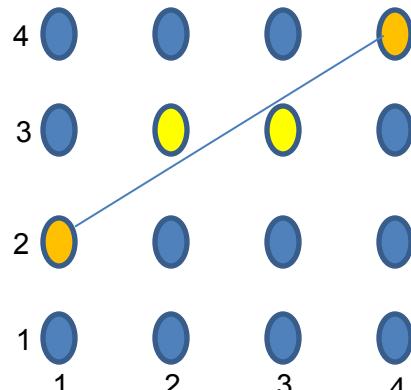
Line Drawing

For drawing lines, we need to calculate **intermediate positions** along the line path between two end points

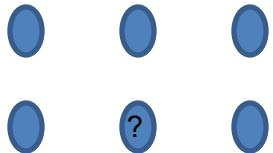
e.g. 10.45 is rounded off to 10 (this causes **stair cases** or **jaggies** to be formed)

To *load* intensity value into frame buffer at position x, y we use function
setpixel(x, y, intensity)

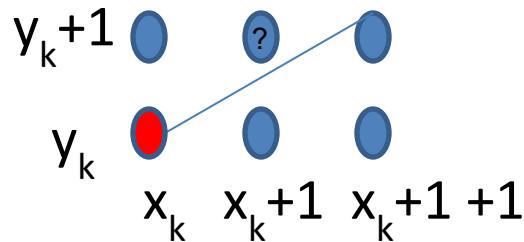
To *retrieve* current frame buffer intensity value for specified location we use
getpixel(x,y)



Line Drawing



$x_{k+1} \Rightarrow \text{next}$



$x_k + 1$ one hop in x direction

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

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

$x_{k+1} = x_k + 2 ??$

DDA (Digital Differential Analyzer)

Slope intercept equation of a straight line is $y = m x + c$

For points (x_1, y_1) and (x_2, y_2) slope $m = y_2 - y_1 / x_2 - x_1$ or $\Delta y / \Delta x$

Algorithms for displaying lines depend on these equations for given interval x we can compute y interval

$$\Delta y = m \cdot \Delta x$$

' x ' interval Δx is obtained by $\Delta x = \Delta y / m$

These equations **form the basis for determining deflection voltages in analog devices**

DDA (Digital Differential Analyzer)

For lines with slope $m = 1$, Δy and Δx are equal

DDA scan conversion algorithm is **based on calculating Δx , Δy**

We **sample line at unit interval** in one coordinate and **determine corresponding integer values** nearest the line path for other coordinate

DDA (Digital Differential Analyzer)

Eight Cases:

Lines with positive slope:

Slope less than 1

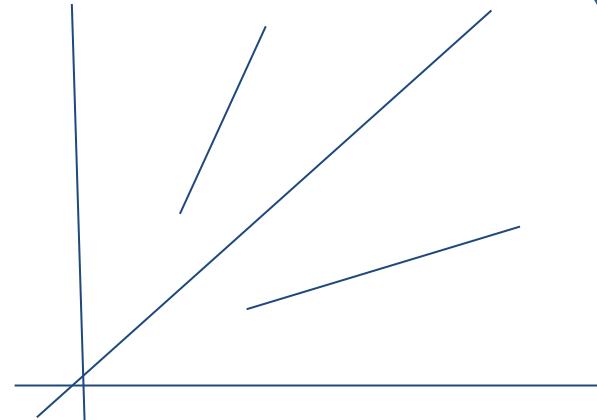
moving from Left to Right

moving from Right to Left

Slope greater than 1

moving from Left to Right

moving from Right to Left



Lines with negative slope:

|Slope| less than 1 $|-1/3| = 1/3$

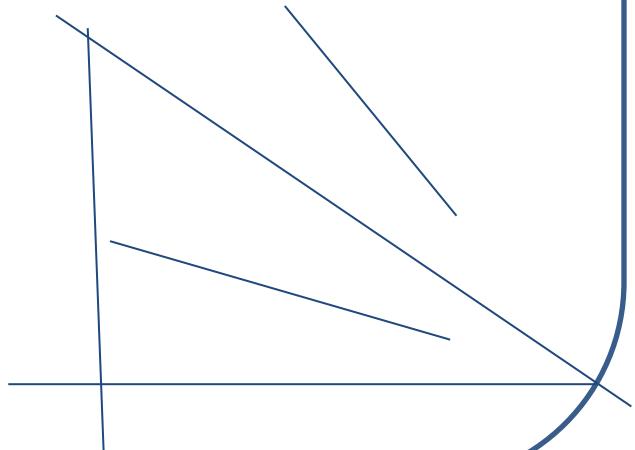
moving from Left to Right

moving from Right to Left

|Slope| greater than 1 $|-3/2| = 3/2$

moving from Left to Right

moving from Right to Left



DDA (Digital Differential Analyzer)

Positive Slope

For slope ≤ 1

$$\Delta x > \Delta y \quad 1/5$$

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

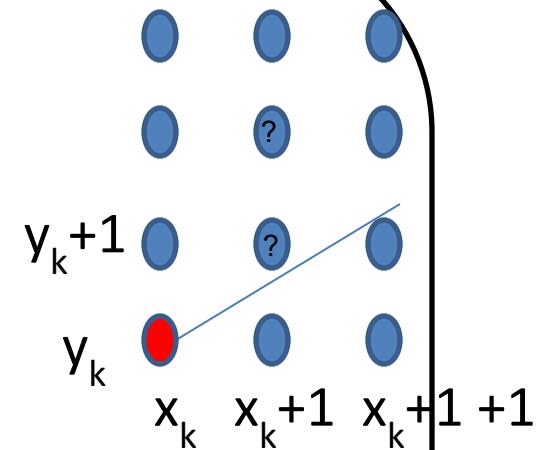
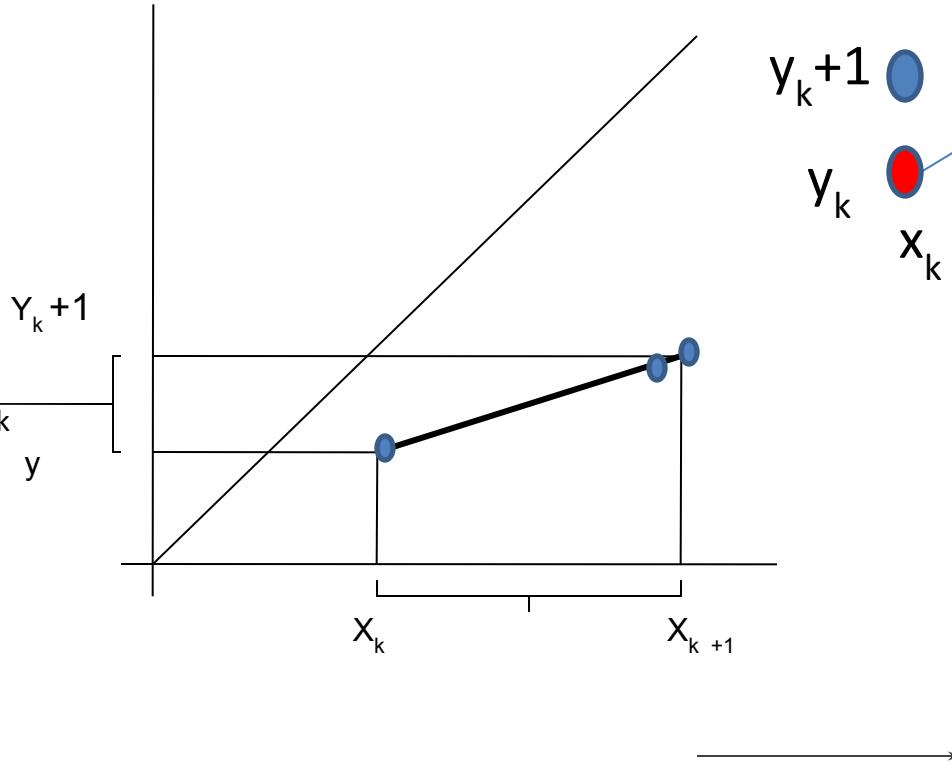
$$x_{k+1} = x_k + 1$$

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

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

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

$$\text{or } y_{k+1} = y_k + m$$



DDA (Digital Differential Analyzer)

Positive Slope

For slope > 1

$$\Delta y > \Delta x$$

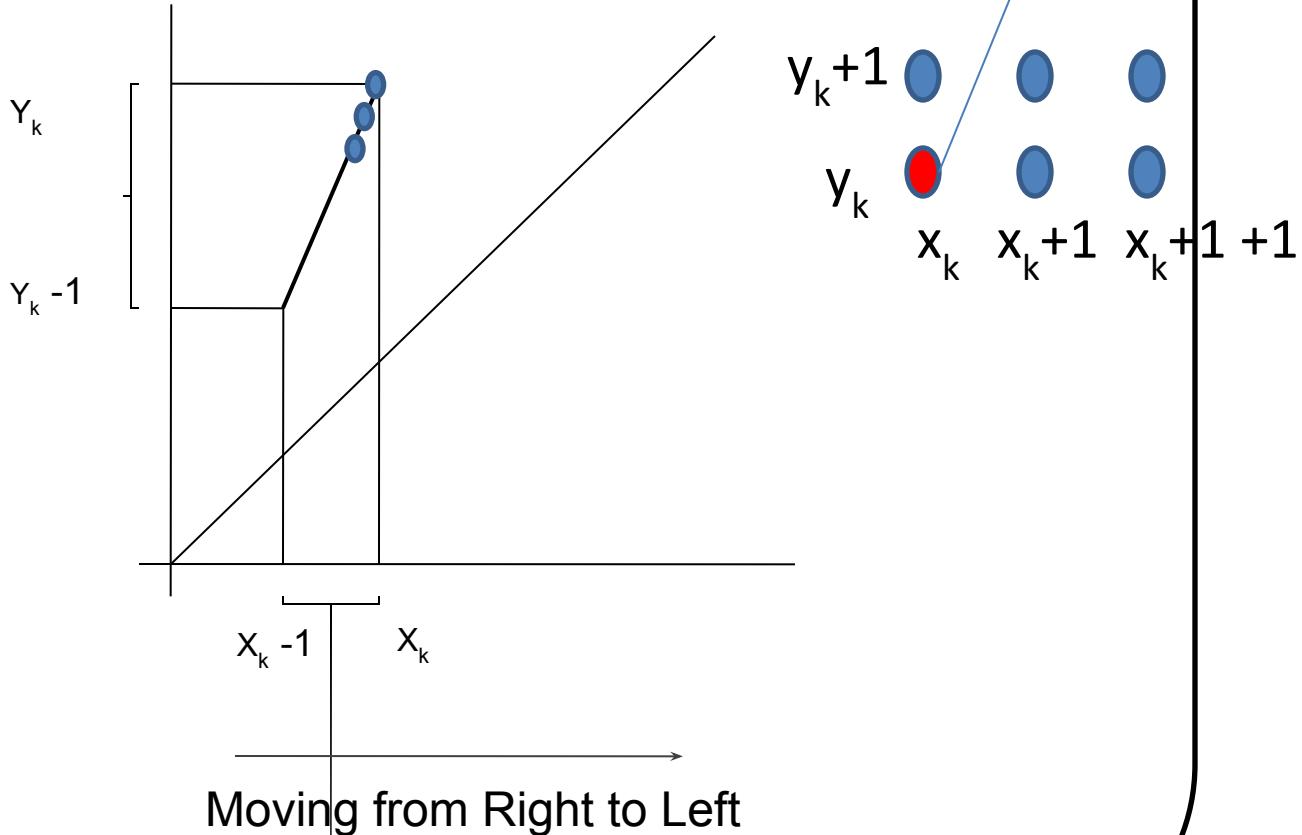
$$y_{k+1} - y_k = -1$$

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

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

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

$$\text{or } x_{k+1} = x_k - 1/m$$

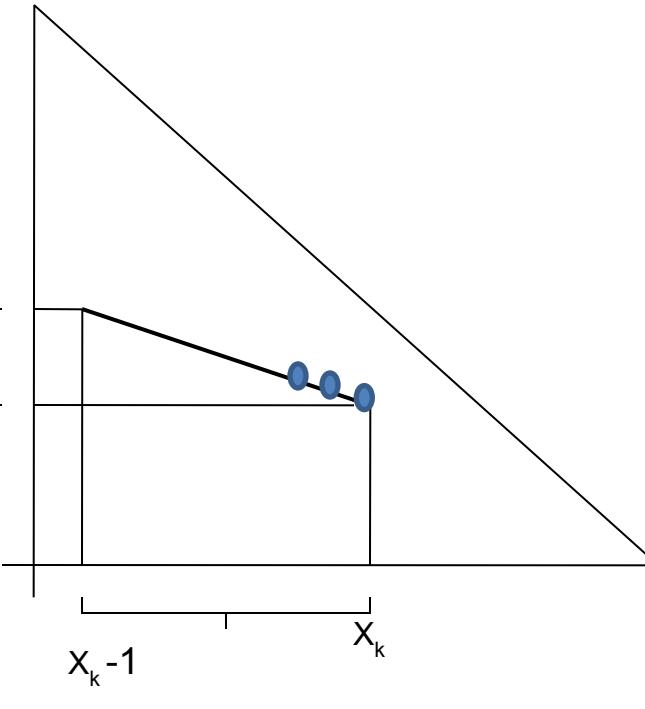


DDA (Digital Differential Analyzer)

Negative Slope
For $|slope| < 1$

$$\Delta x = -1$$
$$-1/3$$
$$x_{k+1} - x_k = -1 \quad x_{k+1} = x_k - 1 \quad y_k$$
$$m = y_{k+1} - y_k / x_{k+1} - x_k$$
$$m = y_{k+1} - y_k / -1$$

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



Moving from Right to Left

DDA (Digital Differential Analyzer)

Negative Slope

For $|slope| > 1$

$$m = -4/3$$

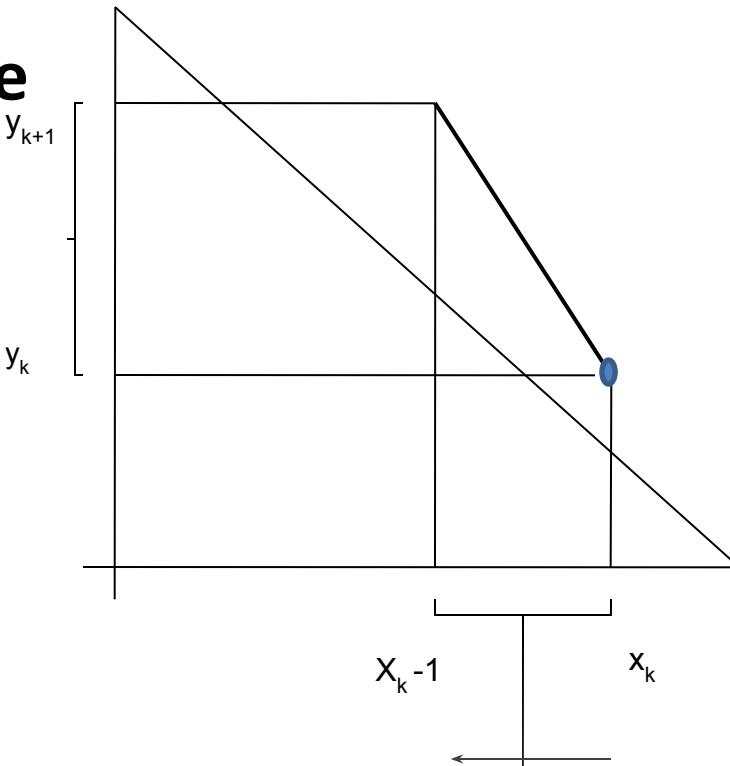
$$\Delta y = 1$$

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

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

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

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



DDA (Digital Differential Analyzer)

Advantages:

- i.Accurate

Disadvantages :

- i.Slower as it uses floating point arithmetic
- ii.The accumulation of round off error

DDA (Digital Differential Analyzer)

```
20   10   30   13
void lineDDA (int xa, int ya, int xb, int yb){
    int dx = xb - xa, 10 dy = yb - ya, 3 steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
    if (abs (dx) > abs (dy))
        steps = abs (dx) ; 10
    else
        steps = abs (dy); 1
    xIncrement = dx / (float) steps; 1
    yIncrement = dy / (float) steps 0.3 = 0
    setpixel (ROUND(x), ROUND(y));
    for (k=0; k<steps; k++) {
        x += xIncrement;
        y += yIncrement;
        setpixel (ROUND(x), ROUND(y));
    } 21 10
}
```

Bresenham's Line Drawing Algorithm

Accurate and easier line drawing algorithm by Bresenham

Vertical axis: *scan line*

Horizontal axis: *pixel column*

For slope < 1 sample at unit 'x' interval in 'x' direction, determine which of the two possible pixel positions is closer to the line path at each sample step.

Bresenham's algorithm gives the solution to this problem by testing the sign of the integer parameter whose value is proportional to difference between the separation of two pixel positions from actual line path.

Accurate and easier line drawing algorithm by Bresenham

Vertical axis: *scan line*

Horizontal axis: *pixel column*

For slope < 1 sample at unit 'x' interval in 'x' direction, determine which of the two possible pixel positions is closer to the line path at each sample step.

Bresenham's algorithm gives the solution to this problem by testing the sign of the integer parameter whose value is proportional to difference between the separation of two pixel positions from actual line path.

Line Drawing

Use DDA Algorithm to digitize a line with end points A(3, 1)

B(6,3)

$$m = 2/3 = 0.6$$

positive slope/slope <1/
left to right

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

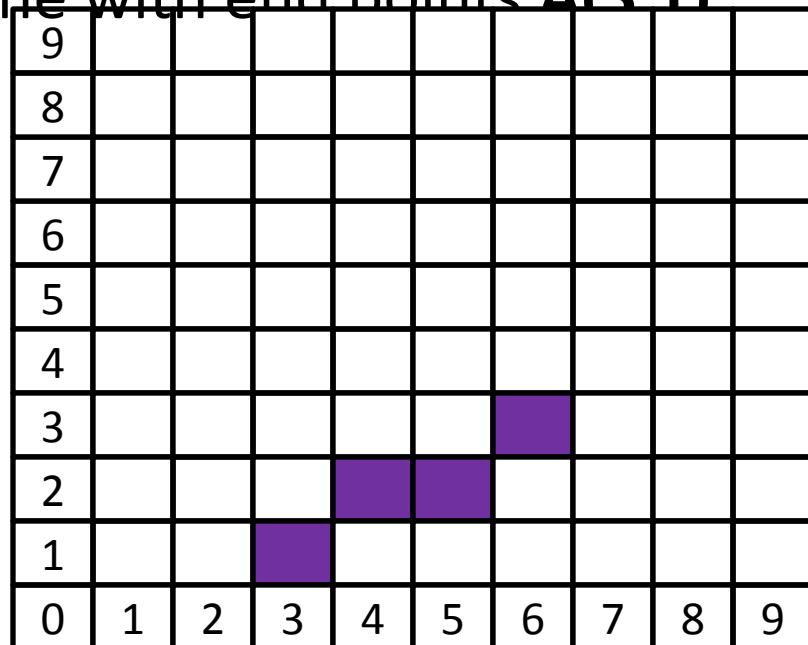
$$m = y_{k+1} - y_k / x_{k+1} - x_k$$

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

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

starting point(3,1)

x_{k+1}	y_{k+1}	rounded value y_{k+1}
4	$y_{k+1} = 1 + 0.6 = 1.6$	2
5	$y_{k+1} = 1.6 + 0.6 = 2.2$	2
6	$y_{k+1} = 2.2 + 0.6 = 2.8$	3



Line Drawing

Use DDA Algorithm to digitize a line with end points

A(10,10) B(12,15)

$$m = 5/2 = 2.5$$

Positive slope/slope >1/left to right

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

$$m = y_{k+1} - y_k / x_{k+1} - x_k \quad m = 1 / x_{k+1} - x_k$$

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

starting point(10,10)

x_{k+1}	rounded value x_{k+1}	$y_{k+1} = y_k + 1$
$x_{k+1} = 10 + 0.4 = 10.4$	10	11
$x_{k+1} = 10.4 + 0.4 = 10.8$	11	12
$x_{k+1} = 10.8 + 0.4 = 11.2$	11	13
$x_{k+1} = 11.2 + 0.4 = 11.6$	12	14
$x_{k+1} = 11.6 + 0.4 = 12$	12	15

Line Drawing

Use DDA Algorithm to digitize a line with end points

A(30,15) B(20,26)

$m = 11/-10 = -1.1$ starting point A(30,15)

negative slope, $|slope| > 1$ $|m| =$

Left to Right

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

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

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

$$\begin{matrix} x_{k+1} \\ x_{k+1} = x_k + 1/m \end{matrix}$$

16

17

18

19

20

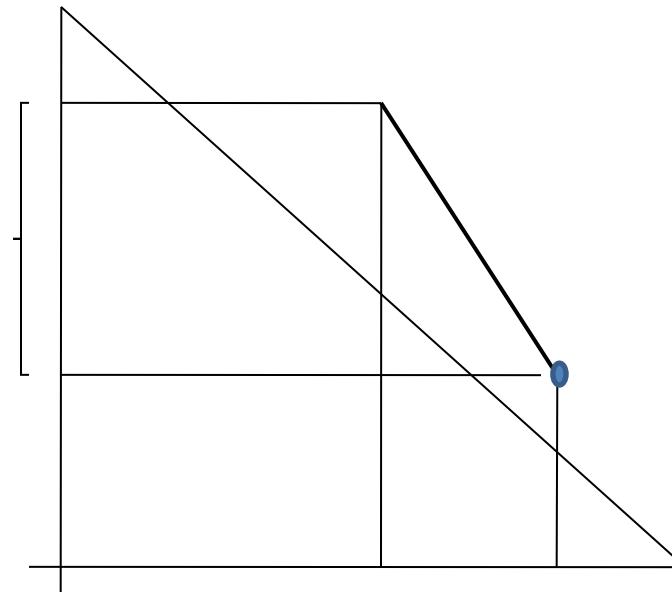
21

22

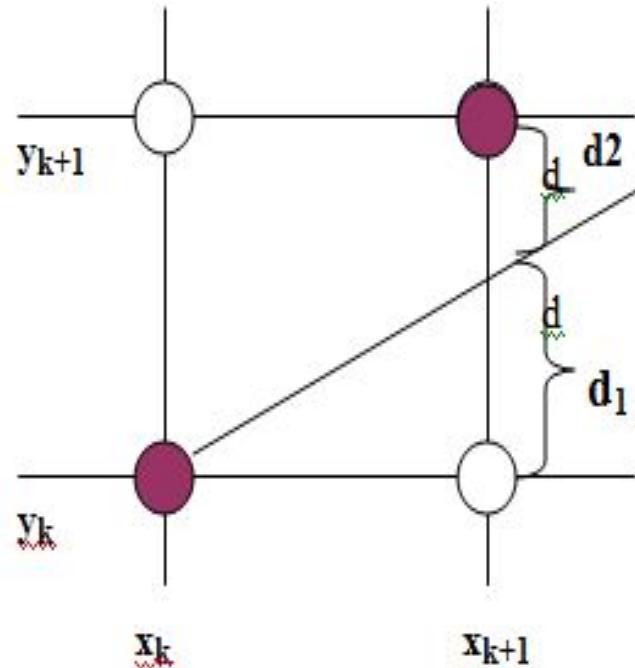
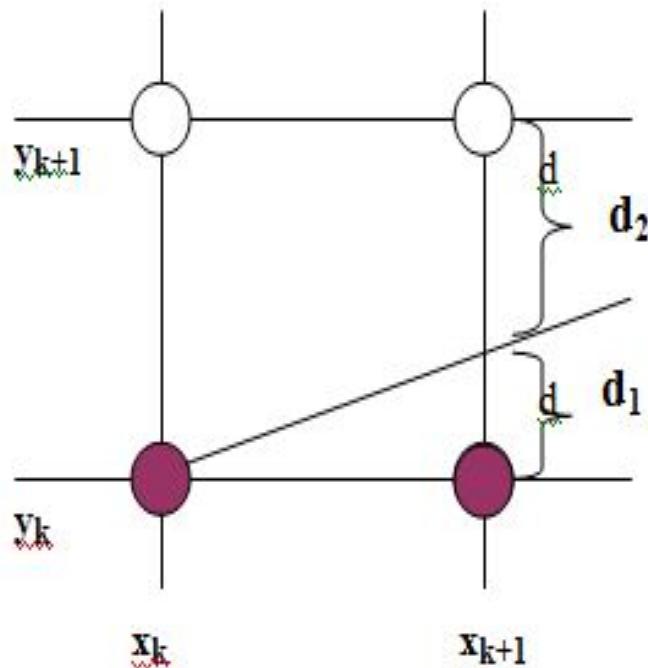
23

24

25



Line Drawing



(x_k, y_k) plotted next to plot (x_{k+1}, y_k) or (x_{k+1}, y_{k+1})

Line Drawing

Algorithm for $0 < m < 1$

Input the two line endpoints and store the left end point in (x_0, y_0)

Load (x_0, y_0) into the frame buffer; that is, plot the first point

Calculate constants Δx , Δy , $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

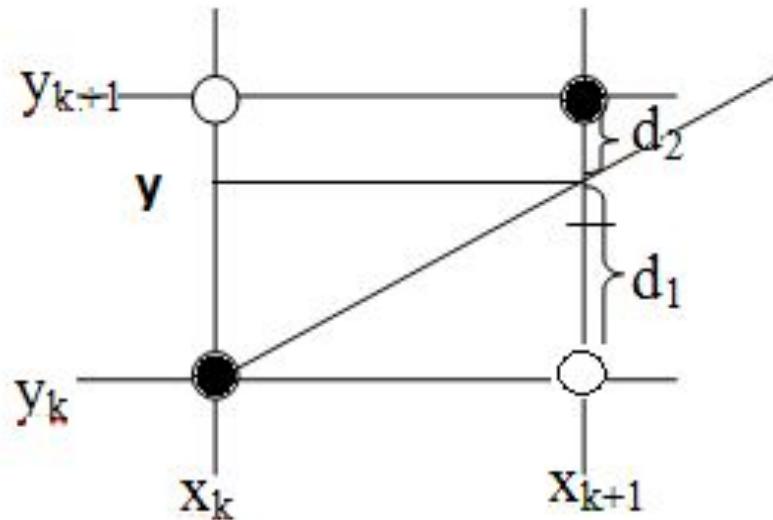
At each x_k along the line, start at $k = 0$, perform the following test:

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

Otherwise, the next point to plot is $(x_k + 1, y_k + 1)$ and

Repeat step 4 Δx times

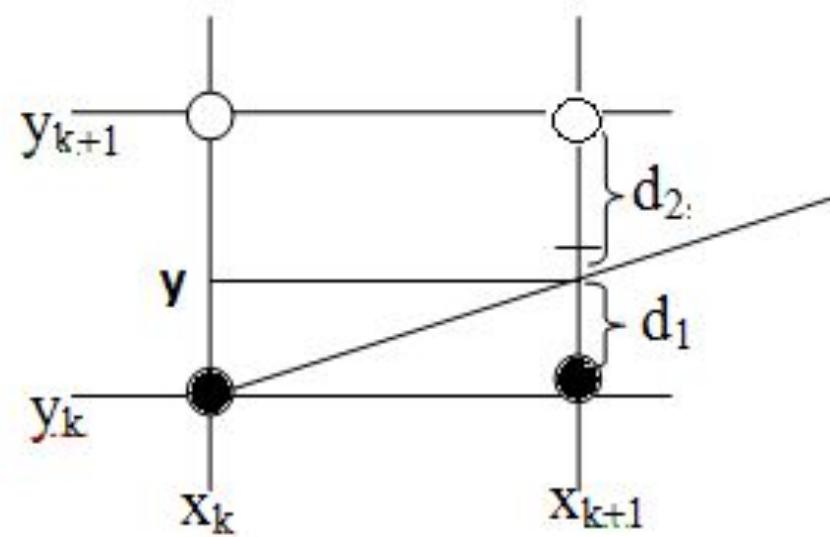
Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



If $d_1 - d_2 \geq 0$ then the pixel on scanline ' $y_k + 1$ ' is closer to the line path and $y_{k+1} = y_k + 1$

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

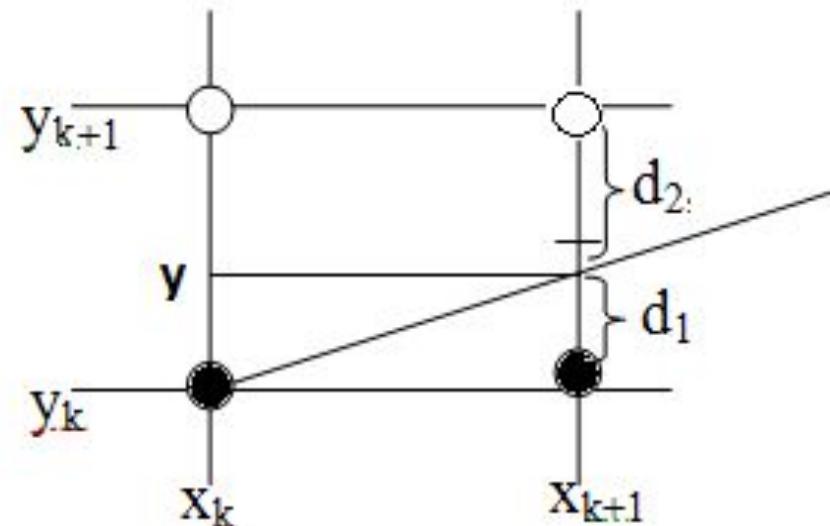
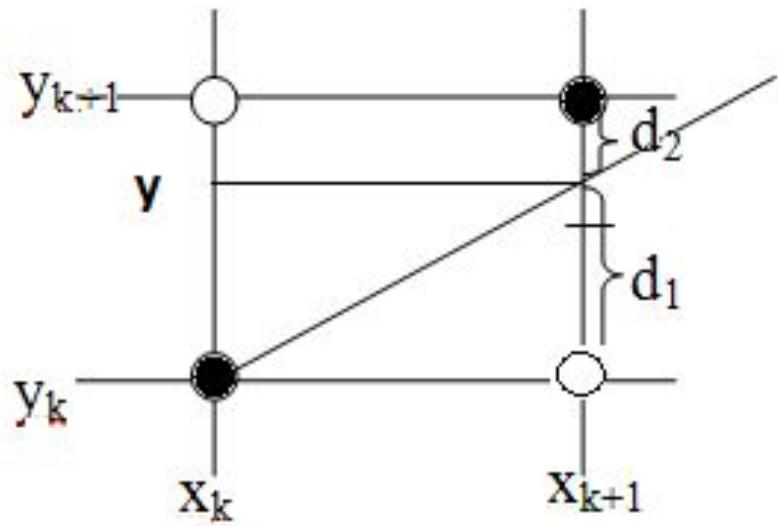
At $x_k + 1$ e.g. of line is $y = m(x_k + 1) + c$



If $d_1 - d_2 < 0$ then the pixel on scanline ' y_k ' is closer to the line path and $y_{k+1} = y_k$

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

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



$$\text{At } x_k + 1 \text{ e.q. of line} \quad y = m(x_k + 1) + c$$

The distance of lower pixel from the ideal location

$$d_1 = y - y_k \quad \text{or} \quad d_1 = m(x_k + 1) + c - y_k$$

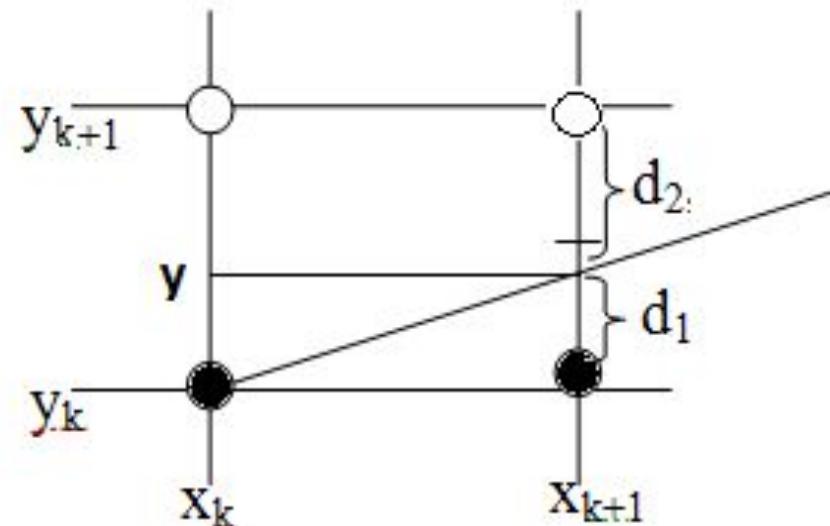
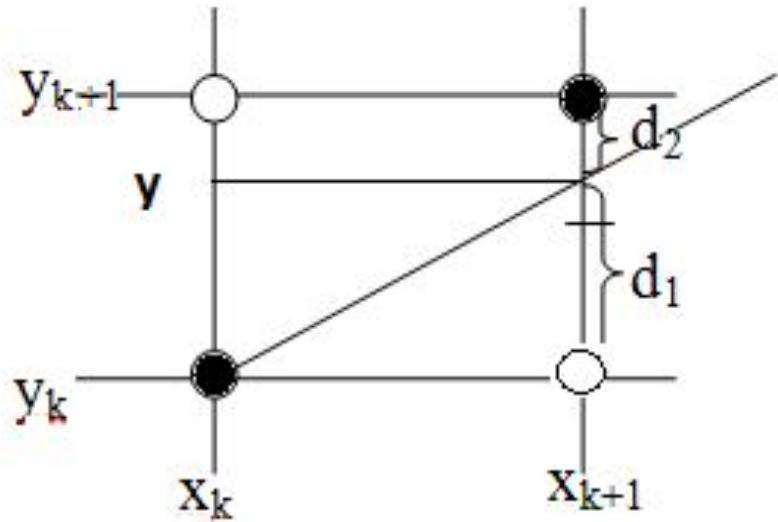
The distance of the ideal location from the upper pixel

$$d_2 = y_k + 1 - y \quad \text{or} \quad d_2 = y_k + 1 - m(x_k + 1) - c$$

Thus the difference between the separations of two pixel positions from the actual line path,

$$d_1 - d_2 = 2m(x_k + 1) + 2c - 2y_k - 1$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



$$\text{At } x_k + 1 \text{ e.q. of line } y = m(x_k + 1) + c$$

The distance of lower pixel from the ideal location

$$d_1 = y - y_k \quad \text{or} \quad d_1 = m(x_k + 1) + c - y_k$$

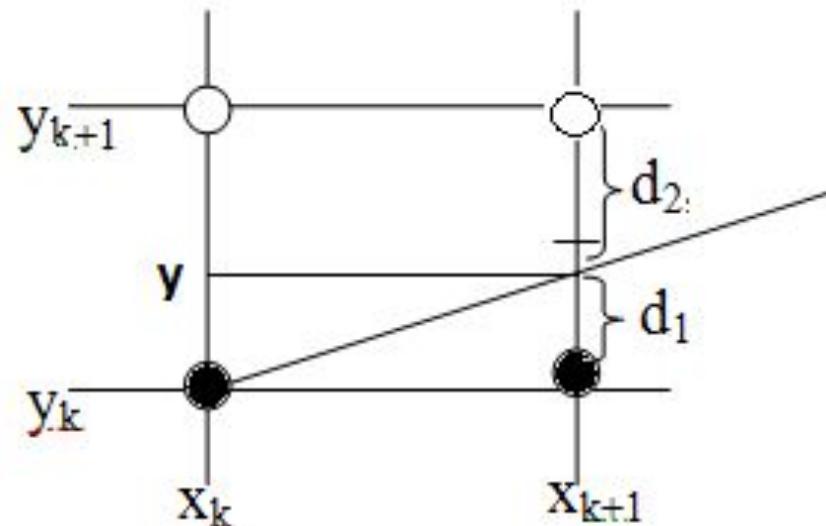
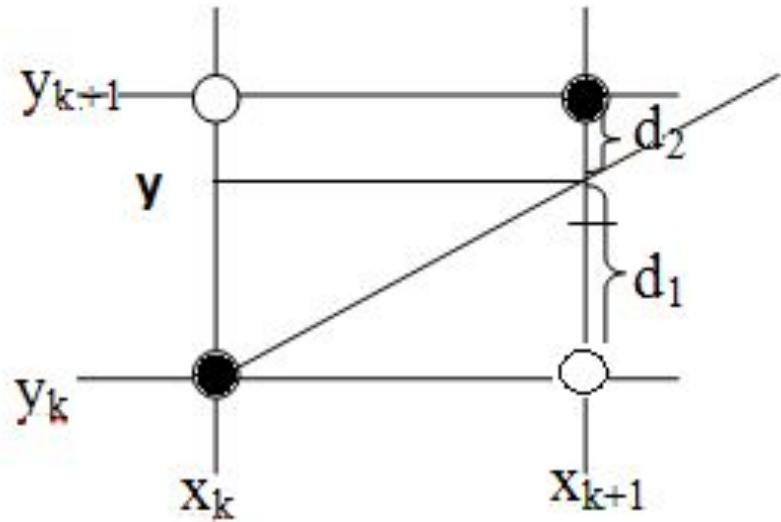
The distance of the ideal location from the upper pixel

$$d_2 = y_k + 1 - y \quad \text{or} \quad d_2 = y_k + 1 - m(x_k + 1) - c$$

Thus the difference between the separations of two pixel positions from the actual line path,

$$d_1 - d_2 = m(x_k + 1) + c - y_k - (y_k + 1) + m(x_k + 1) + c$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



$$d_1 - d_2 = 2m(x_k + 1) + 2c - 2y_k - 1$$

Substituting $m = \Delta y / \Delta x$, we get

$$(d_1 - d_2) = 2 \Delta y / \Delta x (x_k + 1) + 2c - 2y_k - 1$$

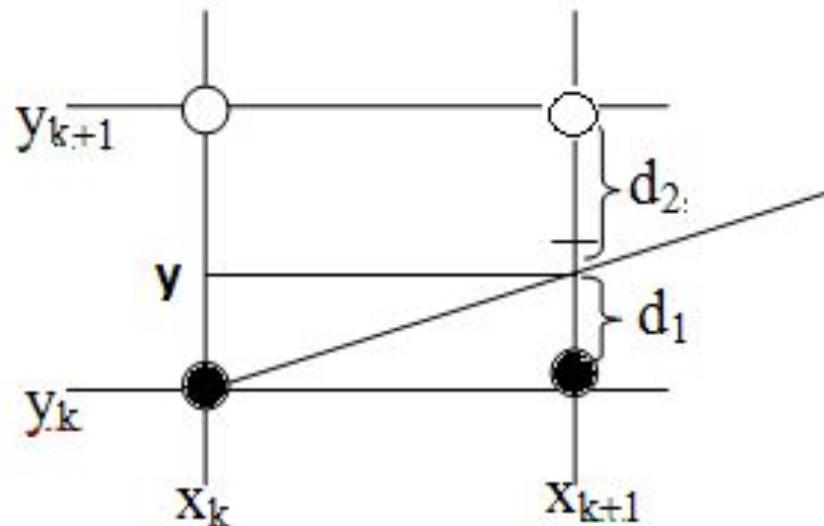
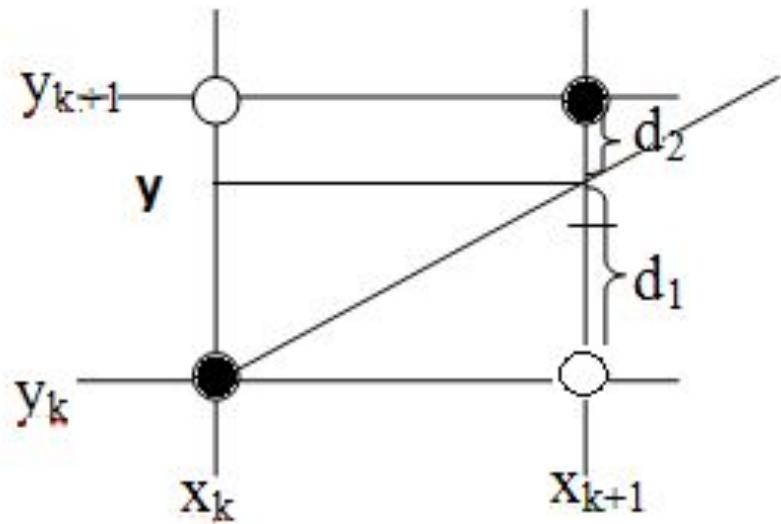
$$P_k = \Delta x \cdot (d_1 - d_2) = 2 \cdot \Delta y \cdot x_k - 2 \cdot \Delta x \cdot y_k + b \quad \dots (i)$$

where $b = 2\Delta y + \Delta x \cdot 2c - \Delta x$ and P_k is the decision parameter at the k^{th} step

At the $k+1^{\text{th}}$ step

$$P_{k+1} = 2 \cdot \Delta y \cdot x_{k+1} - 2 \cdot \Delta x \cdot y_{k+1} + b \quad \dots (ii)$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



$$d_1 - d_2 = 2m(x_k + 1) + 2c - 2y_k - 1$$

Substituting $m = \Delta y / \Delta x$, we get

$$\Delta x(d_1 - d_2) = 2\Delta y(x_k + 1) + 2c\Delta x - 2y_k\Delta x - \Delta x$$

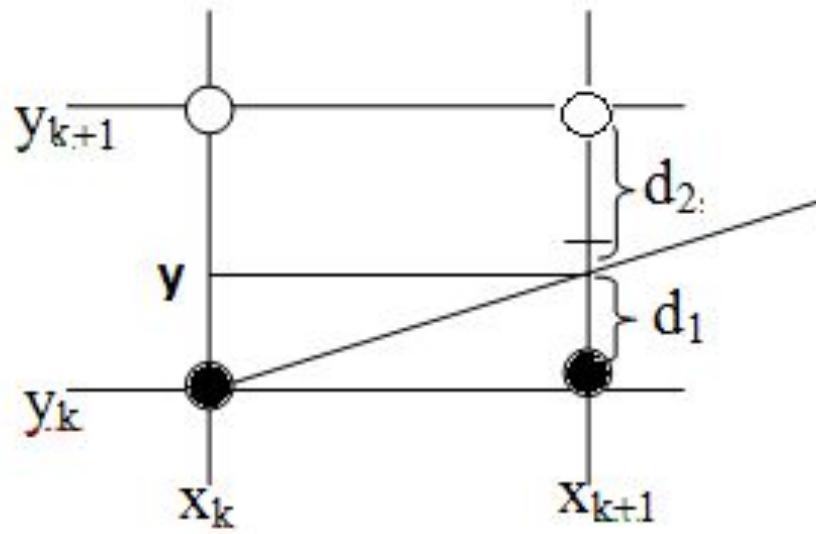
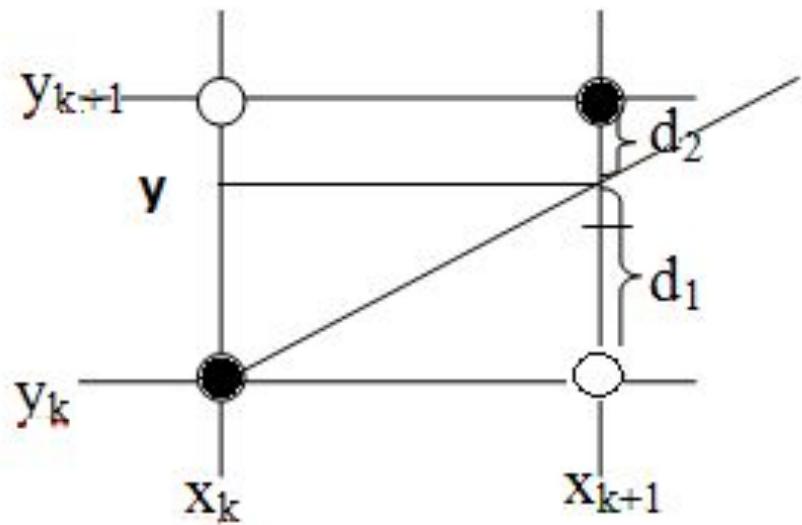
$$P_k = \Delta x.(d_1 - d_2) = 2.\Delta y.x_k - 2.\Delta x.y_k + b \quad \dots (i)$$

where $b = 2\Delta y + \Delta x.2c - \Delta x$ and P_k is the decision parameter at the k^{th} step

At the $k+1^{\text{th}}$ step

$$P_{k+1} = 2.\Delta y.x_{k+1} - 2.\Delta x.y_{k+1} + b \quad \dots (ii)$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



$$P_k = 2 \Delta y \cdot x_k - 2 \Delta x \cdot y_k + b \quad \dots \text{(i)}$$

$$P_{k+1} = 2 \Delta y \cdot x_{k+1} - 2 \Delta x \cdot y_{k+1} + b \quad \dots \text{(ii)}$$

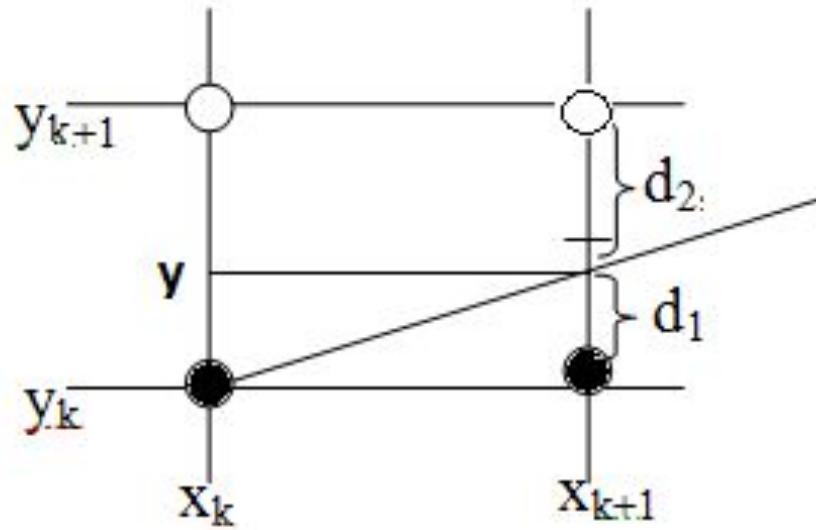
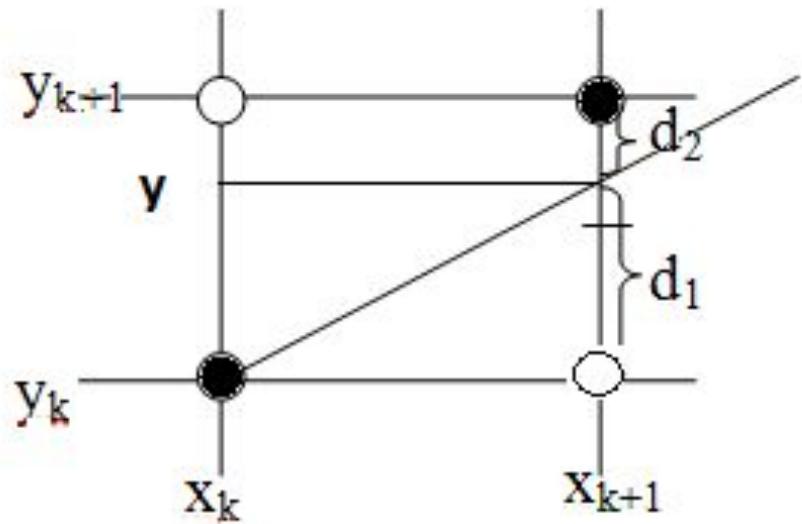
Now subtracting (i) and (ii)

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

Since the slope of the line is less than one, we sample in 'x' direction
i.e. $x_{k+1} = x_k + 1$ so,

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x \cdot (y_{k+1} - y_k) \quad \dots \text{(iii)}$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



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

.... (i)

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

.... (ii)

Now subtracting (i) and (ii)

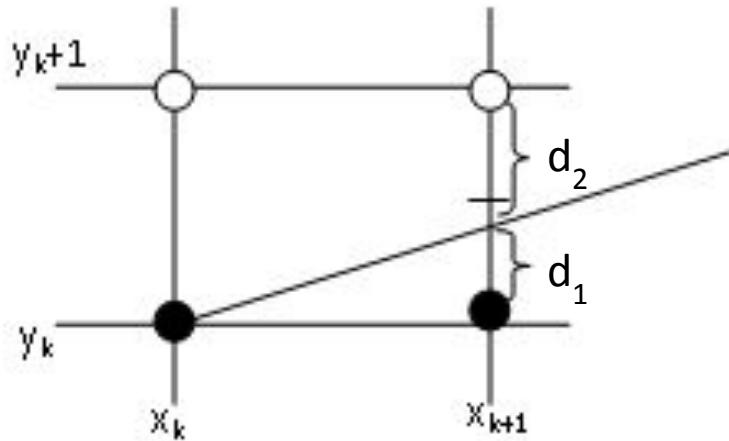
$$P_{k+1} = P_k + 2 \Delta y(x_{k+1} - x_k) - 2 \Delta x(y_{k+1} - y_k)$$

Since the slope of the line is less than one, we sample in 'x' direction

i.e. $x_{k+1} = x_k + 1$ so,

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

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



Case 1:

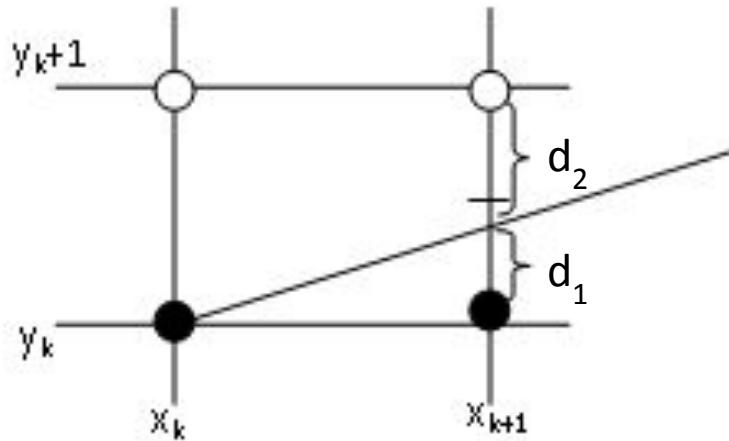
if $P_k < 0$ i.e. $d_1 - d_2 < 0$ or $d_1 < d_2$ then the pixel on scanline ' y_k ' is closer to the line path and $y_{k+1} = y_k$

i.e. from equation (iii)

$$P_{k+1} = P_k + 2 \Delta y - 2\Delta x.(y_{k+1} - y_k) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2 \Delta y \quad \dots \text{ (iii) when lower pixel is chosen}$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



Case 1:

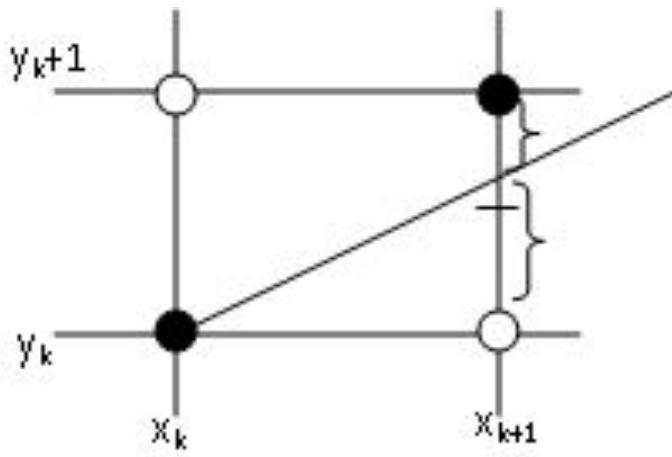
if $P_k < 0$ i.e. $d_1 - d_2 < 0$ or $d_1 < d_2$ then the pixel on scanline ' y_k ' is closer to the line path and $y_{k+1} = ?$

i.e. from equation (iii)

$$P_{k+1} = P_k + 2 \Delta y - 2\Delta x.(y_k - y_k) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2 \Delta y \quad \dots \text{ (iii) when lower pixel is chosen}$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



Case 2:

if $P_k \geq 0$ i.e. $d_1 - d_2 \geq 0$ or $d_1 \geq d_2$ then the pixel on scanline ' $y_k + 1$ ' is closer to the line path and $y_{k+1} = ?$

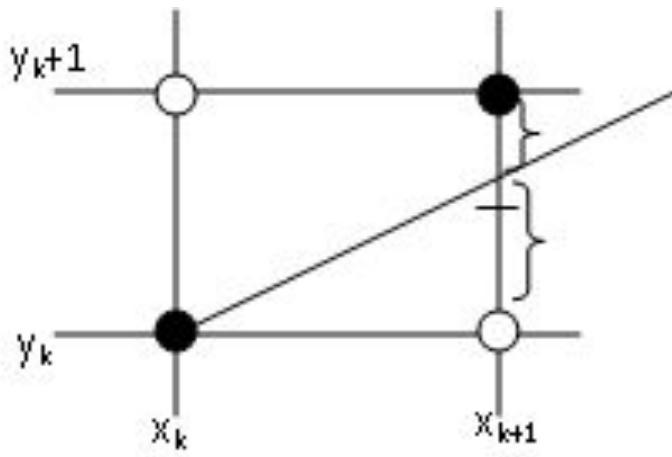
i.e. from equation (iii)

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x.(y_{k+1} - y_k) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x.(y_k + 1 - y_k) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x \quad \dots \text{ (iii)}$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



Case 2:

if $P_k \geq 0$ i.e. $d_1 - d_2 \geq 0$ or $d_1 \geq d_2$ then the pixel on scanline ' $y_k + 1$ ' is closer to the line path and $y_{k+1} = y_k + 1$

i.e. from equation (iii)

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x.(y_{k+1} - y_k) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x.(+1) \quad \dots \text{ (iii)}$$

$$P_{k+1} = P_k + 2\Delta y - 2\Delta x \quad \dots \text{ (iii)}$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1

The Initial Decision Parameter $P_0 = ?$

We have,

$$d_1 = y - y_k \quad \text{or}$$

$$d_1 = m(x_k + 1) + c - y_k$$

$$d_2 = y_k + 1 - y \quad \text{or}$$

$$d_2 = y_k + 1 - m(x_k + 1) - c$$

$$\textcolor{red}{mx_0 + c - y_0 = 0}$$

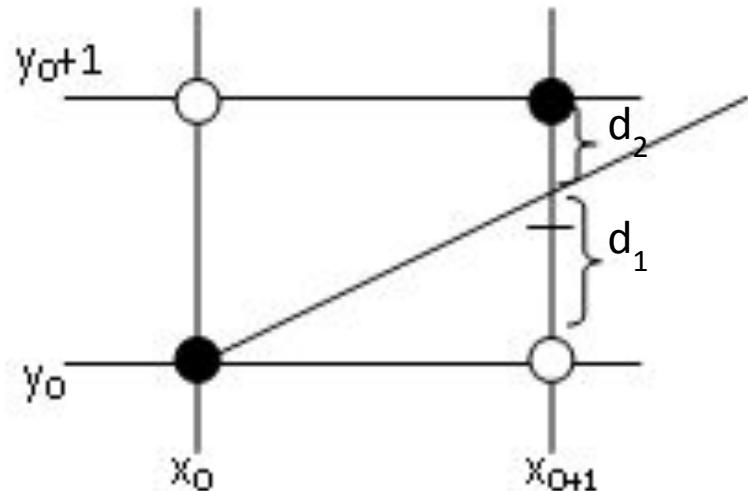
$$d_1 - d_2 = 2m(x_k + 1) + 2c - 2y_k - 1$$

if the line passes thru (x_0, y_0) then

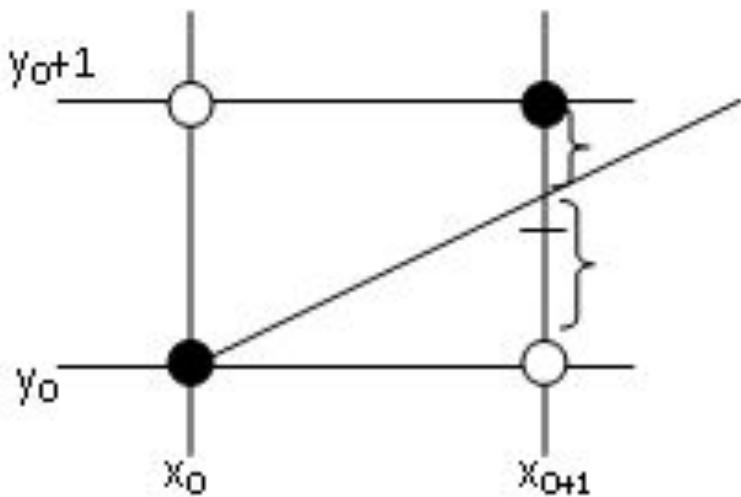
$$\begin{aligned} d_1 - d_2 &= 2m(x_0 + 1) + 2c - 2y_0 - 1 \\ &= 2mx_0 + 2m + 2c - 2y_0 - 1 \end{aligned}$$

$$\text{or } d_1 - d_2 = 2m - 1 = 2 \Delta y / \Delta x - 1 \quad \Delta \quad \text{since, ?}$$

$$\text{or } P_0 = \Delta x(d_1 - d_2) = 2 \Delta y - \Delta x$$



Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



The Initial Decision Parameter $P_0 = ?$

We have,

$$d_1 - d_2 = 2m(x_k+1) + 2c - 2y_k - 1$$

if the line passes thru (x_0, y_0) then

$$\begin{aligned} d_1 - d_2 &= 2m - 1 + 2mx_0 + 2c - 2y_0 \\ &= 2m - 1 \end{aligned}$$

or $d_1 - d_2 = 2\Delta y / \Delta x - 1$ since, ?

or $P_0 = \Delta x (d_1 - d_2) = 2\Delta y - \Delta x$

Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1

Equation of line is $y = m(x_k + 1) + c$

$$d_1 = ? \quad d_2 = ?$$

$$d_1 - d_2 = ?$$

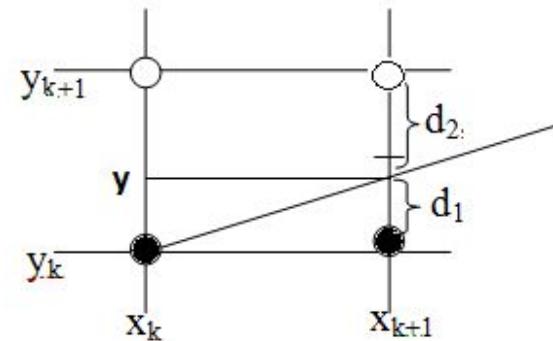
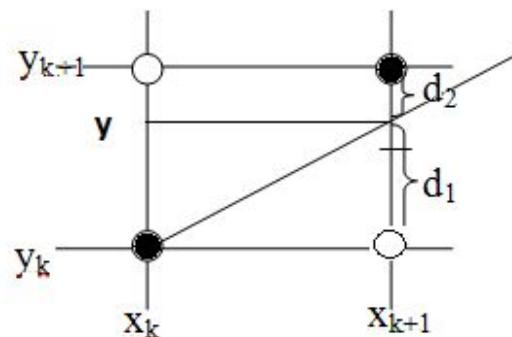
$$\text{at kth step } P_k = \Delta x (d_1 - d_2) = ? \dots \dots \dots (1)$$

$$P_{k+1} - P_k = ? \quad \dots \dots \dots (3)$$

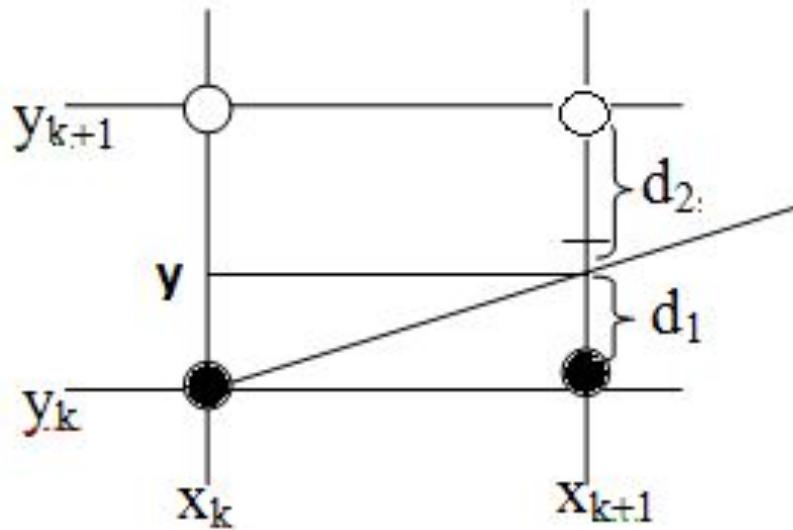
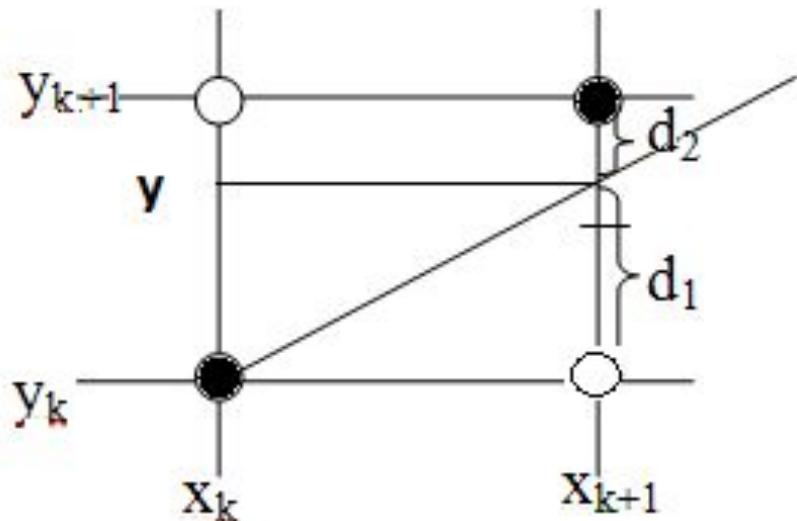
Two cases:

Case 1: $P_k \leq 0$

Case 2: $P_k > 0$



Bresenham's Line Drawing Algorithm for Lines with Slope ≤ 1



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

if $P_k < 0$ ($d_1 - d_2 < 0$) then $y_{k+1} = y_k$ i.e. we plot the lower pixel (x_{k+1}, y_k)

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

if $P_k \geq 0$ ($d_1 - d_2 > 0$) then $y_{k+1} = y_k + 1$ i.e. we plot the upper pixel (x_{k+1}, y_{k+1})

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

Bresenham's Line Drawing Algorithm for $|m| < 1$

- i. Read x_a, y_a, x_b, y_b (Assume $-1 \leq m \leq 1$)
- ii. Load (x_0, y_0) into the frame buffer (i.e. plot the first point)
- iii. Calculate constants $\Delta y, \Delta x, 2\Delta y$ and $2\Delta y - 2\Delta x$
Obtain the first decision parameter $p_0 = 2\Delta y - \Delta x$

At each x_k along the line starting at $k = 0$ perform the tests:
If $p_k < 0$ then the next point to plot is $(x_k + 1, y_k)$ and
$$p_{k+1} = p_k + 2\Delta y$$

else the next point to plot is $(x_k + 1, y_k + 1)$ and
$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$
- v. Repeat step iv Δx times

Bresenham's Line Drawing Algorithm for $|m| < 1$

Advantages

- **Faster incremental algorithm**, makes use of integer arithmetic calculations only, avoids floating point computations
- Uses faster operations such as addition/subtraction and bit shifting
- **CPU intensive rounding off** operations are avoided

Disadvantages

- Used for drawing **basic lines**, antialiasing is not a part of this algorithm so for drawing smooth lines it is not suitable

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

if $P_k < 0$ ($d_1 - d_2 < 0$) then $y_{k+1} = y_k$ i.e. we plot the lower pixel $(x_k + 1, y_k)$

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

if $P_k \geq 0$ ($d_1 - d_2 \geq 0$) then $y_{k+1} = y_k + 1$ i.e. we plot the upper pixel $(x_k + 1, y_k + 1)$

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

Use Bresenham Line Algorithm to digitize a line with end points A(10,10) B(14,13)

$m = 3/4$ starting point(10,10)

positive slope , slope less than 1

$$P_0 = 2 \times 3 - 4 = 2 \quad \Delta y = 3 \quad \Delta x = 4$$

since slp is less than 1 $x_{k+1} = x_k + 1$ and $y_{k+1} = y_k + 1$ or y_k

P_k	$x_k + 1$	$y_k + 1$
$P_0 = 2$	11	11
$P_1 = 2 + 6 - 8 = 0$	12	12
$P_2 = 0 + 6 - 8 = -2$	13	12
$P_3 = -2 + 6 = 4$	14	13

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

if $P_k < 0$ ($d_1 - d_2 < 0$) then $y_{k+1} = y_k$ i.e. we plot the lower pixel $(x_k + 1, y_k)$

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

if $P_k \geq 0$ ($d_1 - d_2 \geq 0$) then $y_{k+1} = y_k + 1$ i.e. we plot the upper pixel $(x_k + 1, y_k + 1)$

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

Use BLA Algorithm to digitize a line with end points A(20,10) B(30,13)

$m = 3/10$ starting point(20,10)

positive slope , slope less than 1

$$P_0 = 6-10 = -4$$

$$\Delta y = 3 \quad \Delta x = 10$$

$$2\Delta y - 2\Delta x = 6-20 = -14$$

$$P_k$$

$$x_k + 1 \quad y_k + 1$$

$$P_0 = -4$$

$$21 \quad 10$$

$$P_1 = -4+6=2$$

$$22 \quad 11$$

$$P_2 = 2-14=-12$$

$$23 \quad 11$$

$$P_3 = -12+6=-6$$

$$24 \quad 11$$

$$P_4 = -6+6=0$$

$$25 \quad 12$$

$$P_5 = 0-14=-14$$

$$26 \quad 12$$

$$P_6 = -14+6=-8$$

$$27 \quad 12$$

$$P_7 = -8+6=-2$$

$$28 \quad 12$$

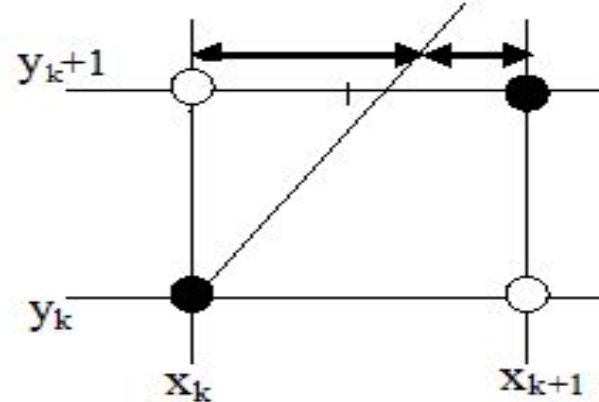
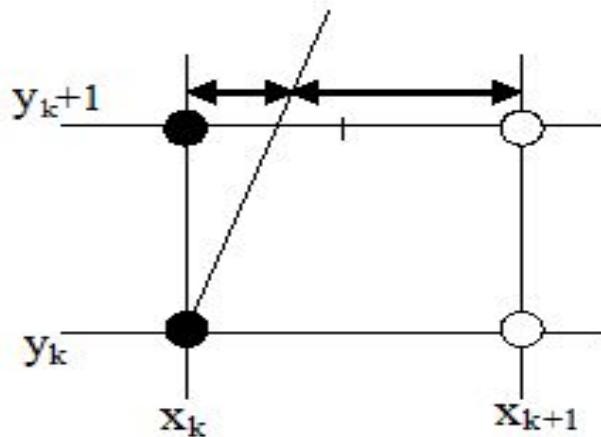
$$P_8 = -2+6=4$$

$$29 \quad 13$$

$$P_9 = 4-14=-12$$

$$30 \quad 13$$

Bresenham's Line Drawing Algorithm for Lines with Slope ≥ 1



$$y_{k+1} = mx + c$$

$$x = (y_{k+1} - c)/m$$

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

$$d_1 = x - x_k$$

$$d_2 = x_{k+1} - x$$

$$d_1 - d_2 = m$$

$$P_k =$$

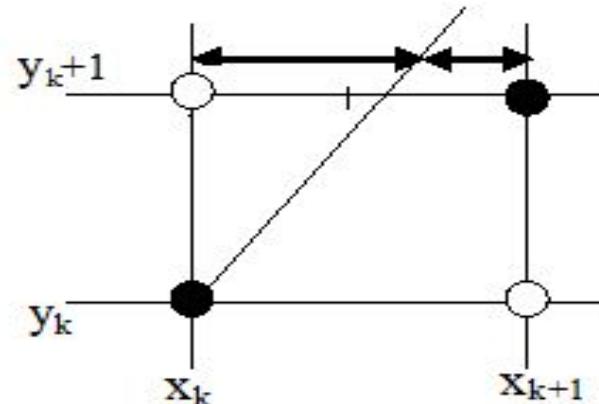
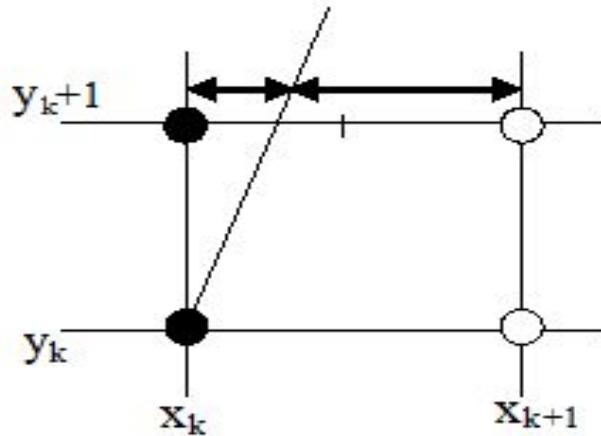
$$P_{k+1} =$$

$$P_{k+1} = P_k \quad x_{k+1} = x_k + 1 \text{ or } x_k$$

if $d_1 - d_2 > 0$ then the pixel at
'?' is closer to the line path
and ?

if $d_1 - d_2 < 0$ then the pixel at

Bresenham's Line Drawing Algorithm for Lines with Slope ≥ 1



At $y_k + 1$ e.q. of line $x = (y_k + 1 - c)/m$

$$d_1 = x - x_k = (y_k + 1 - c)/m - x_k \quad d_2 = x_{k+1} - (y_k + 1 - c)/m$$

$$d_1 - d_2 = (y_k + 1 - c)/m - x_k - x_k - 1 + (y_k + 1 - c)/m$$

$$\Delta y(d_1 - d_2) = 2\Delta x.y_k + 2\Delta x - 2\Delta x_c - 2\Delta y x_k - \Delta y$$

$$P_k = \Delta y(d_1 - d_2) = 2\Delta x.y_k - 2\Delta y x_k + 2\Delta x - 2\Delta x_c - \Delta y \quad \dots \text{(i)}$$

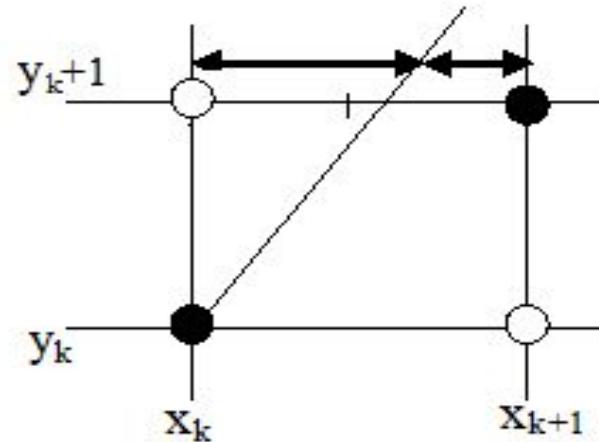
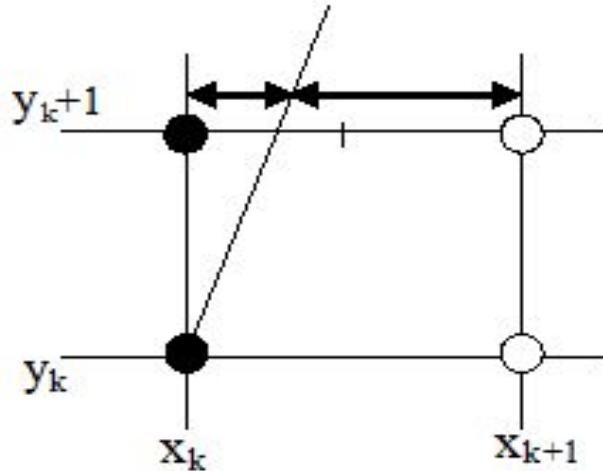
$$P_k = \Delta y(d_1 - d_2) = 2\Delta x.y_k - 2\Delta y x_k + b \quad \dots \text{(i)}$$

$$P_{k+1} = 2\Delta x.y_{k+1} - 2\Delta y x_{k+1} + b \quad \dots \text{(ii)}$$

$$P_{k+1} = P_k + 2\Delta x.y_{k+1} - 2\Delta x.y_k - 2\Delta y x_{k+1} + 2\Delta y x_k \quad \dots \text{(iii)}$$

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

Bresenham's Line Drawing Algorithm for Lines with Slope ≥ 1



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

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

if $P_k < 0$ ($d_1 - d_2 < 0$) then $x_{k+1} = x_k$ i.e. we plot the left pixel $(x_k, y_k + 1)$

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

if $P_k \geq 0$ ($d_1 - d_2 > 0$) then $x_{k+1} = x_k + 1$ i.e. we plot the upper pixel $(x_k + 1, y_k + 1)$

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

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

if $P_k < 0$ ($d_1 - d_2 < 0$) then $x_{k+1} = x_k$ i.e. we plot the left pixel $(x_k, y_k + 1)$

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

if $P_k \geq 0$ ($d_1 - d_2 \geq 0$) then $x_{k+1} = x_k + 1$ i.e. we plot the upper pixel $(x_k + 1, y_k + 1)$

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

Use BLA Algorithm to digitize a line with end points

A(10,13) B(14,18)

$m = 5/4$ starting point A(10,13) $\Delta x = 4$ $\Delta y = 5$

P_k	$x_k + 1$	$y_k + 1$
$P_0 = 8 - 5 = 3$	11	14
$P_1 = 3 + 8 - 10 = 1$	12	15
$P_2 = 1 - 2 = -1$	12	16
$P_3 = -1 + 8 = 7$	13	17
$P_4 = 7 + 8 - 10 = 5$	14	18

The algo runs $18 - 13 = 5$

Line Drawing

Use BIA Algorithm to digitize a line with end points

A(14,13) B(10,10)

m = starting point

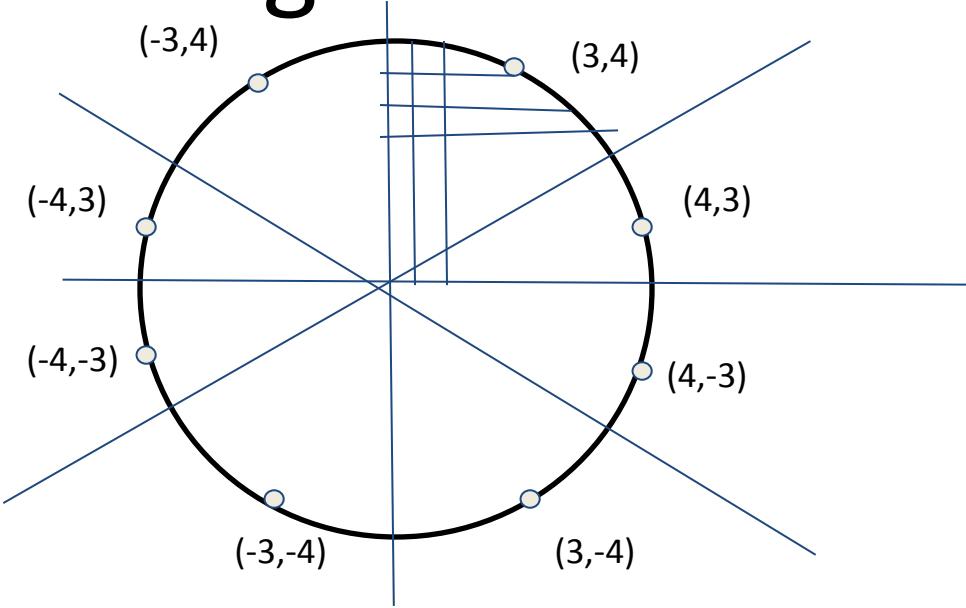
positive slope , slope

P_k

$x_k + 1$

$y_k + 1$

Midpoint Circle Algorithm



The equation of a circle is given by

$$x^2 + y^2 = r^2$$

To apply the midpoint method, we define a circle function

as $F_{\text{circle}}(x,y) = x^2 + y^2 - r^2$

now $F_{\text{circle}}(x,y)$

- < 0 if $P(x,y)$ is inside circle boundary
- = 0 if $P(x,y)$ is on circle boundary
- > 0 if $P(x,y)$ is outside circle boundary

This circle function $F_{\text{circle}}(x,y)$ serves as the decision parameter

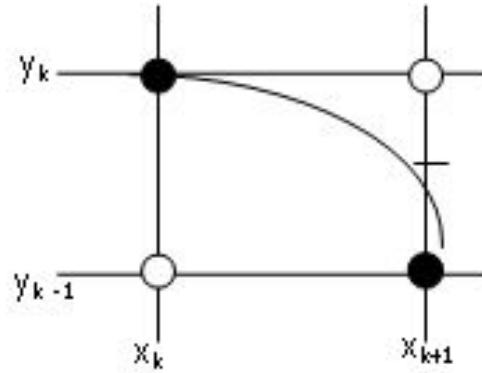
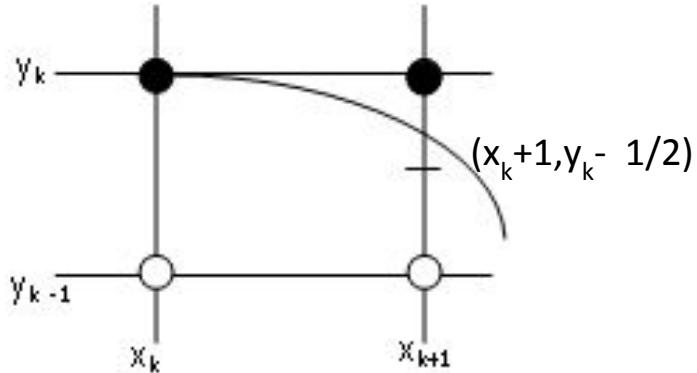
Assumptions:

Circle is at origin

Moving in clockwise direction

first Point's coordinate $(0,r)$ where r is radius of circle

Midpoint Circle Algorithm



Assuming position (x_k, y_k) has been selected at previous step we determine next position (x_k+1, y_{k+1}) as either (x_k+1, y_k) or $(x_k+1, y_k - 1)$ along circle path by evaluating the decision parameter (circle function).

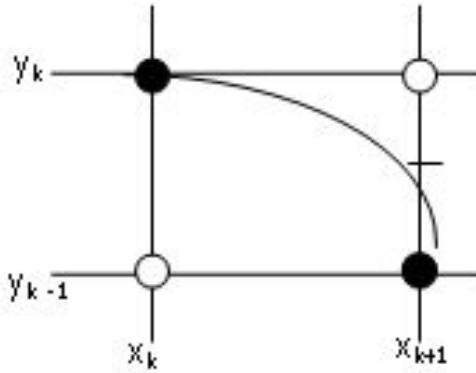
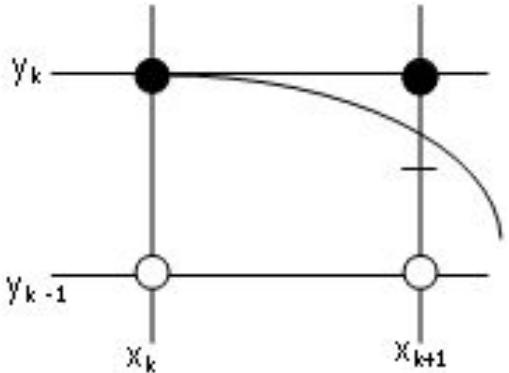
The decision parameter is the circle function evaluated at the midpoint between these two pixels

$$\text{At } k^{\text{th}} \text{ Step } P_k = F_{\text{circle}}(x_k+1, y_k - 1/2) = (x_k+1)^2 + (y_k - 1/2)^2 - r^2 \quad \dots \dots 1$$

At the next sampling position $(x_{k+1} + 1 = x_k + 2)$, the decision parameter is evaluated as

$$P_{k+1} = F_{\text{circle}}(x_{k+1} + 1, y_{k+1} - 1/2) = [(x_k+1)+1]^2 + (y_{k+1} - 1/2)^2 - r^2 \quad \dots \dots 2$$

Midpoint Circle Algorithm

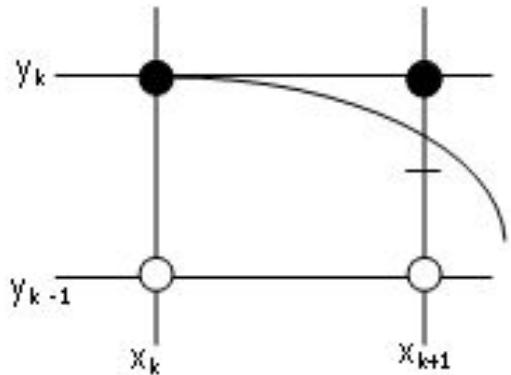


$$P_k = F_{\text{circle}}(x_k + 1, y_k - \frac{1}{2}) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \quad \dots \dots \dots \quad 1$$

Now subtracting eq (i) and (ii),

$$P_{k+1} = P_k + 2(x_k + 1) + 1 + (y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2 \quad \dots \dots \dots \quad 3$$

Midpoint Circle Algorithm



$$P_{k+1} = P_k + 2(x_k + 1) + 1 + (y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2 \quad \dots \dots \dots \quad 3$$

Case 1:

If $P_k < 0$ then the mid point is inside the circle, so pixel on scanline ' y_k ' is closer to the circle boundary and $y_{k+1} = y_k$

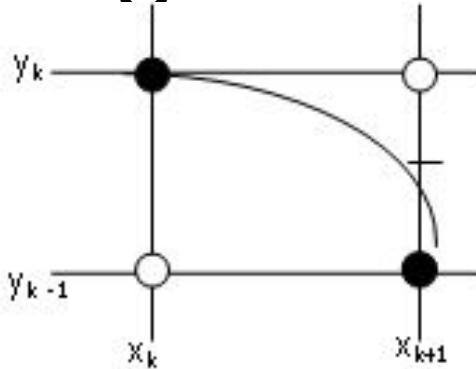
From equation (iii)

or $P_{k+1} = P_k + 2(x_k + 1) + 1 + (y_k - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2 \quad \dots \dots \dots \quad (a)$

or $P_{k+1} = P_k + 2(x_k + 1) + 1 \quad \dots \dots \dots \quad (a)$

or $P_{k+1} = P_k + 2x_{k+1} + 1 \quad \dots \dots \dots \quad (a)$ since $x_{k+1} = x_k + 1$

Midpoint Circle Algorithm



$$P_{k+1} = P_k + 2(x_k + 1) + (y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2 + 1 \quad \dots \dots \dots 3$$

Case 2:

If $P_k \geq 0$ then the mid point is outside circle, so pixel on scanline ' $y_k - 1$ ' is closer to the circle boundary and $y_{k+1} = y_k - 1$

From equation (iii)

or $P_{k+1} = P_k + 2(x_k + 1) + [(y_k - 3/2)^2 - (y_k - 1/2)^2 + 1] \dots (b)$

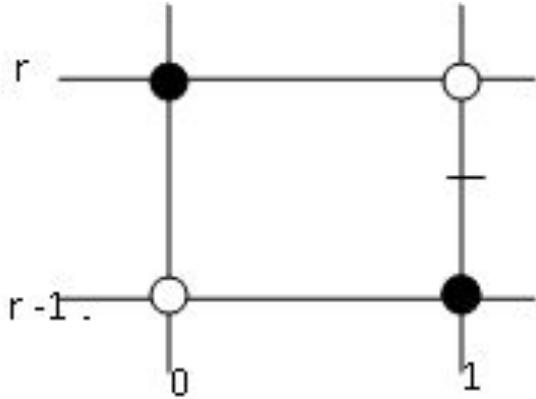
or $P_{k+1} = P_k + 2(x_k + 1) + [y_k^2 - 3y_k + 9/4 - y_k^2 + y_k - 1/4] + 1 \dots (b)$

or $P_{k+1} = P_k + 2(x_k + 1) + [-2y_k + 2] + 1 \dots (b)$

or $P_{k+1} = P_k + 2(x_k + 1) - 2(y_k - 1) + 1 \dots (b)$

or $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1 \dots (b) \quad x_{k+1} = x_k + 1 \quad y_{k+1} = y_k - 1$

Midpoint Circle Algorithm



Initial decision parameter P_0 is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$. Next pixel to plot is either $(1, r)$ or $(1, r-1)$.

So, midpoint coordinate position is $(1, r - \frac{1}{2})$

$$\begin{aligned} F_{\text{circle}}(1, r - \frac{1}{2}) &= 1^2 + (r - \frac{1}{2})^2 - r^2 \\ &= 1^2 + r^2 - r + \frac{1}{4} - r^2 \\ &= 1 - r + \frac{1}{4} \\ &= 1 + \frac{1}{4} - r \end{aligned}$$

$$P_0 = 5/4 - r = 1 - r$$

Midpoint Circle Algorithm

1. Input radius r and circle center (x_c, y_c) , and obtain the first point on the circumference of a circle centered on the origin as $(x_0, y_0) = (0, r)$
2. Calculate the initial value of the decision parameter as

$$P_0 = 5/4 - r$$

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

If $P_k < 0$, the next point along the circle centered on $(0,0)$ is $(x_k + 1, y_k)$ and

$$P_{k+1} = P_k + 2x_{k+1} + 1$$

If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and

$$P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$$

where $2x_{k+1} = 2x_k + 2$ and $2y_{k+1} = 2y_k - 2$.

4. Determine symmetry points in the other seven octants.

5. Move each calculated pixel position (x, y) onto circular path centered on (x_c, y_c) and plot the coordinate values:

$$x = x + x_c, \quad y = y + y_c$$

6. Repeat steps 3 through 5 until $x \geq y$.

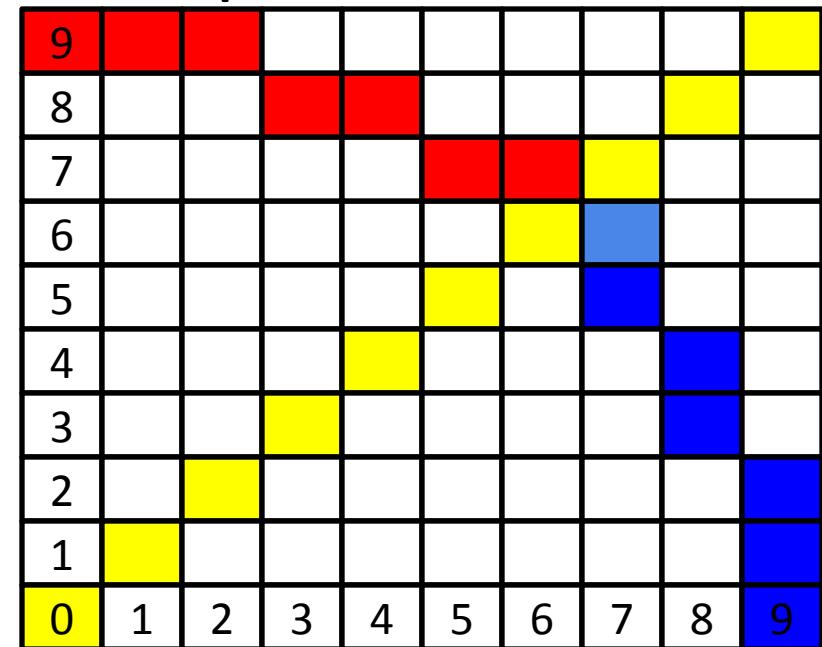
Midpoint Circle Algorithm

Digitize a circle with a radius of 9 pixels

Starting Pixel = (0,9)

$$P_0 = 1 - r =$$

P_k	x_{k+1}, y_{k+1}
-8	1,9
-5	2,9
0	3,8
-9	4,8
0	5,7
-3	6,7
10	7,6



If $P_k < 0$, the next point along the circle centered on (0,0) is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$
If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

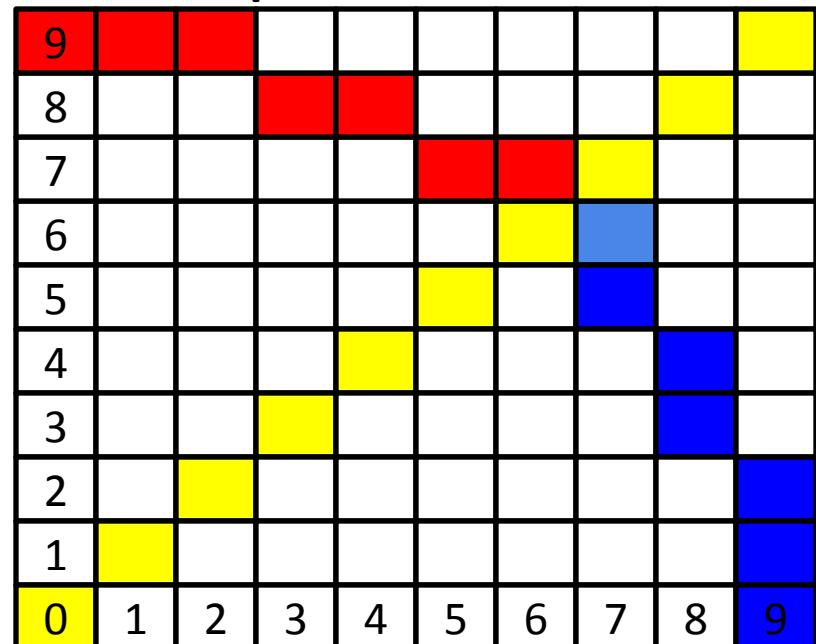
Midpoint Circle Algorithm

Digitize a circle with a radius of 9 pixels centered at (5,6)

Starting Pixel = (0,9)

$$P_0 = 1 - r =$$

P_k	x_{k+1}, y_{k+1}	x_{k+1}, y_{k+1}
-8	1,9	1+5,9+6
-5	2,9	2+5,9+6
0	3,8	3+5,8+6
-9	4,8	4+5,8+6
0	5,7	5+5,7+6
-3	6,7	6+5,7+6
10	7,6	7+5,6+6



If $P_k < 0$, the next point along the circle centered on (0,0) is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$
If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Midpoint Circle Algorithm

Digitize a circle with the equation $(x - 5)^2 + (y + 6)^2 = 25$

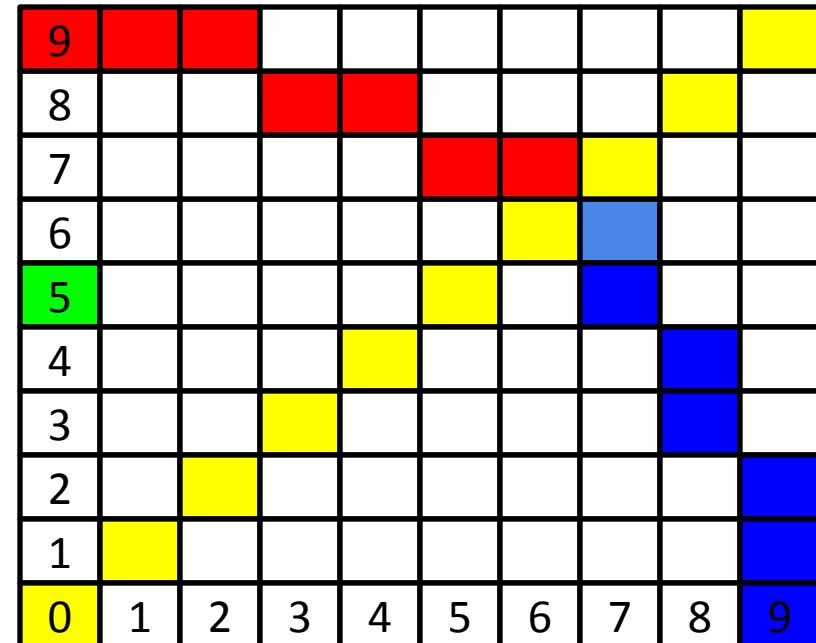
$r = 5$ circle centered at $(5, -6)$

Starting Pixel = $(0, 5)$

$$P_0 = 1 - r =$$

$$P_k$$

x_{k+1}, y_{k+1}



If $P_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$
If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Midpoint Circle Algorithm

Digitize a circle with the equation $(x - 5)^2 + (y + 6)^2 = 25$

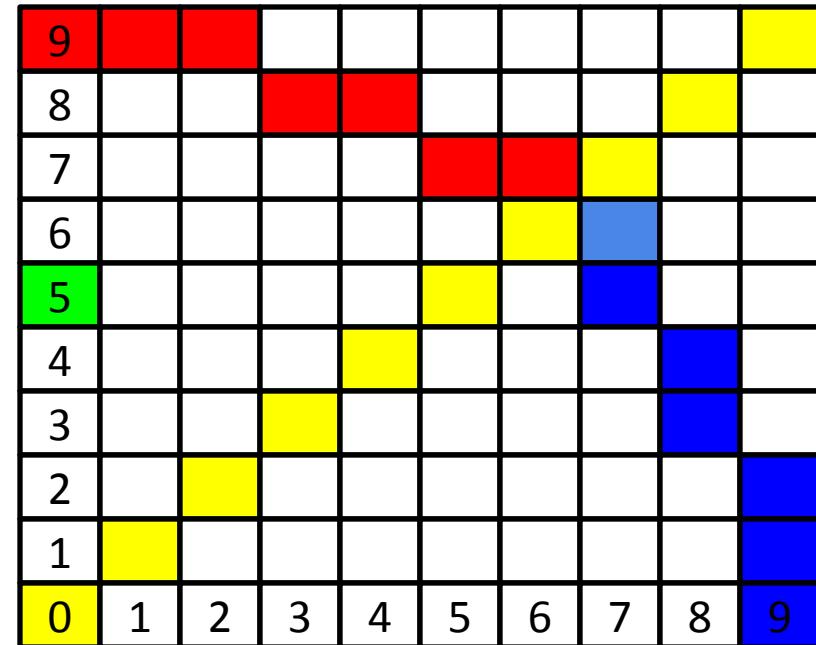
$r = 5$ circle centered at $(5, -6)$

Starting Pixel = $(0, r) = (0, 5)$

$$P_0 = 1 - r =$$

$$P_k$$

x_{k+1}, y_{k+1}



If $P_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$
If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Midpoint Circle Algorithm

Digitize a circle with the equation $(x - 5)^2 + (y + 6)^2 = 25$

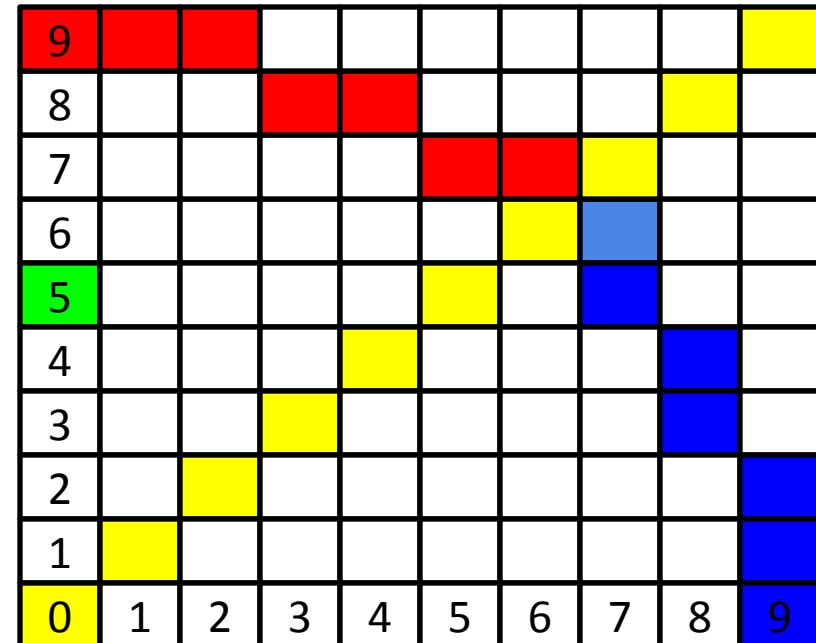
$r = 5$ circle centered at $(5, -6)$

Starting Pixel = $(0, 5)$

$$P_0 = 1 - r =$$

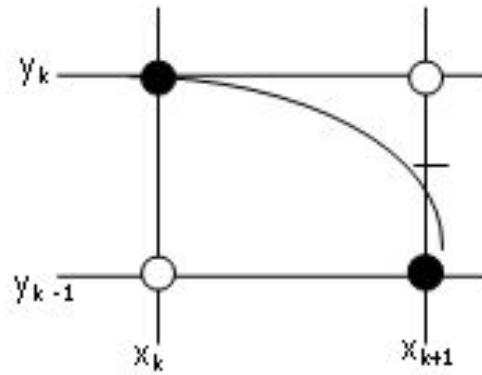
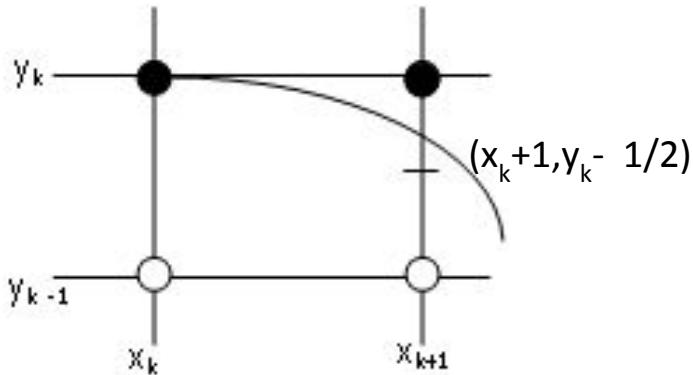
$$P_k$$

x_{k+1}, y_{k+1}



If $P_k < 0$, the next point along the circle centered on $(0,0)$ is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2x_{k+1} + 1$
If $P_k \geq 0$, the next point along the circle is $(x_k + 1, y_k - 1)$ and $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$

Midpoint Circle Algorithm



$$\text{At } k^{\text{th}} \text{ step } P_k = F_{\text{circle}}(x_k+1, y_k - 1/2) = (x_k+1)^2 + (y_k - 1/2)^2 - r^2 \quad \dots \text{ i}$$

$$\text{At } k+1^{\text{th}} \text{ Step } P_{k+1} = F_{\text{circle}}(x_{k+1}+1, y_{k+1} - 1/2) = [(x_k+1)+1]^2 + (y_{k+1} - 1/2)^2 - r^2 \quad \dots \text{ ii}$$

Now subtracting eq (i) and (ii),

$$P_{k+1} = P_k + 2(x_k+1) + 1 + (y_{k+1} - 1/2)^2 - (y_k - 1/2)^2 \quad \dots \text{ iii}$$

Case 1:

If $P_k < 0$ then mid point is inside the circle, so pixel on scanline ' y_k ' is closer to the circle boundary and $y_{k+1} = y_k$

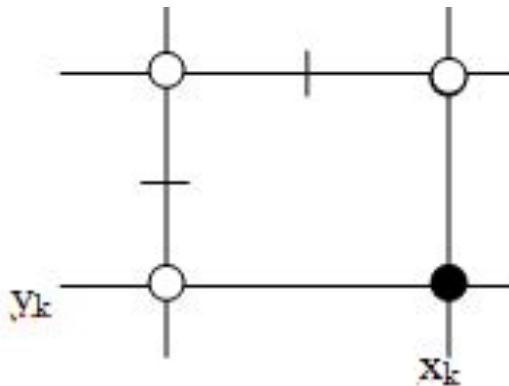
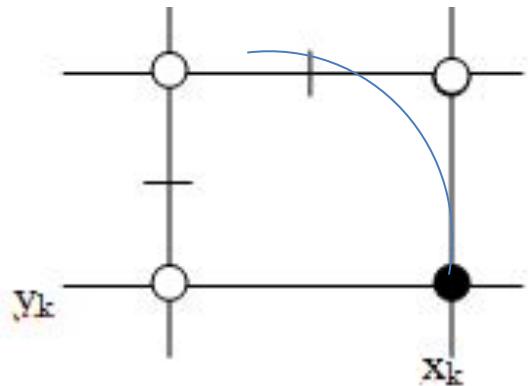
$$\text{or } P_{k+1} = P_k + 2x_{k+1} + 1 \quad \dots \text{ (a) since } x_{k+1} = x_k + 1$$

Case 2:

If $P_k > 0$ then mid point is outside circle, so pixel on scanline ' $y_k - 1$ ' is closer to the circle boundary and $y_{k+1} = y_k - 1$ From equation (iii)

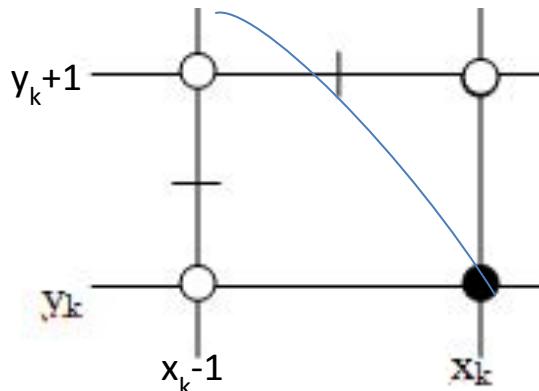
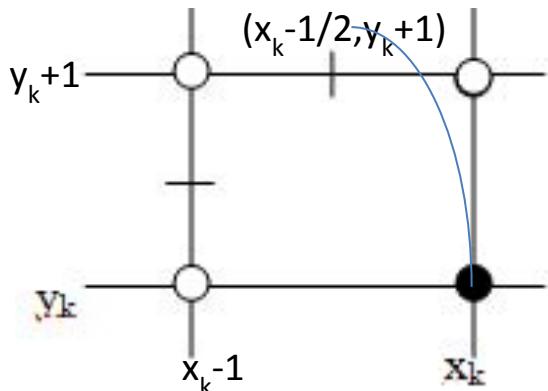
$$\text{or } P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1 \quad \dots \text{ (b) } x_{k+1} = x_k + 1 \quad y_{k+1} = y_k - 1$$

Midpoint Circle Algorithm



Starting point is $(r,0)$ moving in anti clockwise direction

Midpoint Circle Algorithm



$$\text{At } k^{\text{th}} \text{ Step } P_k = F_{\text{circle}}(x_k - \frac{1}{2}, y_k + 1) = (x_k - \frac{1}{2})^2 + (y_k + 1)^2 - r^2 \dots \dots \text{i}$$

$$\begin{aligned} \text{At } k+1^{\text{th}} \text{ Step } P_{k+1} &= F_{\text{circle}}(x_{k+1} - \frac{1}{2}, y_{k+1} + 1) = (x_{k+1} - \frac{1}{2})^2 + [(y_k + 1) + 1]^2 - r^2 \dots \dots \text{ii} \\ &= (x_{k+1} - \frac{1}{2})^2 + (y_k + 1)^2 + 2(y_k + 1) + 1 - r^2 \end{aligned}$$

Now subtracting eq (i) and (ii),

$$P_{k+1} = P_k + 2(y_k + 1) + 1 + (x_{k+1} - \frac{1}{2})^2 - (x_k - \frac{1}{2})^2 \dots \dots \text{iii}$$

Case 1:

If $P_k < 0$ then mid point is inside the circle, so pixel on scanline ' x_k ' is closer to the circle boundary and $x_{k+1} = x_k$

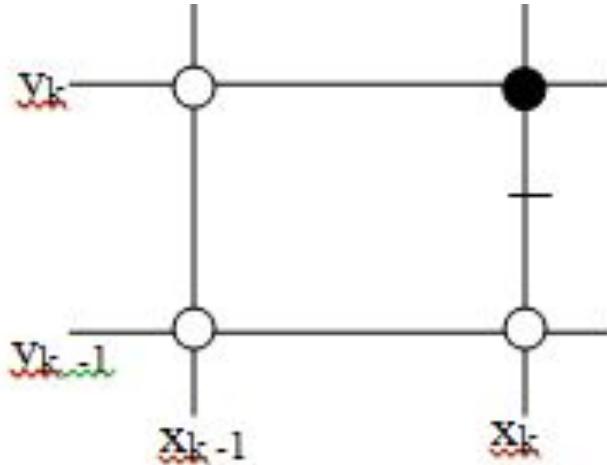
$$\text{or } P_{k+1} = P_k + 2y_{k+1} + 1 \dots \dots \text{(a) since } y_{k+1} = y_k + 1$$

Case 2:

If $P_k > 0$ then mid point is outside circle, so pixel on scanline ' $x_k - 1$ ' is closer to the circle boundary and $x_{k+1} = x_k - 1$ From equation (iii)

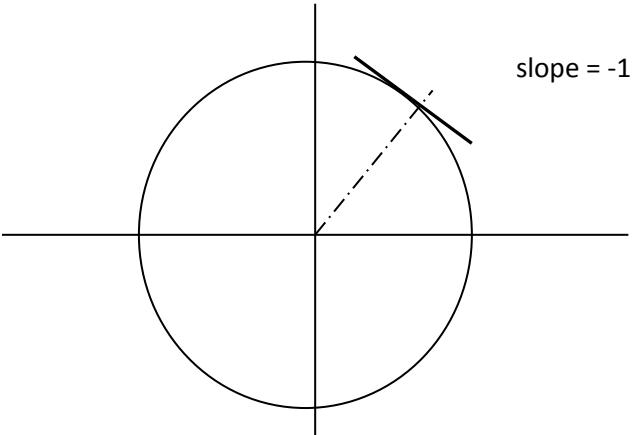
$$\text{or } P_{k+1} = P_k + 2y_{k+1} - 2x_{k+1} + 1 \dots \dots \text{(b) } x_{k+1} = x_k - 1 \quad y_{k+1} = y_k - 1$$

Midpoint Circle Algorithm



Starting point is $(r,0)$ moving in anti clockwise direction
Use the algorithm to digitize a circle with a radius of 9 pixels

Midpoint Ellipse Algorithm



Mid point ellipse method is applied throughout first quadrant in two parts (according to the slope of ellipse)

The equation of an ellipse is given by

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

or $F_{\text{ellipse}}(x,y) = r_x^2 x^2 + r_y^2 y^2 - r_x^2 r_y^2$

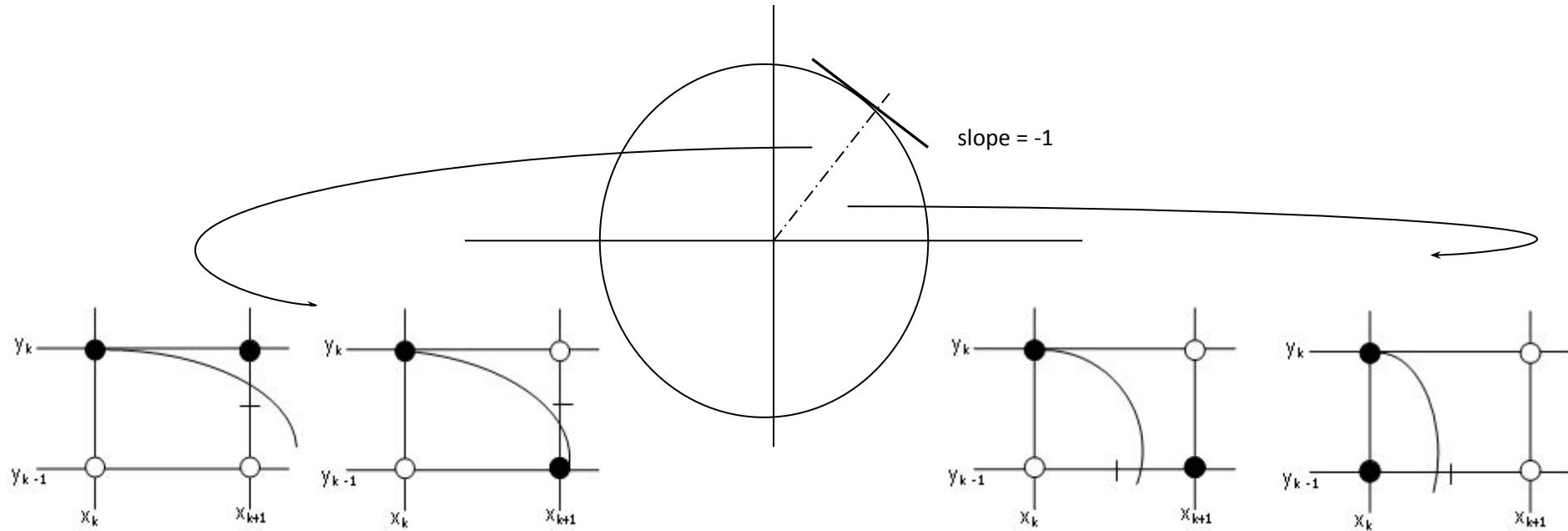
now $F_{\text{ellipse}}(x,y) < 0$ if (x,y) is inside the ellipse boundary

$= 0$ if (x,y) is on the ellipse boundary

> 0 if (x,y) is outside the ellipse boundary

This ellipse function $F_{\text{ellipse}}(x,y)$ serves as the decision parameter

Midpoint Ellipse Algorithm



$$P1_k = F_{\text{ellipse}}(x_k + 1, y_k - \frac{1}{2}) \\ = r_y^2 (x_k + 1)^2 + r_x^2 (y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$P2_k = F_{\text{ellipse}}(x_k + \frac{1}{2}, y_k - 1) \\ = r_y^2 (x_k + \frac{1}{2})^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$$

$$P1_{k+1} = F_{\text{ellipse}}(x_{k+1} + 1, y_{k+1} - \frac{1}{2}) \\ = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 (y_{k+1} - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$P2_{k+1} = F_{\text{ellipse}}(x_{k+1} + \frac{1}{2}, y_{k+1} - 1) \\ = r_y^2 (x_{k+1} + 1/2)^2 + r_x^2 [(y - 1) - 1]^2 - r_x^2 r_y^2$$

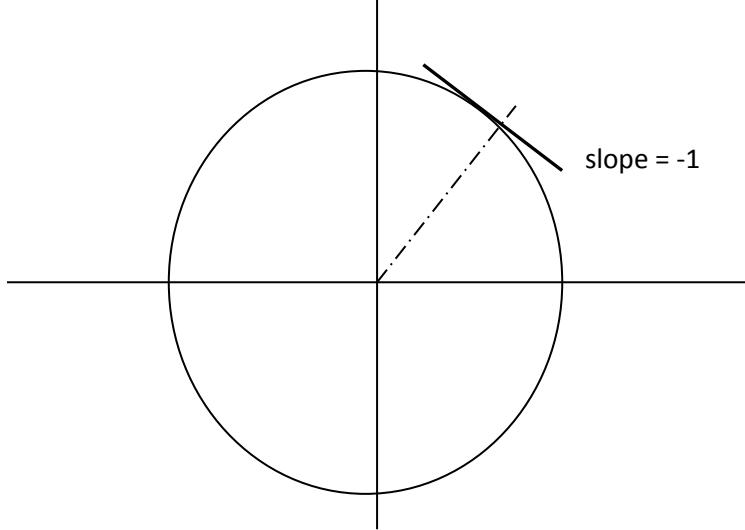
$$P1_{k+1} = P1_k + \dots$$

$$P2_{k+1} = P2_k - \dots$$

$$P1_0 = ?$$

$$P2_0 = ?$$

Midpoint Ellipse Algorithm



$$r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2 = 0$$

The ellipse slope is given by slope = -1

$$2 r_y^2 x + 2 r_x^2 y \frac{dy}{dx} = 0$$

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

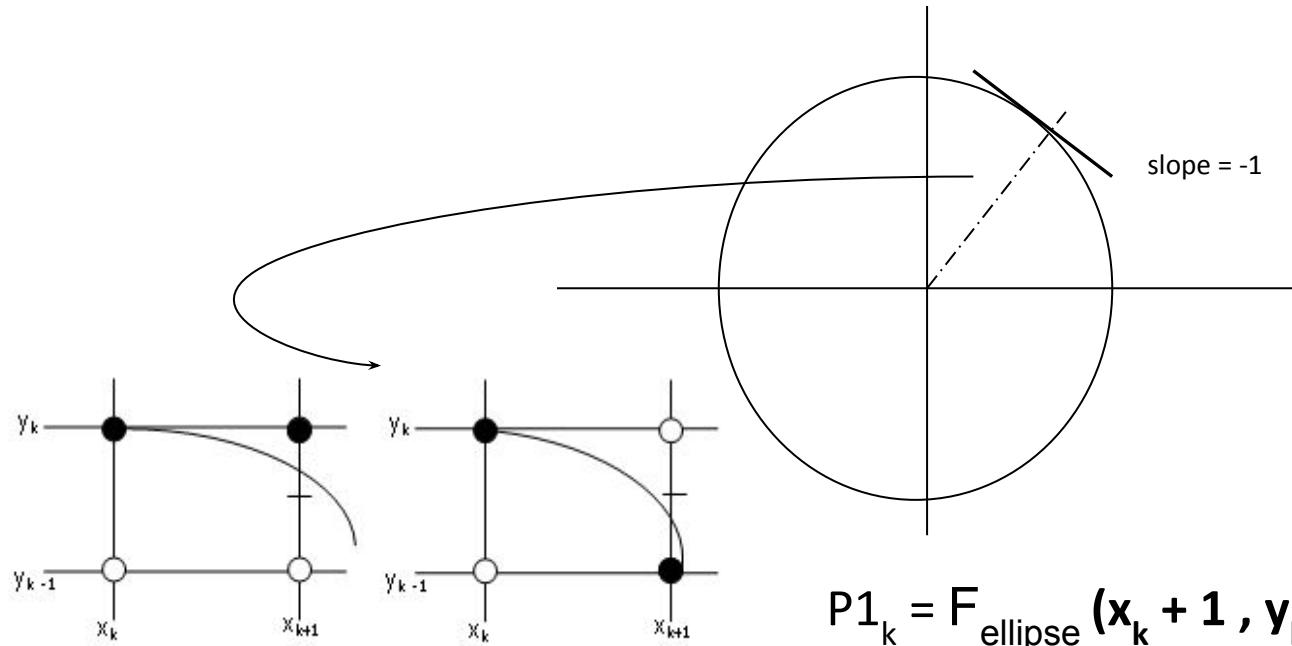
At the boundary between region 1 and 2 $\frac{dy}{dx} = -1$

$$\text{so, } 2 r_y^2 x = 2 r_x^2 y$$

and we move out of the region 1 when

$$2 r_y^2 x >= 2 r_x^2 y$$

Midpoint Ellipse Algorithm



Region 1.

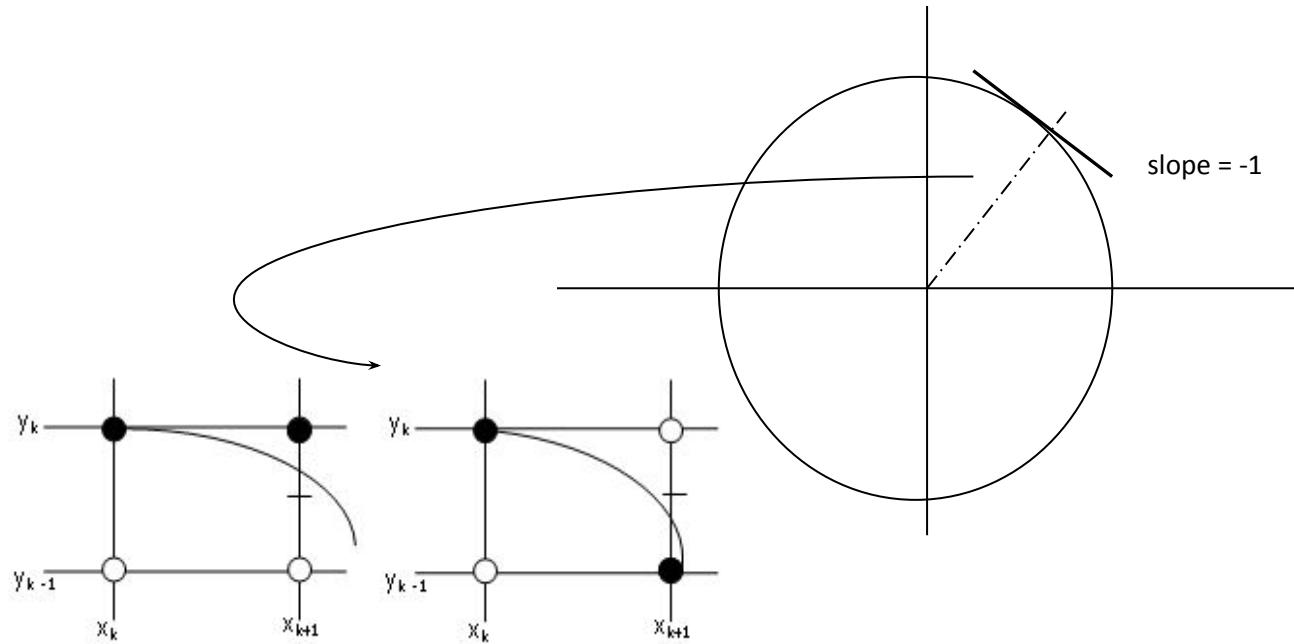
Sample at unit steps in 'x' direction, the midpoint is taken between horizontal pixels at each step now. Assuming, (x_k, y_k) has been plotted, next pixel to plot is (x_{k+1}, y_{k+1}) where y_{k+1} is either y_k or $y_k - 1$ and x_{k+1} is $x_k + 1$ i.e. choose either (x_{k+1}, y_k) or $(x_{k+1}, y_k - 1)$

$$P_{1_k} = F_{\text{ellipse}}(x_k + 1, y_k - \frac{1}{2}) \\ = r_y^2 (x_k + 1)^2 + r_x^2 (y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$P_{1_{k+1}} = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 (y_{k+1} - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$P_{1_{k+1}} = P_{1_k} + \dots$$

Midpoint Ellipse Algorithm



$$\begin{aligned}
 P1_k &= F_{\text{ellipse}}(x_k + 1, y_k - \frac{1}{2}) \\
 &= r_y^2 (x_k + 1)^2 + r_x^2 (y_k - \frac{1}{2})^2 - r_x^2 r_y^2
 \end{aligned}$$

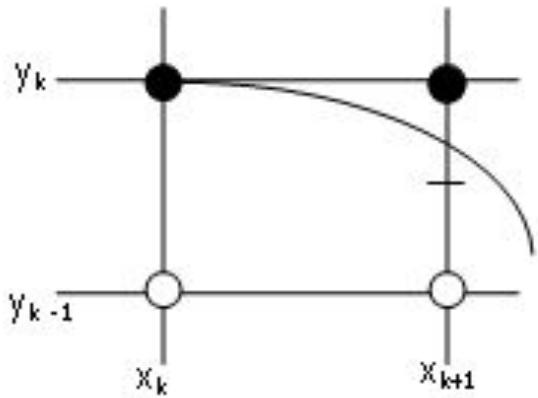
$$P1_{k+1} = r_y^2 [(x_k + 1) + 1]^2 + r_x^2 (y_{k+1} - \frac{1}{2})^2 - r_x^2 r_y^2$$

Now subtracting eq (i) and (ii),

$$P1_{k+1} = P1_k + 2r_y^2 (x_k + 1) + r_y^2 + r_x^2 [(y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2] \quad (iii)$$

where y_{k+1} is either y_k or $y_k - 1$ depending on the sign of $P1_k$.

Midpoint Ellipse Algorithm



Now subtracting eq (i) and (ii),

$$P1_{k+1} = P1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2] \quad (\text{iii})$$

where y_{k+1} is either y_k or y_{k-1} depending on the sign of $P1_k$.

Case 1:

if $P1_k < 0$ then the mid point is inside the ellipse, so pixel on scanline

' y_k ' is closer to the ellipse boundary and $y_{k+1} = y_k$

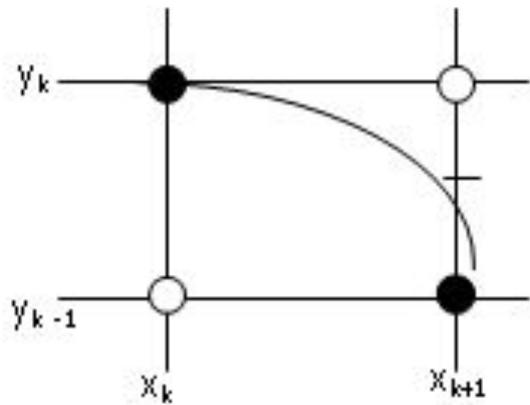
so the increment will be $2r_y^2 x_{k+1} + r_y^2$

i.e. from equation (iii)

$$\text{or } P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2 \quad (\text{a})$$

$$\begin{aligned} \text{Where } x_{k+1} &= x_k + 1 \\ \text{or } 2r_y^2 x_{k+1} &= 2r_y^2 x_k + 2r_y^2 \end{aligned}$$

Midpoint Ellipse Algorithm



Now subtracting eq (i) and (ii),

$$P1_{k+1} = P1_k + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2] \quad (\text{iii})$$

where y_{k+1} is either y_k or y_{k-1} depending on the sign of $P1_k$.

Case 2:

if $P1_k \geq 0$ then the mid point is outside or on the boundary of the ellipse, so we select the pixel on scan line ' $y_k - 1$ ' then $y_{k+1} = y_k - 1$

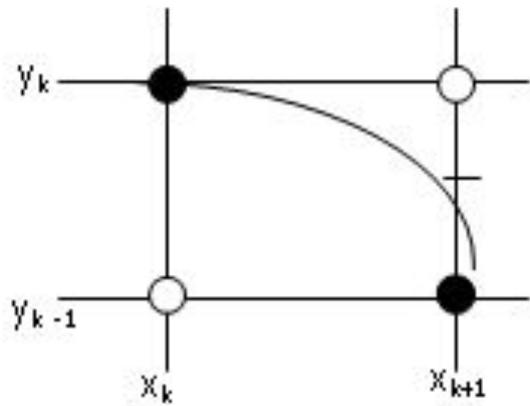
so the increment will be $2r_y^2 x_{k+1} - 2r_x^2 y_{k+1}$

i.e. from equation (iii)

$$\text{or } P1_{k+1} = P1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2 \quad (\text{b})$$

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

Midpoint Ellipse Algorithm



Case 2:

$$P_{1_{k+1}} = P_{1_k} + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y_{k+1} - \frac{1}{2})^2 - (y_k - \frac{1}{2})^2] \quad (\text{iii})$$

where y_{k+1} is either y_k or y_{k-1} depending on the sign of P_{1_k} .

$$\text{Here } y_{k+1} = y_k - 1$$

$$P_{1_{k+1}} = P_{1_k} + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y_k - 3/2)^2 - (y_k - \frac{1}{2})^2] \quad (\text{iii})$$

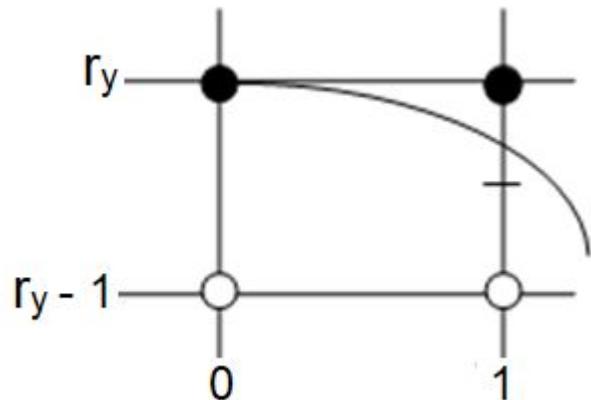
$$P_{1_{k+1}} = P_{1_k} + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(y_k^2 - 3y_k + 9/4 - y_k^2 + y_k - 1/4)] \quad (\text{iii})$$

$$P_{1_{k+1}} = P_{1_k} + 2r_y^2(x_k + 1) + r_y^2 + r_x^2[(-3y_k + y_k + 9/4 - 1/4)] \quad (\text{iii})$$

$$P_{1_{k+1}} = P_{1_k} + 2r_y^2(x_k + 1) + r_y^2 - 2r_x^2[y_k - 1] \quad (\text{iii})$$

$$\text{or } P_{1_{k+1}} = P_{1_k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2 \quad (\text{b})$$

Midpoint Ellipse Algorithm



Initial decision parameter for Region 1 = $P1_0$

The starting position is $(0, r_y)$

Next pixel to plot is either $(1, r_y)$ or $(1, r_y - 1)$

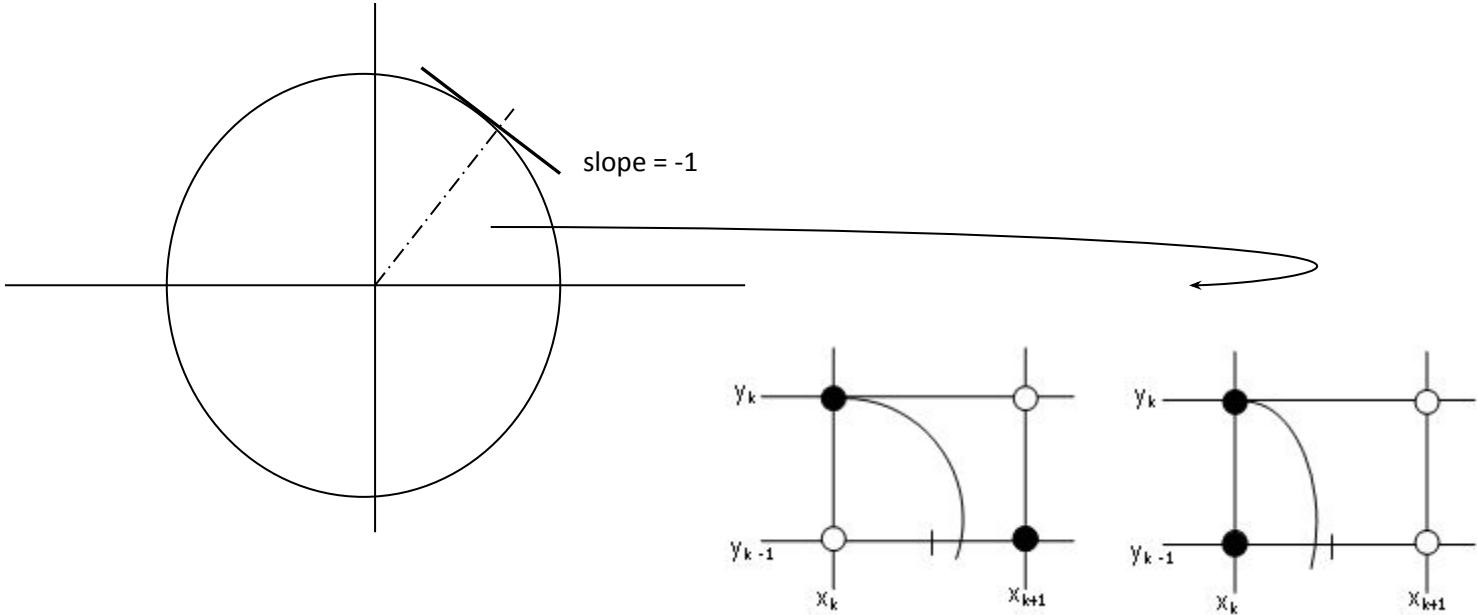
So, midpoint coordinate position is $(1, r_y - \frac{1}{2})$

$$F_{\text{ellipse}}(1, r_y - \frac{1}{2}) = r_y^2 + r_x^2 (r_y - \frac{1}{2})^2 - r_y^2 r_x^2$$

Thus,

$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 r_y^2$$

Midpoint Ellipse Algorithm



$$P_{2k} = F_{\text{ellipse}}(x_k + \frac{1}{2}, y_k - 1) \\ = r_y^2 (x_k + \frac{1}{2})^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$$

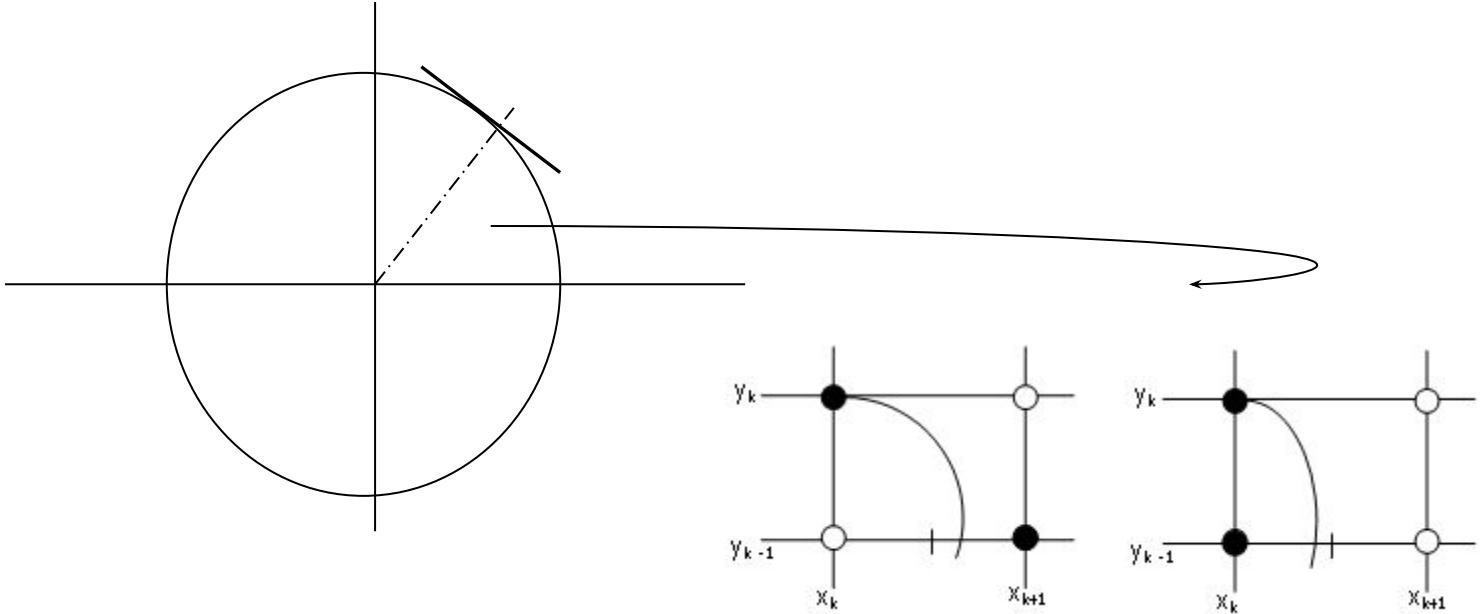
$$P_{2k+1} = r_y^2 (x_{k+1} + 1/2)^2 + r_x^2 [(y_k - 1)^2 - r_x^2 r_y^2]$$

$$P_{2k+1} = P_{2k} - \dots$$

Region 2.

Sample at unit steps in 'y' direction, the midpoint is taken between horizontal pixels at each step now. Assuming, (x_k, y_k) has been plotted, next pixel to plot is (x_{k+1}, y_{k+1}) where x_{k+1} is either x_k or $x_k + 1$ and y_{k+1} is $y_k - 1$ i.e. choose either $(x_k, y_k - 1)$ or $(x_{k+1}, y_k - 1)$

Midpoint Ellipse Algorithm



$$\begin{aligned}
 P2_k &= (x_k + \frac{1}{2}, y_k - 1) \\
 &= r_y^2 (x_k + \frac{1}{2})^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2
 \end{aligned}$$

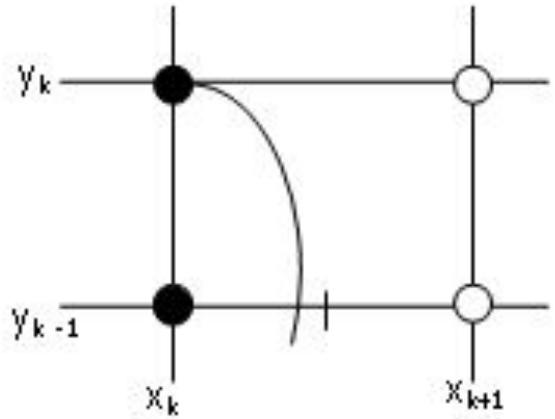
$$P2_{k+1} = r_y^2 (x_{k+1} + 1/2)^2 + r_x^2 [(y - 1) - 1]^2 - r_x^2 r_y^2$$

Now subtracting eq (i) and (ii),

$$P2_{k+1} = P2_k - 2r_x^2 (y_k - 1) + r_x^2 + r_y^2 [(x_{k+1} + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2] \quad (vi)$$

where x_{k+1} is either x_k or $x_k + 1$ depending on the sign of $P2_k$.

Midpoint Ellipse Algorithm

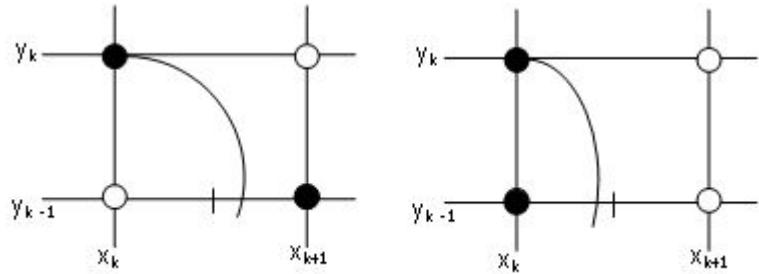


$$P_{2_{k+1}} = P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [(x_{k+1} + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2] \quad (\text{vi})$$

Case 1: if $P_{2_k} > 0$ then the mid point is outside the boundary of the ellipse, so we select the pixel at ' x_k '

Or $P_{2_{k+1}} = P_{2_k} - 2r_x^2(y_k - 1) + r_x^2$
 $= P_{2_k} - 2r_x^2 y_{k+1} + r_x^2$ (c) Where $y_{k+1} = y_k - 1$
 $2r_x^2 y_{k+1} = 2r_x^2 y_k - 2 r_x^2$

Midpoint Ellipse Algorithm



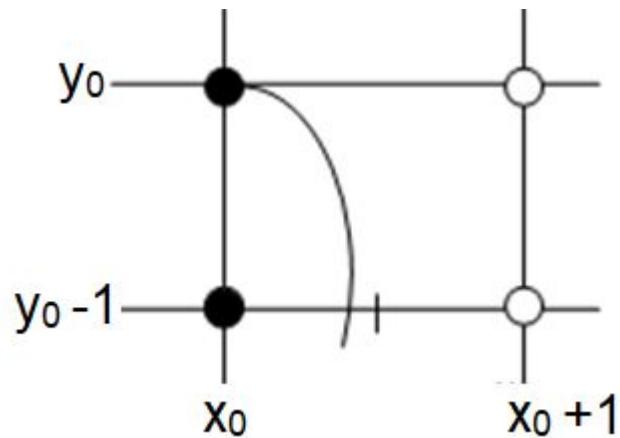
$$P_{2_{k+1}} = P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [(x_{k+1} + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2] \quad (\text{vi})$$

Case 2: if $P_{2_k} \leq 0$ then the mid point is inside or on the boundary of the ellipse , so we select pixel at ' $x_k + 1$ '
 i.e. from equation (vi)

$$\begin{aligned} \text{or } P_{2_{k+1}} &= P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [(x_{k+1} + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2] \\ &= P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [(x_k + 1 + \frac{1}{2})^2 - (x_k + \frac{1}{2})^2] \\ &= P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [(x_k + 3/2)^2 - (x_k + \frac{1}{2})^2] \\ &= P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [x_k^2 + 3x_k + 9/4 - x_k^2 - x - 1/4] \\ &= P_{2_k} - 2r_x^2(y_k - 1) + r_x^2 + r_y^2 [2x_k + 2] \\ &= P_{2_k} - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1} \end{aligned} \quad (\text{d})$$

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

Midpoint Ellipse Algorithm



For region 2, the initial position (x_0, y_0) is taken as the last position selected in region 1 and thus the initial decision parameter in region 2 is

$$\begin{aligned} P_{2_0} &= F_{\text{ellipse}}(x_0 + \frac{1}{2}, y_0 - 1) \\ &= r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2 \end{aligned}$$

Midpoint Ellipse Algorithm

1. Input r_x , r_y , and obtain the first point on an ellipse centered on the origin as
 $(x_0, y_0) = (0, r_y)$

2. Calculate the initial value of the decision parameter in region 1 as

$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 r_y$$

3. At each x , position in region 1, starting at $k = 0$, perform the following test:

If $P1_k < 0$, the next point along the ellipse centered on $(0, 0)$ is (x_{k+1}, y_k) and

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is or $(x_{k+1}, y_k - 1)$ and with

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

and continue until $2r_y^2 x \geq 2r_x^2 y$

4. Calculate the initial value of the decision parameter in region 2 using the last point (x_0, y_0) calculated in region 1 as

$$P2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in region 2, starting at $k = 0$, perform the following test:

If $P2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

6. Determine symmetry points in the other three quadrants.

7. Repeat the steps for region 2 until the value of 'y' becomes zero

Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 6$ pixels and $r_x = 8$ pixels

Starting Pixel = (0,6)

$$P1_0 = 36 + (0.25 \times 64) - 64 \times 6 = -332$$

K $P1_k$ x_{k+1}, y_{k+1} $2 r_y^2 x >= 2 r_x^2 y ?$

0 $P1_0 = -332$ 1,6 $2 \times 36 \times 1 = 72 >= 2 \times 64 \times 6 = 768$ False

1 $P1_1 = -224$ 2,6 $2 \times 36 \times 2 = 144 >= 2 \times 64 \times 6 = 768$ False

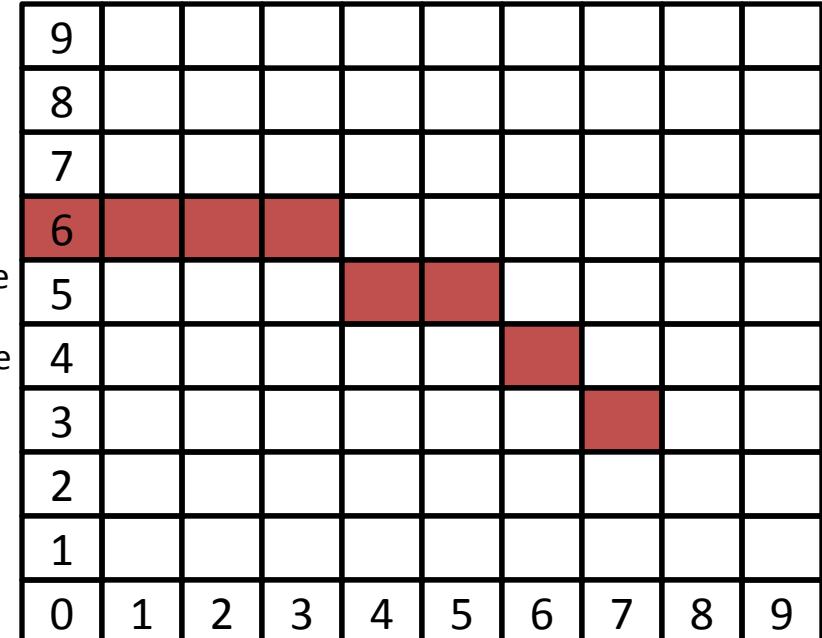
2 $P1_2 = -44$ 3,6 $2 \times 36 \times 3 = 216 >= 2 \times 64 \times 6 = 768$ False

3 $P1_3 = 208$ 4,5 $2 \times 36 \times 4 = 288 >= 2 \times 64 \times 5 = 640$ False

4 $P1_4 = -108$ 5,5 $2 \times 36 \times 5 = 360 >= 2 \times 64 \times 5 = 640$ False

5 $P1_5 = 288$ 6,4 $2 \times 36 \times 6 = 432 >= 2 \times 64 \times 4 = 512$ False

6 $P1_6 = 244$ 7,3 $2 \times 36 \times 7 = 504 >= 2 \times 64 \times 3 = 384$ True



$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 r_y$$

If $P1_k < 0$, the next point along the ellipse centered on (0, 0) is (x_{k+1}, y_k) and

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is or $(x_{k+1}, y_k - 1)$ and with

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 6$ pixels and $r_x = 8$ pixels

Starting Pixel = (0,6)

$$P1_0 = -332$$

K	$P1_k$	x_{k+1}, y_{k+1}	$2 r_y^2 x >= 2 r_x^2 y ?$
0	-332	1,6	72 768
1	-224	2,6	144 768
2	-44	3,7	216 768
3	208	4,5	288 640
4	-108	5,5	360 640
5	288	6,4	432 512
6	244	7,3	504 384

9									
8									
7									
6									
5									
4									
3									
2									
1									
0	1	2	3	4	5	6	7	8	9

$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 r_y$$

If $P1_k < 0$, the next point along the ellipse centered on (0, 0) is (x_{k+1}, y_k) and

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is or $(x_{k+1}, y_k - 1)$ and with

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

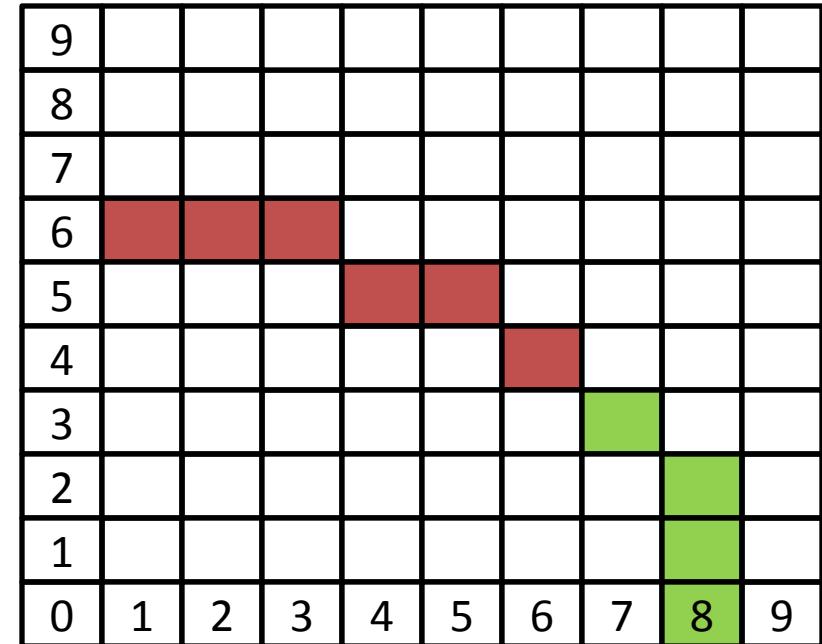
Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 6$ pixels and $r_x = 8$ pixels

Starting Pixel $(x_0, y_0) = (7,3)$

$$P2_0 = -23$$

K	$P2_k$	x_{k+1}, y_{k+1}
0	-23	8,2
1	361	8,1
2	297	8,0



$$P2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

At each y_k position in region 2, starting at $k = 0$, perform the following test:

If $P2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 6$ pixels and $r_x = 8$ pixels

Starting Pixel = (7,3)

$$P2_0 = -151$$

$$K \quad P2_k \quad x_{k+1}, y_{k+1}$$

$$0 \quad -151 \quad 8,2$$

$$1 \quad 233 \quad 8,1$$

$$2 \quad 745 \quad 8,0$$

9									
8									
7									
6									
5									
4									
3									
2									
1									
0	1	2	3	4	5	6	7	8	9

$$P2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

At each y_k position in region 2, starting at $k = 0$, perform the following test:

If $P2_k > 0$, the next point along the ellipse centered on (0, 0) is $(x_k, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 8$ pixels and $r_x = 6$ pixels

Starting Pixel = (0,8)

$$P1_0 = -215$$

$$K \quad P1_k \quad x_{k+1}, y_{k+1} \quad 2r_y^2 x \geq 2r_x^2 y ?$$

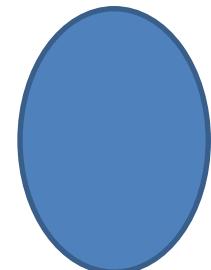
$$0 \quad -215 \quad 1,8 \quad 128 > 576 \text{ False}$$

$$1 \quad -23 \quad 2,8 \quad 256 > 576 \text{ False}$$

$$2 \quad 297 \quad 3,7 \quad 384 > 504 \text{ False}$$

$$3 \quad 113 \quad 4,6 \quad \mathbf{512 > 432 True}$$

9									
8									
7									
6									
5									
4									
3									
2									
1									
0	1	2	3	4	5	6	7	8	9



$$P1_0 = r_y^2 + \frac{1}{4} r_x^2 - r_x^2 r_y$$

If $P1_k < 0$, the next point along the ellipse centered on (0, 0) is (x_{k+1}, y_k) and

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise, the next point along the ellipse is or $(x_{k+1}, y_k - 1)$ and with

$$P1_{k+1} = P1_k + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

Midpoint Ellipse Algorithm

Digitize an Ellipse with $r_y = 8$ pixels and $r_x = 6$ pixels

Starting Pixel $(x_0, y_0) = (4, 6)$

$$P2_0 = -108$$

$$\begin{array}{ll} K & P2_k \\ \hline 0 & -108 \end{array}$$

$$x_{k+1}, y_{k+1}$$

$$5, 5$$

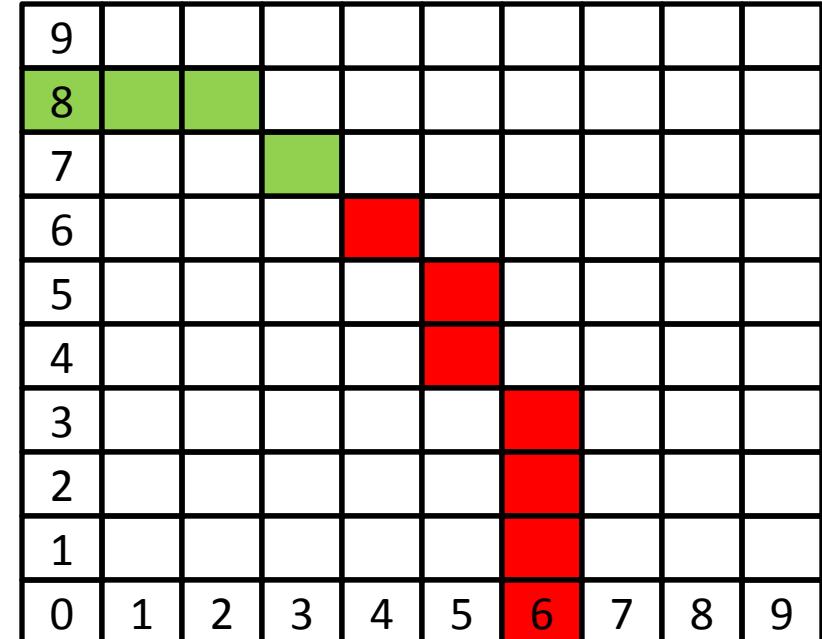
$$1 \quad 208 \quad 5, 4$$

$$2 \quad -44 \quad 6, 3$$

$$3 \quad 544 \quad 6, 2$$

$$4 \quad 436 \quad 6, 1$$

$$5 \quad 400 \quad 6, 0$$



$$P2_0 = r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

At each y_k position in region 2, starting at $k = 0$, perform the following test:

If $P2_k > 0$, the next point along the ellipse centered on $(0, 0)$ is $(x_k, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2$$

Otherwise, the next point along the ellipse is $(x_k + 1, y_k - 1)$ and

$$P2_{k+1} = P2_k - 2r_x^2 y_{k+1} + r_x^2 + 2r_y^2 x_{k+1}$$

Filled Area Primitives

Standard output primitive in graphics packages is solid color ,patterned polygon area

Polygons are easier to process due to linear boundaries

Two basic approaches to area filling on a raster system

- 1. Determine the overlap intervals for scan lines that cross the area.
Typically useful for filling polygons, circles, ellipses**

- 2. Start from a given interior position and paint outwards from
this point until we encounter the specified boundary conditions
useful for filling more complex boundaries, interactive painting
system.**

Filling rectangles

Two things to consider

- i. which pixels **to fill**
- ii. with what value **to fill**

Move along scan line (from left to right) that intersect the primitive and fill in pixels that lay inside

To fill rectangle with solid color

Set each pixel lying on scan line running from left edge to right with same pixel value, each span from x_{max} to x_{min}

```
for( y from  $y_{min}$  to  $y_{max}$  of rectangle)
    for( x from  $x_{min}$  to  $x_{max}$  of rectangle)
        writePixel(x, y, value);
```

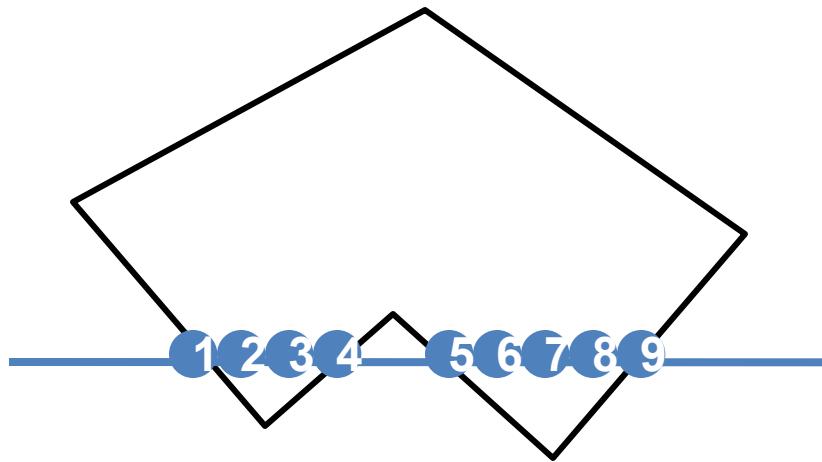
*/*scan line*/*
*/*by pixel*/*

Filling Polygons

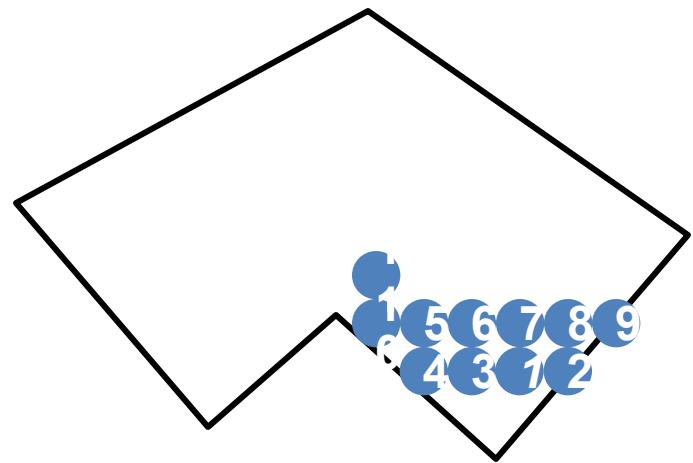
Filling Polygons

Scan-line fill algorithm

Inside-Outside tests



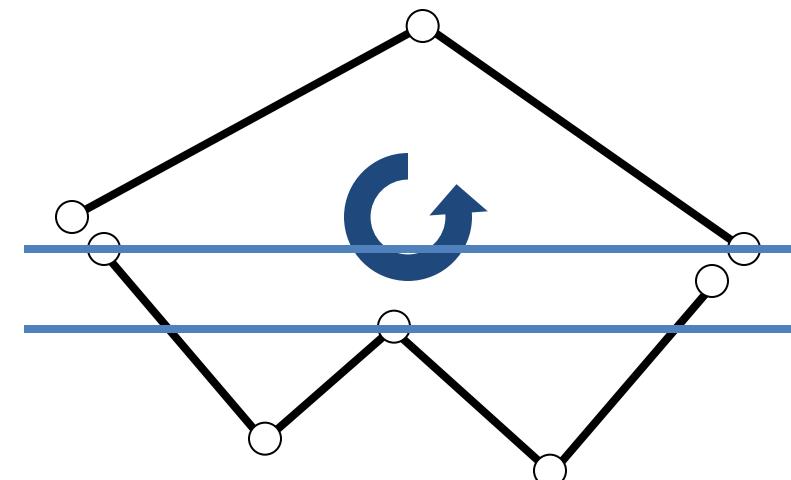
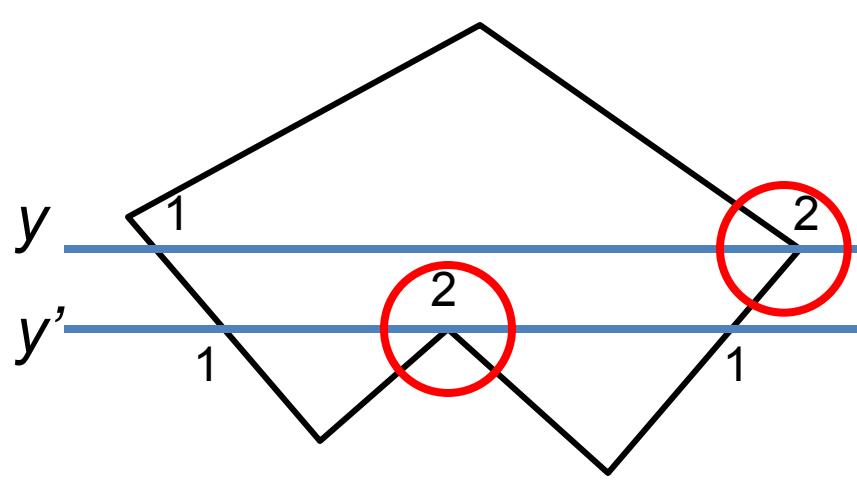
Boundary fill algorithm



Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

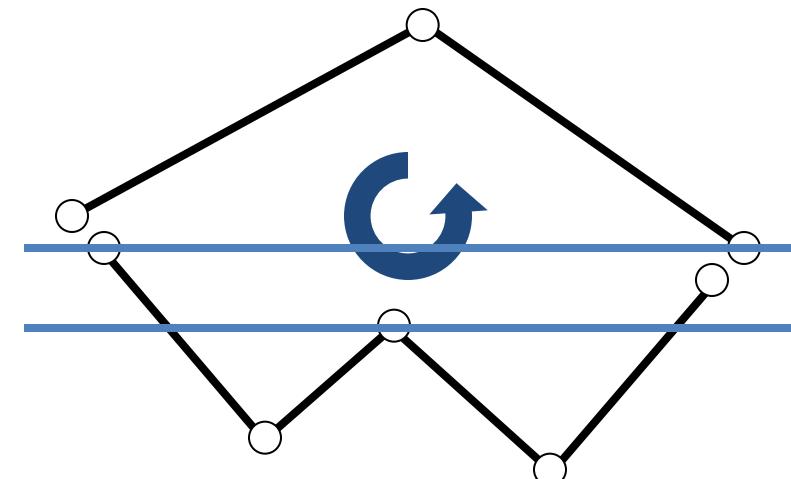
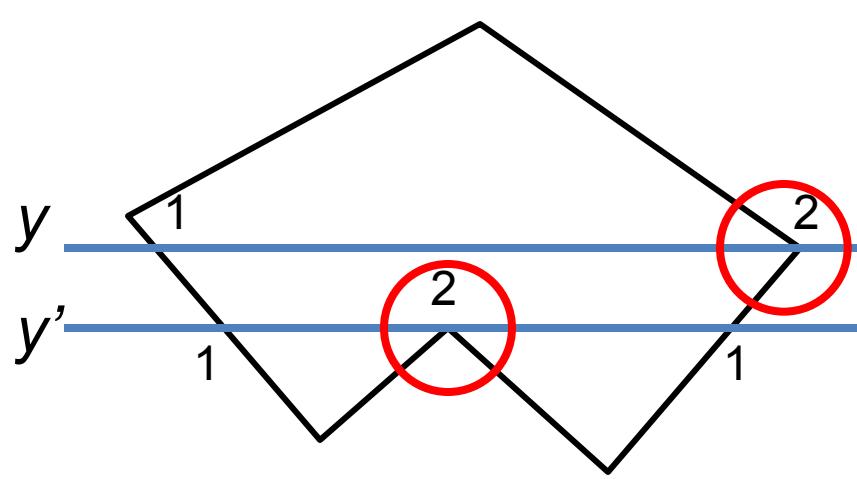
y' : intersection edges are same side



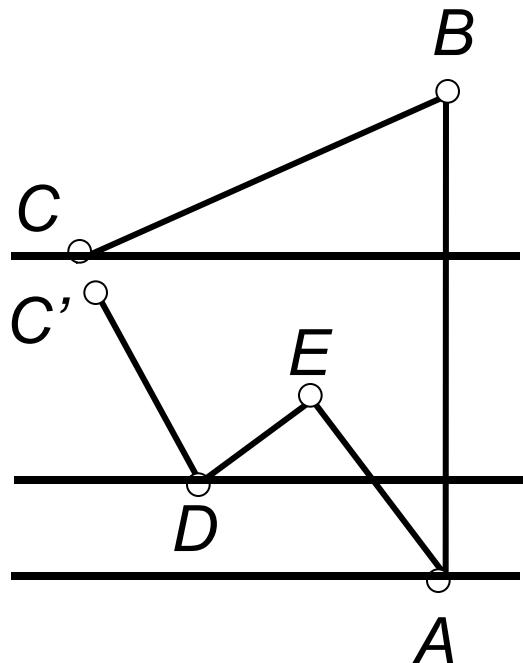
Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

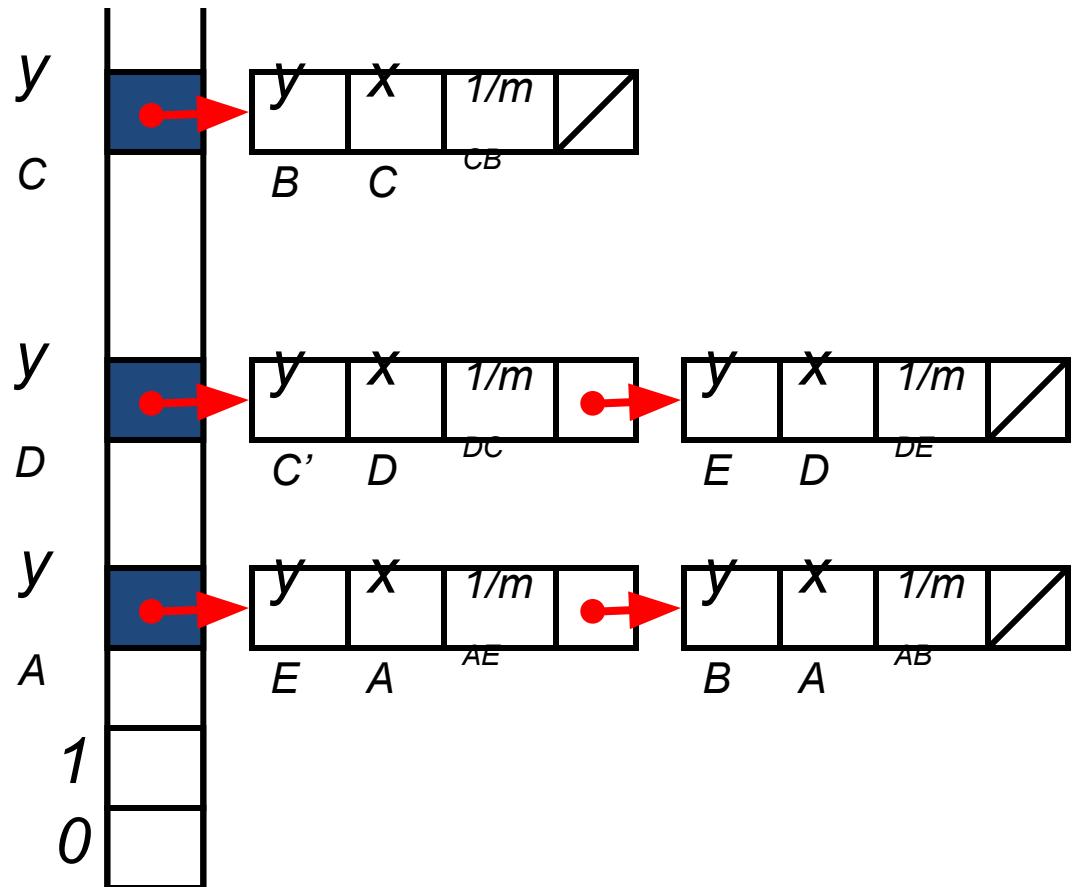
y' : intersection edges are same side



Edge Sorted Table



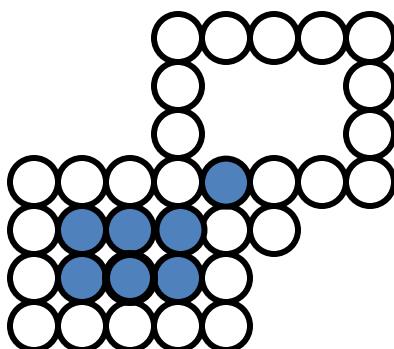
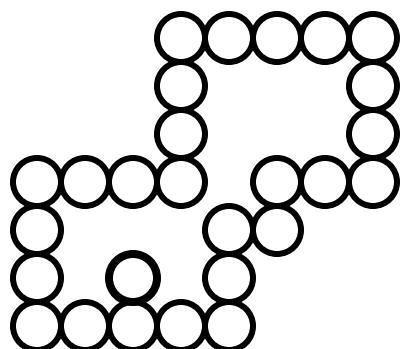
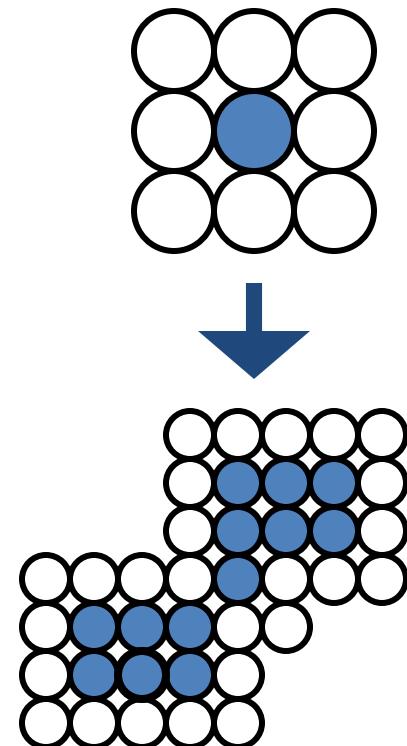
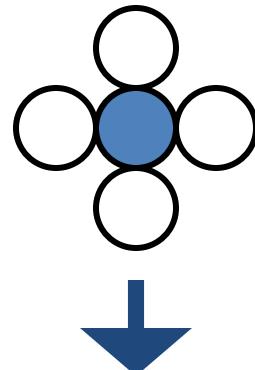
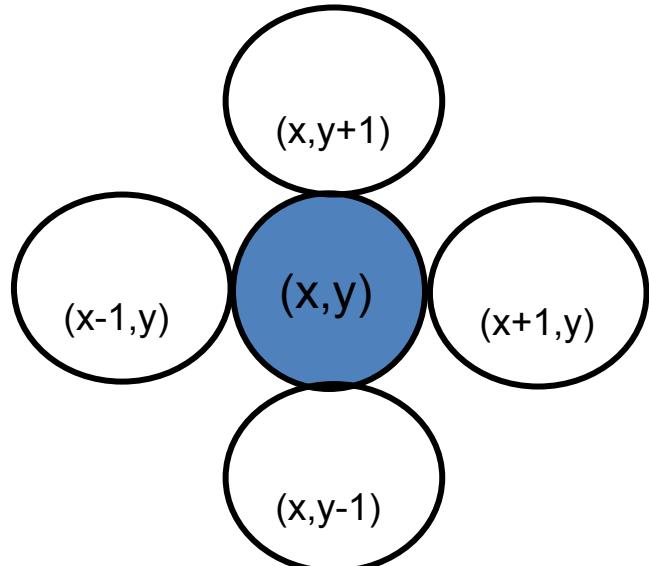
*Scan-Line
Number*



Proceed to Neighboring Pixels

4-Connected

8-Connected



Boundary Fill

Start at a point inside a region and paint the interior outward toward the boundary.

If boundary is specified in a single color the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered

Accepts as input coordinates of an interior point (x, y) a fill color and boundary color.

Starting from (x, y) the procedure tests neighboring position to determine whether they are of the boundary color.

If not they are painted with fill color and their neighbors are tested

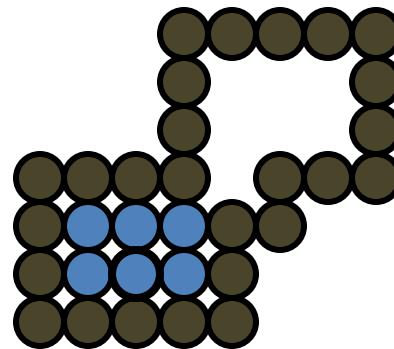
Process continues until all pixels up to boundary color for the area have been tested

boundaryFill ()

Retrive intensity of the pixel inside the boundary

If the retreived intensity is not that of the boundary pixels and AND it is not the fill color

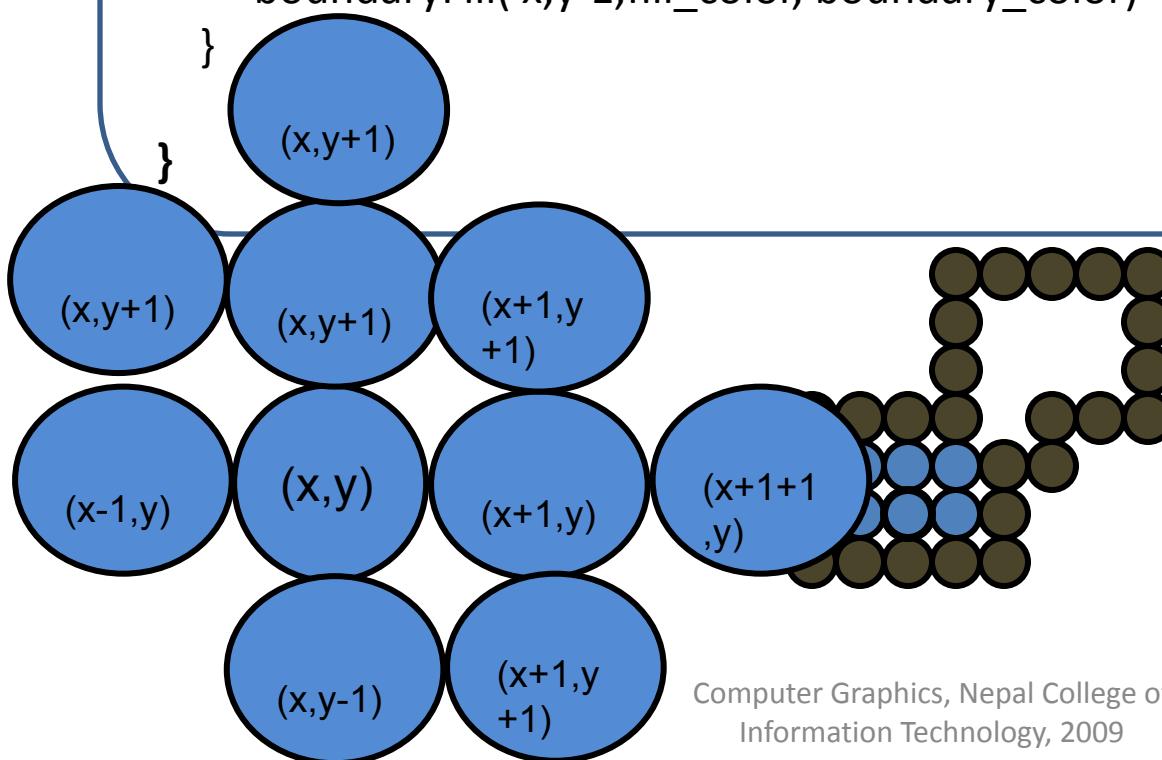
use recursion to plot pixel in all four directions



```

boundaryFill(int x,y,fill_color, boundary_color){
    int color;
    getpixel(x,y,color)
    if(color != boundary_color AND color != fill_color ){
        -> setpixel( x,y,fill_color)
        boundaryFill( x+1,y,fill_color, boundary_color)
        boundaryFill( x,y+1,fill_color, boundary_color)
        boundaryFill( x-1,y,fill_color, boundary_color)
        boundaryFill( x,y-1,fill_color, boundary_color)
    }
}

```



Flood Fill

Filling an area that is not defined within a single color boundary.

In this case replace a specified interior color instead of searching for boundary color value.

Start from specified interior point (x , y) and reassign all pixel values that are currently set to a given interior color with the desired color.

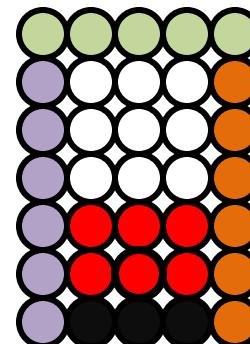
If the area we want to paint has more than 1 interior color , first assign pixel values so that all interior points have same color.

We can then use 8 or 4 connected approach to move on until all interior points have been repainted.

```

floodFill(int x,y,fill_color, original_color){
    int color;
    getpixel(x,y,color)
    if(color == original_color ){
        setpixel(x,y,fill_color)
        floodFill(x+1,y,fill_color, original_color)
        floodFill(x,y+1,fill_color, original_color)
        floodFill(x-1,y,fill_color, original_color)
        floodFill(x,y-1,fill_color, original_color)
    }
}
16777216 16777216 16777216 16777216 16777216
16777215

```



2D Transformations

A point is represented in 2 dimension by its coordinates

These two values are specified as elements of 1 row 2 column matrix

$P(x,y)$

In two dimension $[x, y]$ In three dimension $[x, y, z]$

Or alternatively a point is represented by a 2 row 1 column matrix

$y \quad y$
 z

$\begin{bmatrix} x \\ x \end{bmatrix}$

A.B not equal to B.A

rotate translate enlarge

composite Matrix = $\xrightarrow{\text{rotate x translate x enlarge}}$ (row wise)

composite Matrix = $\xleftarrow{\text{(enlarge x (translate x rotate))}}$ (column wise)

A B Rigid body transformations (no deformation in shape after transformation)

1 2 2 3 Non Rigid body transformations

3 4 3 4 B A
 2 3 1 2
 3 4 3 4

2D Transformations

Translation

Repositioning an object **along a straight line path** from one coordinate location to another

Add translational distance t_x, t_y to original coordinate position $P(x,y)$ to move the point to a new position $P'(x',y')$

$$t_y = 0$$

$$x' = x + t_x \quad y' = y + t_y$$

where the pair (t_x, t_y) is called the *translation vector*.

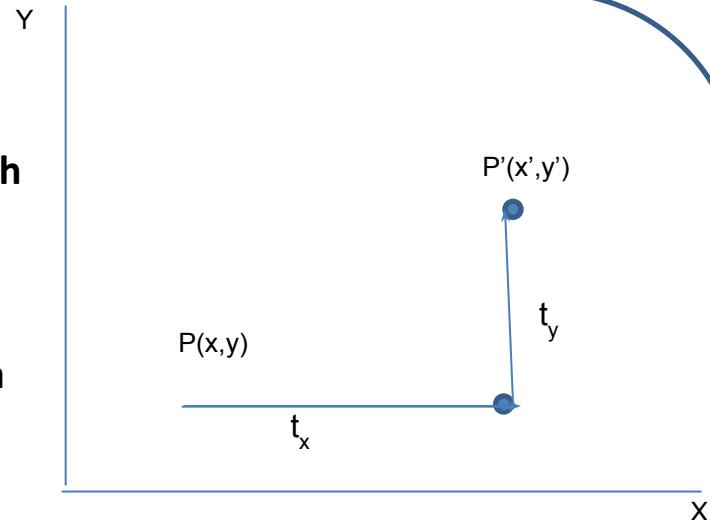
We can write equation as a single matrix equation by using column vectors to represent coordinate points and translation vectors. i.e.

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} P = \quad T = \begin{bmatrix} x \\ y \end{bmatrix} \quad \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

We can write, $P' = P + T$

or

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



2D Transformations

Translation

Repositioning an object **along a straight line path** from one coordinate location to another

Add translational distance t_x, t_y to original coordinate position (x, y) to move the point to a new position (x', y')

$$t_y = 0$$

$$x' = x + t_x \quad y' = y + t_y$$

where the pair (t_x, t_y) is called the *translation vector*.

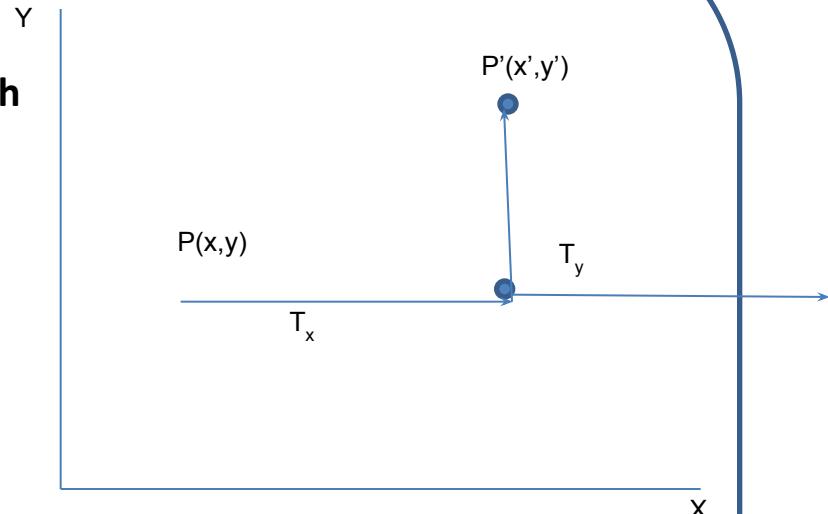
We can write equation as a single matrix equation by using column vectors to represent coordinate points and translation vectors. i.e.

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

We can write, $P_1 = P + T$

or

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$



2D Transformations

Scaling

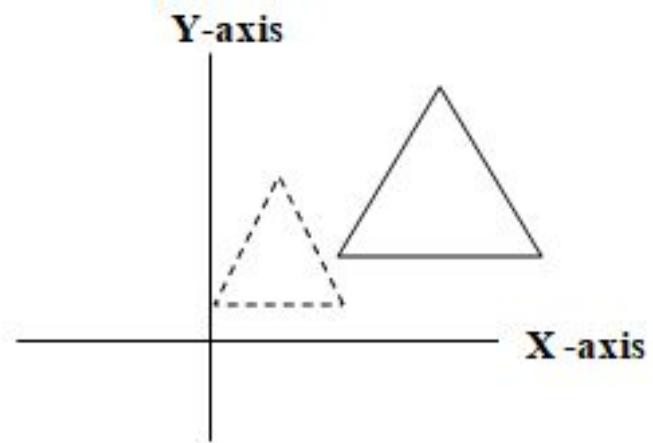
This transformation **changes the size** of an object

Such that we can **magnify** or **reduce** its size

In case of polygons scaling is done by **multiplying coordinate values (x,y) of each vertex by scaling factors s_x s_y** to produce the final transformed coordinates (x',y') .

s_x scales object in 'x' direction

s_y scales object in 'y' direction $x' = x * s_x$ $y' = y * s_y$



$$P' = \begin{pmatrix} P \\ x' \\ y' \end{pmatrix} \quad S = \begin{pmatrix} x & \\ y & \end{pmatrix} \quad \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} = 2$$

$$P' = P * S \text{ so } \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} * \begin{pmatrix} x \\ y \end{pmatrix}$$

Values greater than 1 for s_x s_y produce enlargement

Values less than 1 for s_x s_y ? size of object

$s_x = s_y = 1/2$ uniform scaling

$s_x = 5$ $s_y = 2$ non uniform scaling

2D Transformations

Scaling

This transformation **changes the size** of an object

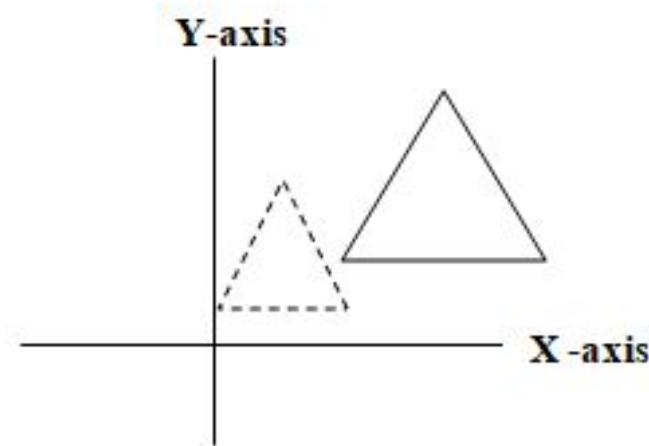
Such that we can **magnify** or **reduce** its size

In case of polygons scaling is done by **multiplying coordinate values (x,y) of each vertex by scaling factors s_x s_y** to produce the final transformed coordinates (x',y') .

s_x scales object in 'x' direction

s_y scales object in 'y' direction

$$x' = x * s_x$$



$$y' = y * s_y$$

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \end{bmatrix} \quad S = \begin{bmatrix} x \\ y \end{bmatrix} \quad \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

$$P' = P * S \text{ so } \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

Values greater than 1 for s_x s_y produce *enlargement*

Values greater than 1 for s_x s_y reduce size of object

$s_x = s_y = 3$ uniform scaling

$s_x = 4 \quad s_y = 3$ non uniform scaling

2D Transformations

Rotation (about origin, in ccw, by theta)

Repositioning an object **along a circular path** in x,y plane

Specify **rotation angle 'θ'** and position (x_r, y_r) of **rotation point** about which the object is to be rotated

+ value for 'θ' define **counter-clockwise** rotation about a point

- value for 'θ' define **clockwise** rotation about a point

If (x, y) is the original point 'r' the constant distance from origin, 'Φ' the original angular displacement from x-axis.

Now the point (x, y) is rotated through angle 'θ' in a counter clock wise direction

Express the transformed coordinates in terms of 'Φ' and 'θ' as

$$\cos(\Phi + \theta) = x'/r \text{ or } x' = r.\cos(\Phi + \theta) \quad \dots(i)$$

$$\sin(\Phi + \theta) = y'/r \text{ or } y' = r.\sin(\Phi + \theta) = \dots(ii)$$

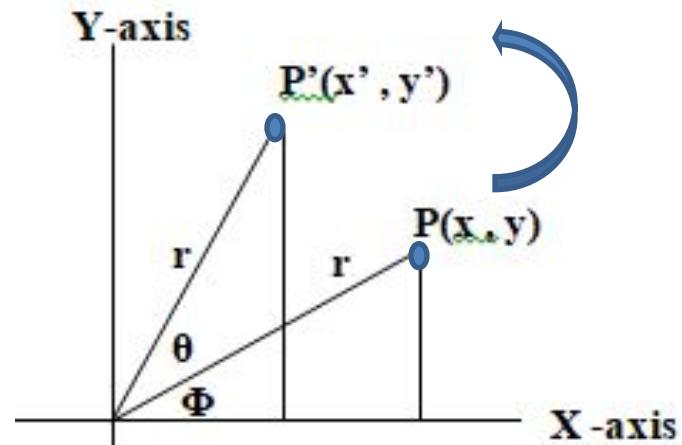
$$x' = r.\cos\Phi .\cos\theta - r.\sin\Phi .\sin\theta$$

$$y' = r.\cos\Phi .\sin\theta + r.\sin\Phi .\cos\theta$$

$$x = r.\cos\Phi \quad y = r.\sin\Phi$$

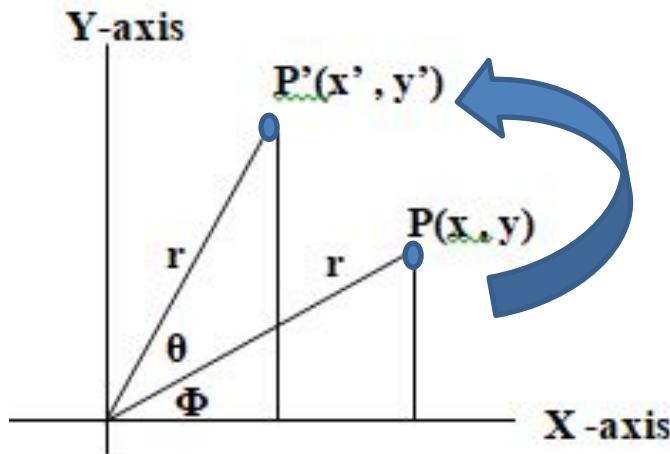
$$x' = x.\cos\theta - y.\sin\theta \quad \dots(i)$$

$$y' = x.\sin\theta + y.\cos\theta \quad \dots(ii)$$



2D Transformations

Rotation



Express the transformed coordinates in terms of 'Φ' and 'θ' as

$$x' = r \cos(\Phi + \theta) = r \cos\Phi \cdot \cos\theta - r \sin\Phi \cdot \sin\theta \quad \dots(i)$$

$$y' = r \sin(\Phi + \theta) = r \cos\Phi \cdot \sin\theta + r \sin\Phi \cdot \cos\theta \quad \dots(ii)$$

We know that original coordinates of point in polar coordinates are

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

substituting these values in (i) and (ii)

$$x' = x \cos\theta - y \sin\theta \quad y' = x \sin\theta + y \cos\theta$$

$$\mathbf{P}' = \mathbf{R} \cdot \mathbf{P}$$

using column vector representation for coordinate points the matrix form would be

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Transformations

Reflection

Generates a **mirror image** of the original object

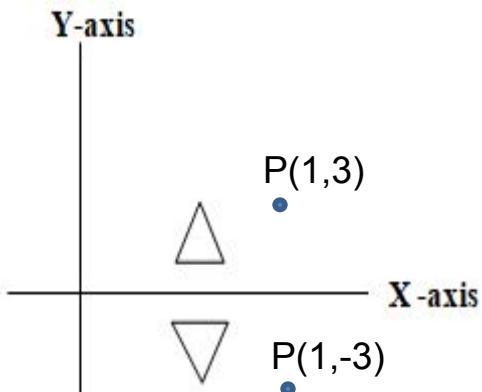
(i). Reflection about x axis or about line $y = 0$

Keeps x value same but flips y value of coordinate points

$$x' = x \text{ and } y' = -y \quad \text{or} \quad P' = R_{fx} * P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$x' = 1 \quad y' = -3 \quad x = 1 \quad y = 3$$



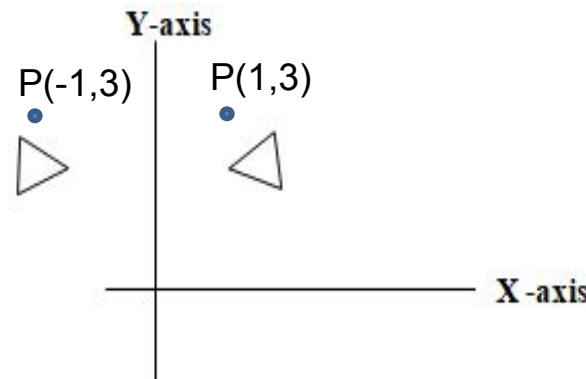
(ii). Reflection about y axis or about line $x = 0$

Keeps y value same but flips x value of coordinate poi

$$y' = y \text{ and } x' = -x \quad \text{or} \quad P' = R_{fy} * P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

$$x' = -1 \quad y' = 3$$



2D Transformations

Reflection

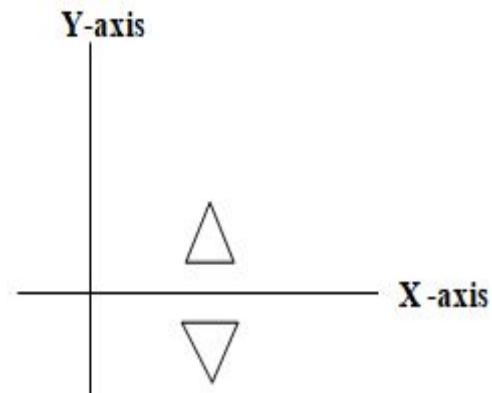
Generates a **mirror image** of the original object

(i). Reflection about x axis or about line $y = 0$

Keeps 'x' value same but flips y value of coordinate points

$$x' = x \text{ and } y' = -y \quad \text{or} \quad P' = R_f * P$$

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} & * \\ & \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

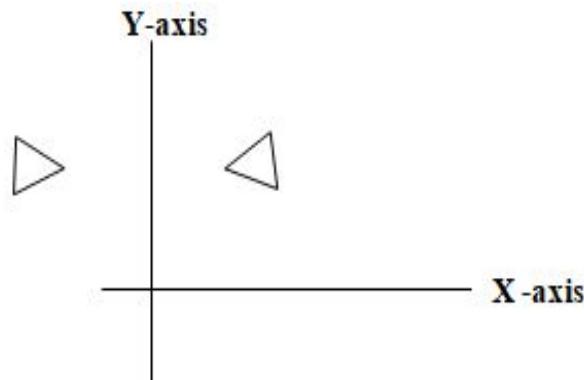


(ii). Reflection about y axis or about line $x = 0$

Keeps 'y' value same but flips x value of coordinate points

$$y' = y \text{ and } x' = -x \quad \text{or} \quad P' = R_f * P$$

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} & \\ & * \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$



2D Transformations

Reflection

Generates a **mirror image** of the original object

(iii). Reflection about origin

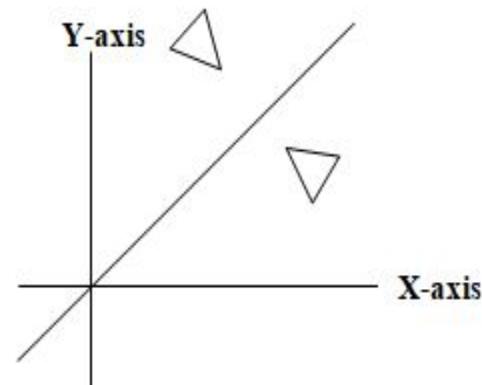
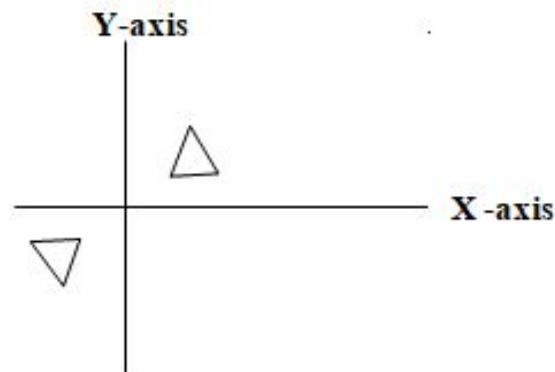
Flip both 'x' and 'y' coordinates of a point

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

(iv). Reflection about line $y = x$

Steps required:

- i. Rotate about origin in clockwise direction by 45 degree which rotates line $y = x$ to x-axis
- ii. Take reflection against x-axis
- iii. Rotate in anti-clockwise direction by same angle



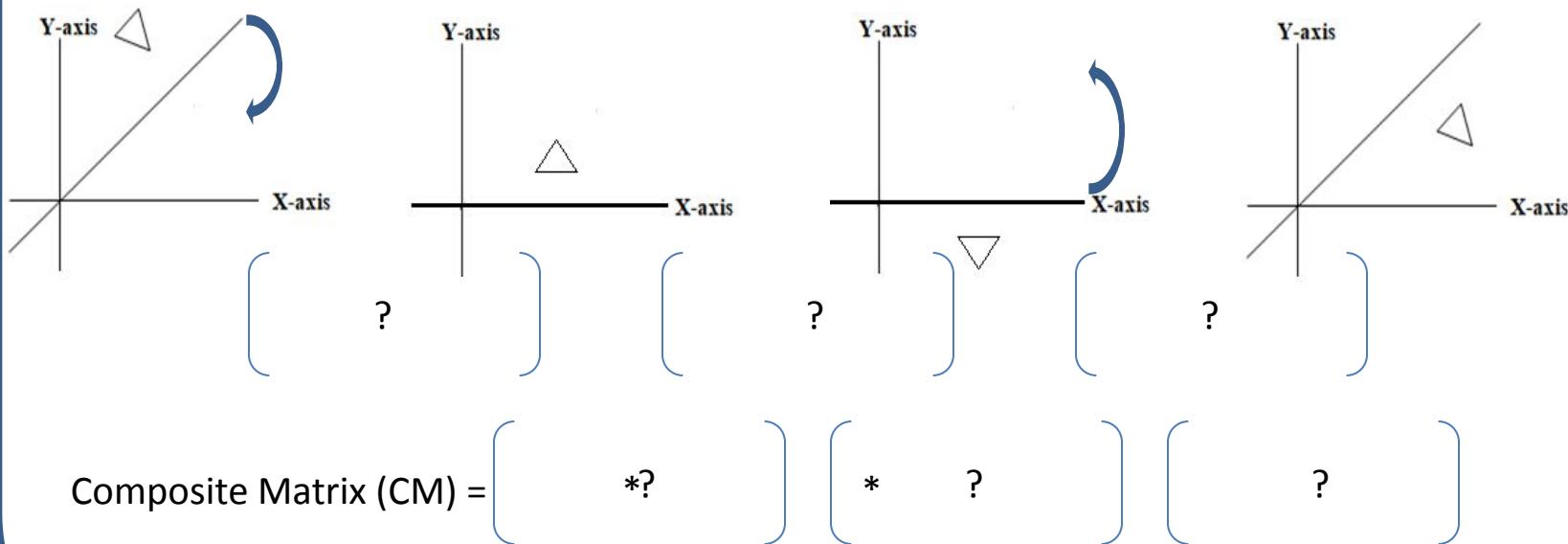
2D Transformations

Reflection

iv). Reflection about line $y = x$

Steps required:

- i. Rotate about origin in clockwise direction by 45 degree which rotates line $y = x$ to x-axis
- ii. Take reflection against x-axis
- iii. Rotate in anti-clockwise direction by same angle

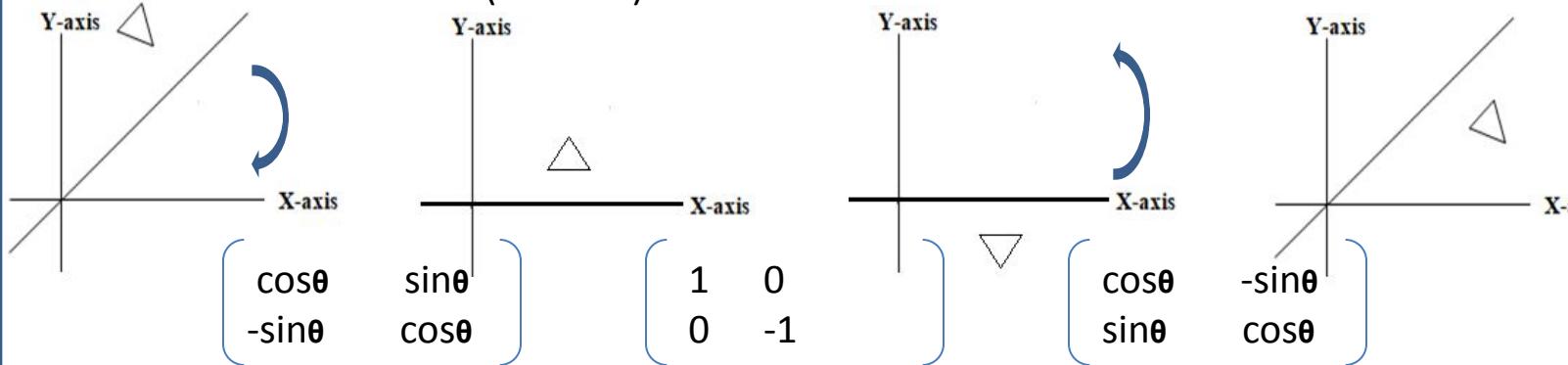


2D Transformations

Reflection

iv). Reflect a point P(1,3) about line $y = -x$

$$cm = R_{ccw} \cdot (R_f \cdot R_{cw})$$



$$\text{Composite Matrix (CM)} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} * \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$\begin{bmatrix} P' = \\ x' \\ y' \end{bmatrix} = CM \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} P \\ x \\ y \end{bmatrix}$$

$$x' = 3 \quad y' = 1$$

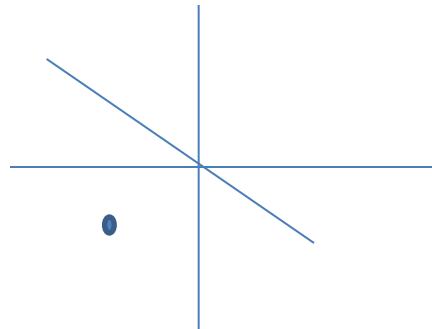
$$x = 1 \quad y = 3$$

2D Transformations

Reflection

iv). Reflect a point P(1,3) about line $y = -x$

$$cm = R_{ccw} \cdot (R_f \cdot R_{cw})$$



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

$$\text{Composite Matrix (CM)} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$P' = CM \cdot P$$
$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix}$$

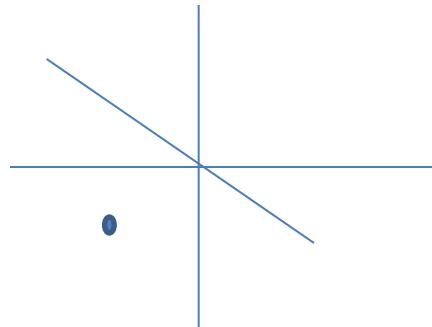
$$\begin{array}{ll} x = 1 & y = 3 \\ x' = 3 & y' = 1 \end{array}$$

2D Transformations

Reflection

iv). Reflect a point P(1,3) about line $y = -3x$

$$cm = R_{ccw} \cdot (R_f \cdot R_{cw})$$



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

$$\text{Composite Matrix (CM)} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} * \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$P' = CM \cdot P$$
$$\begin{pmatrix} X' \\ Y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix}$$

$$\begin{array}{ll} x = 1 & y = 3 \\ x' = 3 & y' = 1 \end{array}$$

2D Transformations

Shearing (non rigid body transformation)

Distorts the shape of object in either 'x' or 'y' or both direction

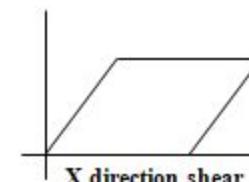
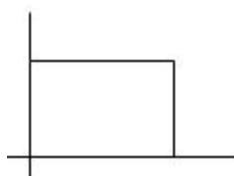
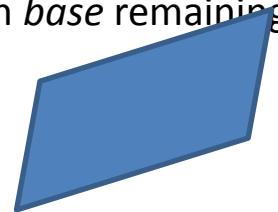
In case of single directional shearing (e.g. in 'x' direction can be viewed as an object made up of very thin layer and slid over each other with *base* remaining where it is)

in 'x' direction,

$$x' = x + s_{hx} \cdot y$$

$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s_{hx} \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

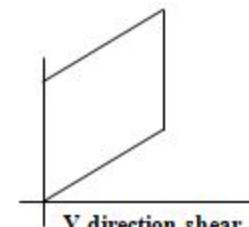
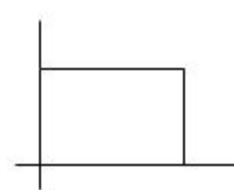


in 'y' direction,

$$x' = x$$

$$y' = y + s_{hy} \cdot x$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ s_{hy} & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

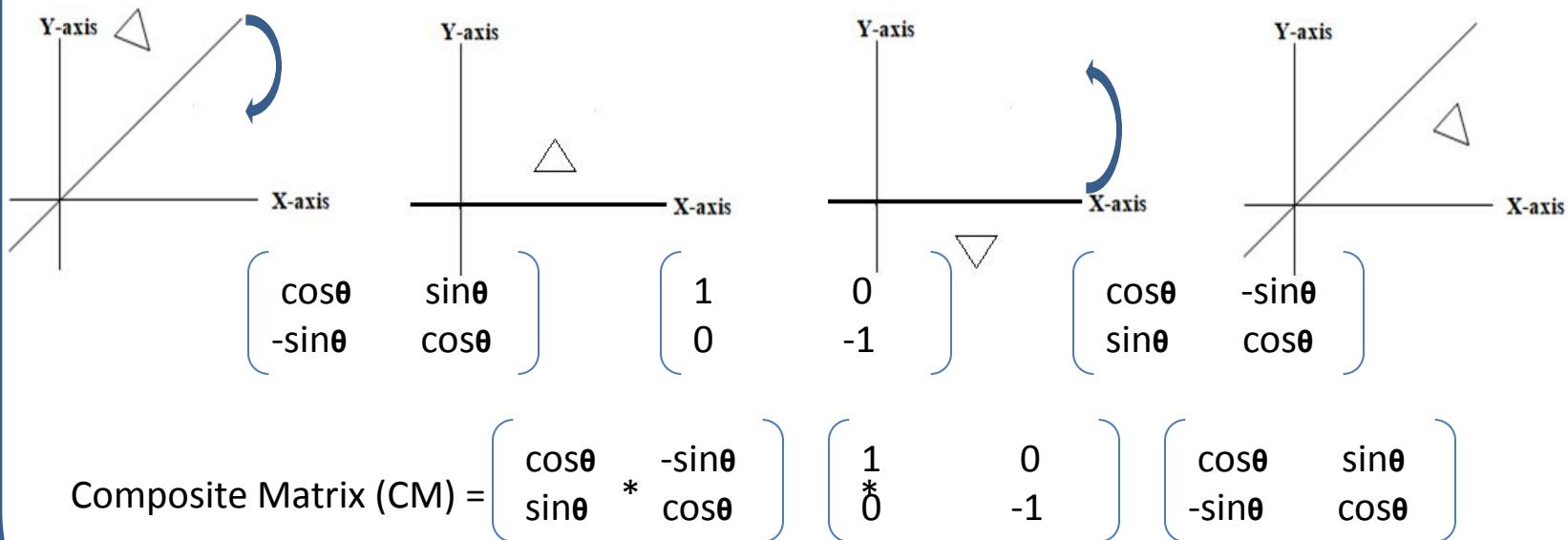


2D Transformations

Composite Transformation

With the matrix representation of transformation equations it is possible to **setup a matrix for any sequence of transformations** as a composite transformation matrix by calculating the matrix product of individual transformation.

For column matrix representation of coordinate positions we form composite transformation by multiplying matrices in order from right to left.



2D Transformations

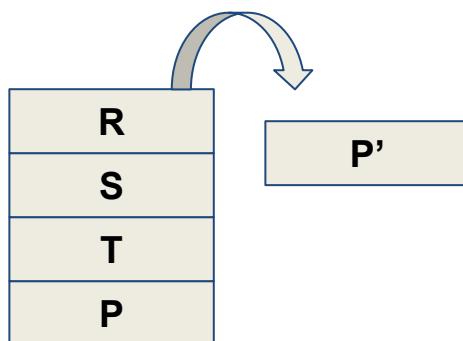
Homogenous coordinate

$$P' = T + P \quad \text{additive}$$

$$P' = R * P \quad \text{multiplicative}$$

$$P' = S * P \quad \text{multiplicative}$$

$$\text{Composite Matrix (CM)} = \begin{pmatrix} \cos\theta & * & -\sin\theta \\ \sin\theta & & \cos\theta \end{pmatrix} \begin{pmatrix} 2 & & 0 \\ 0 & & 2 \end{pmatrix} \begin{pmatrix} T_x \\ T_y \end{pmatrix}$$



2D Transformations

Homogenous coordinate

$$P' = T + P \quad \text{additive}$$

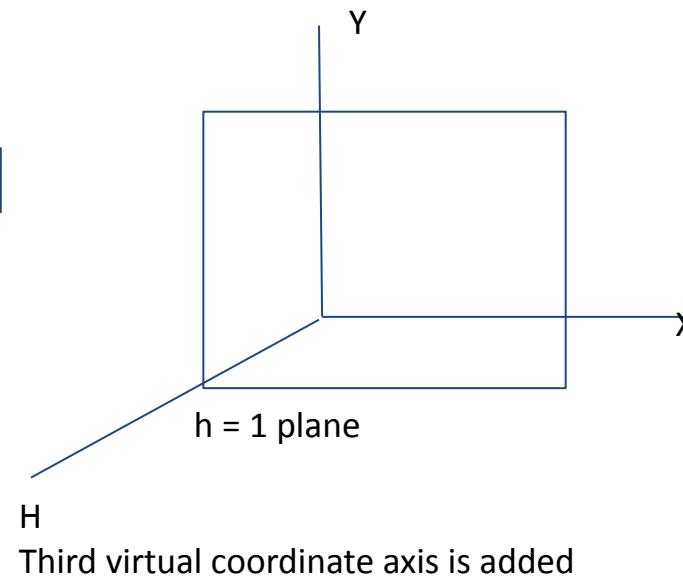
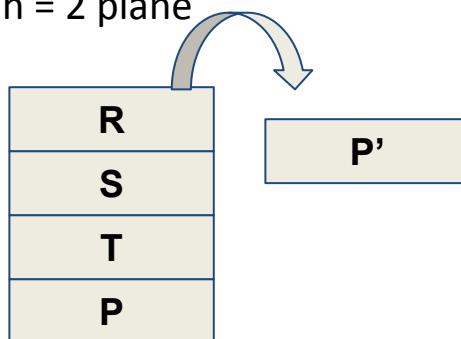
$$P' = R * P \quad \text{multiplicative}$$

$$P' = S * P \quad \text{multiplicative}$$

$$P(2,3) = P(2,3,1) \quad h = 1 \text{ plane}$$

$$\begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix}$$

$$P(2,3) = P(4,6,2) \quad h = 2 \text{ plane}$$



2D Transformations

Homogenous coordinate

$$P' = T + P \quad \text{additive}$$

$$P' = R * P \quad \text{multiplicative}$$

$$P' = S * P \quad \text{multiplicative}$$

$$P(2,3) = P(2,3,1) \quad h = 1 \text{ plane}$$

$$P(2,3) = P(2,3,1) \quad h = 2 \text{ plane}$$

$$cm = S . R . T$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} T_x \\ T_y \\ 1 \end{bmatrix} + \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = T + P$$

$$x' = x + T_x$$

$$y' = y + T_y$$

$$1 = 1$$

$$= \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix}$$

2D Transformations

Homogenous coordinate

$P' = T + P$ additive

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

$P' = R * P$ multiplicative

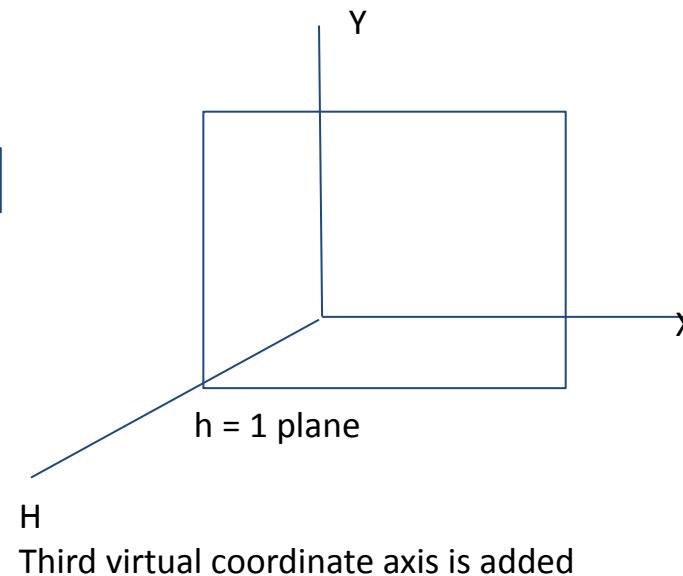
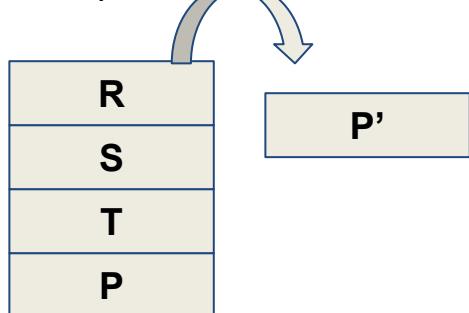
$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$P' = S * P$ multiplicative

$$\begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix}$$

$P(2,3) = P(2,3,1)$ $h = 1$ plane

$P(2,3) = P(2,3,1)$ $h = 2$ plane



2D Transformations

Homogenous coordinate

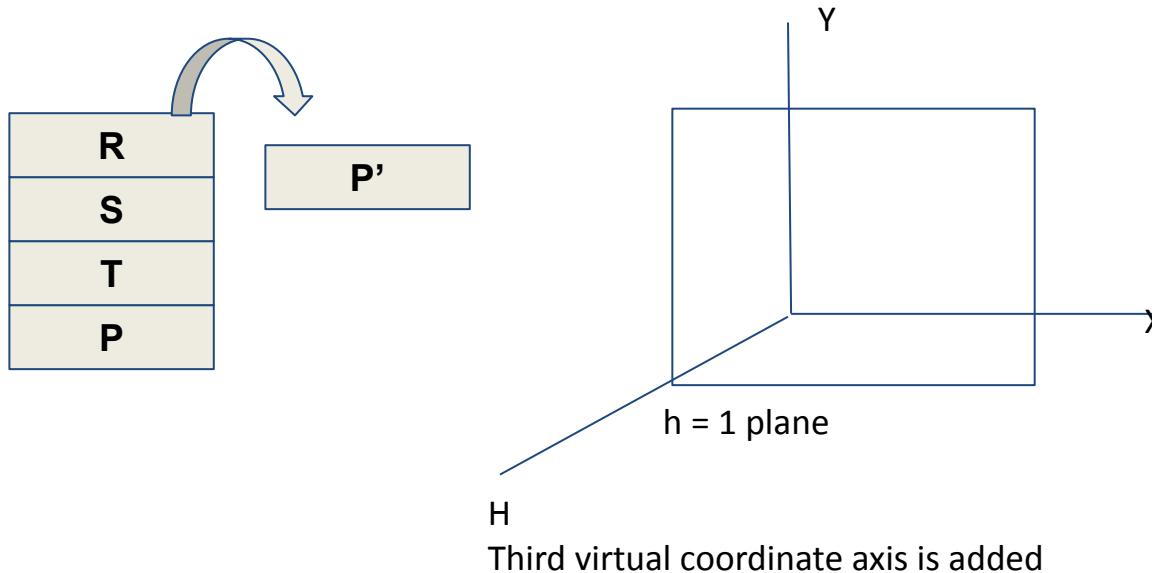
$$P' = T + P \quad \text{additive}$$

$$P' = R * P \quad \text{multiplicative}$$

$$P' = S * P \quad \text{multiplicative}$$

Composite Matrix (CM) =

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix}$$



Fixed Point Rotations

A point P(3,2) is required to be **shifted down by 2 units**, rotated about origin by 45 degrees in anti clock wise direction and scaled by 2 times, what will be the final coordinate of the point after performing these operations?

$$\text{Composite matrix} = S(sx=2, sy=2) \cdot R(\theta = 45 \text{ ccw}) \cdot T(tx = 0, ty = -2)$$

$$\text{Composite matrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} ? \\ ? \\ 1 \end{bmatrix} \quad \text{CM}$$

$$P' = \begin{bmatrix} ? \\ ? \\ 1 \end{bmatrix}$$

Fixed Point Rotations

A line with end points A(-5,-1) B(-3,-4) need to be reflected about line $y = -2$, what will be the final coordinates?

Steps:

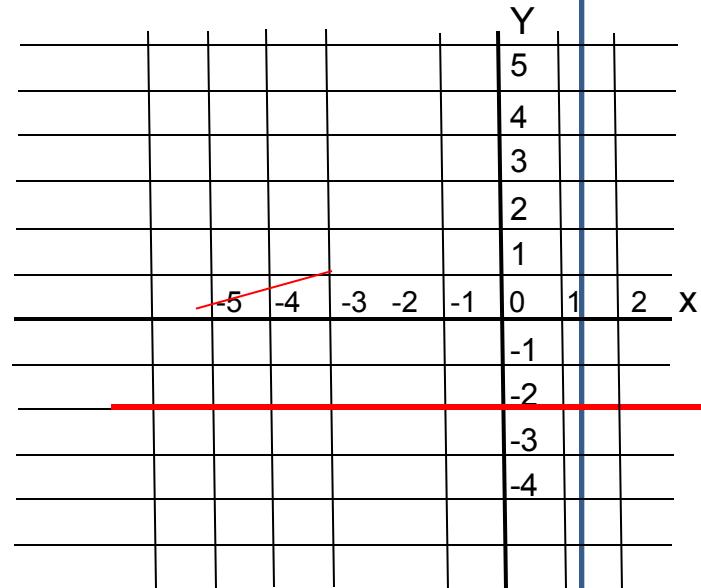
1. Line $y = -2$ is translated in y direction along with the line to be reflected so that it is aligned with x axis
2. Take reflection of the line about x axis
3. Line is translated back to its original location along with the reflected line

$$CM = T(tx = 0, ty = -2) \cdot R_{fx} \cdot T(tx = 0, ty = 2)$$

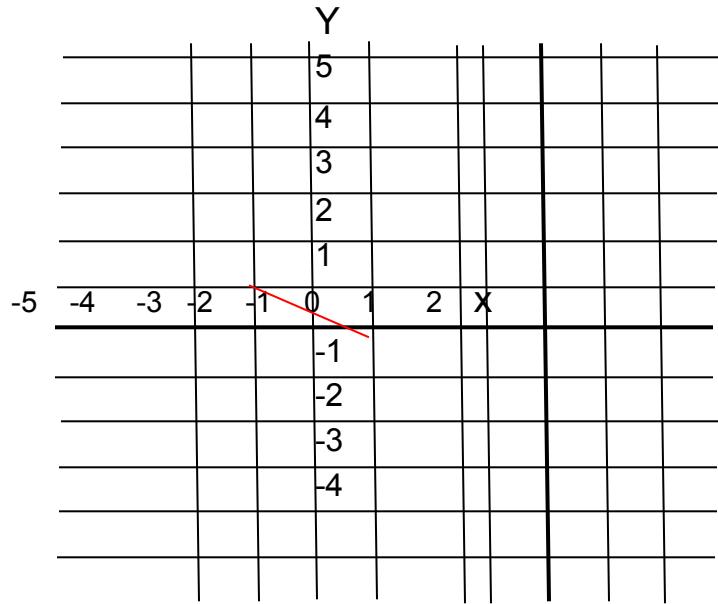
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P' = CM \cdot P$$

$$P' = \begin{pmatrix} A & B \\ A' & B' \\ \end{pmatrix} \cdot \begin{pmatrix} 1 & -1 \\ -1 & -4 \\ 1 & 1 \\ \end{pmatrix}$$



Fixed Point Rotations



$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Fixed Point Rotations

$$\begin{bmatrix} 1 & 0 & xf \\ 0 & 1 & yf \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -xf \\ 0 & 1 & -yf \\ 0 & 0 & 1 \end{bmatrix}$$

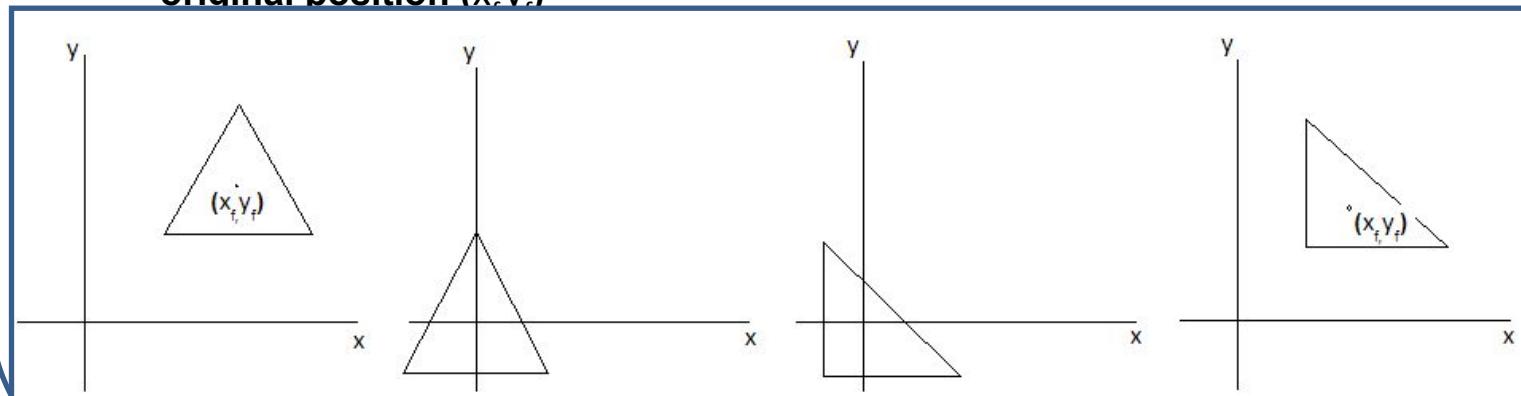
$$CM = T(x_f, y_f) R T(-x_f, -y_f)$$

$$P' = CM \times P$$

An object is required to be rotated about a fixed point (x_f, y_f)

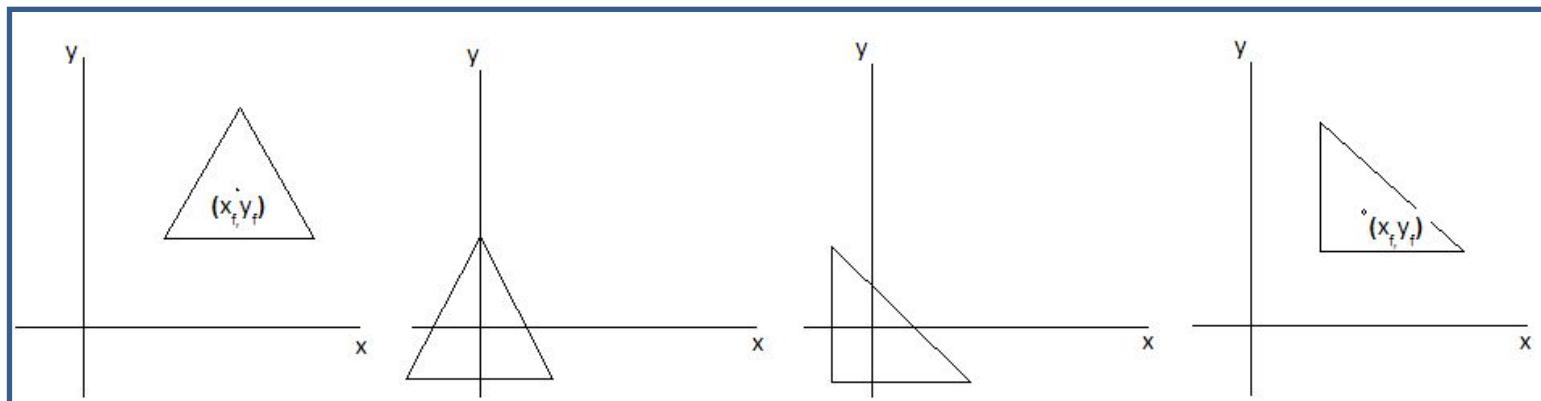
Steps:

1. The fixed point (x_f, y_f) along with the object is translated to origin
2. The object is rotated about origin
3. The fixed point along with the rotated object is translated back to its original position (x_f, y_f)



Fixed Point Rotations

An object is required to be rotated about a fixed point (x_f, y_f)



$$CM = T(x_f, y_f)$$

$$R$$

$$T(-x_f, -y_f)$$

$$\text{Composite Matrix (CM)} = P' = \left\{ \begin{array}{c} \left(\begin{array}{ccc} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{ccc} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{array} \right) \left(\begin{array}{ccc} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{array} \right) \\ CM \\ \left(\begin{array}{ccc} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{array} \right) \end{array} \right\}$$

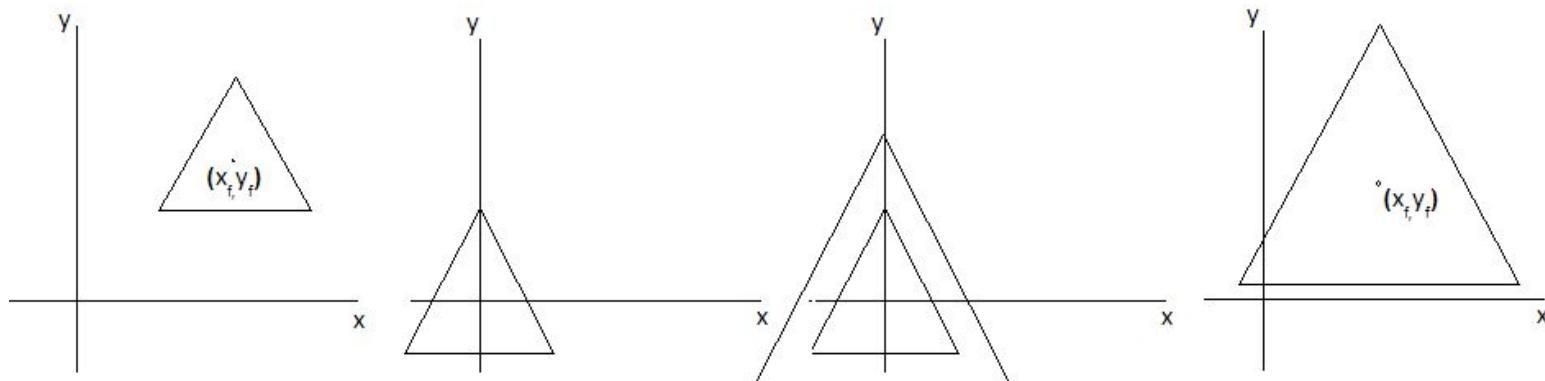
Fixed Point Scaling

An object is required to be scaled about a fixed point (x_f, y_f)

$$\begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix}$$

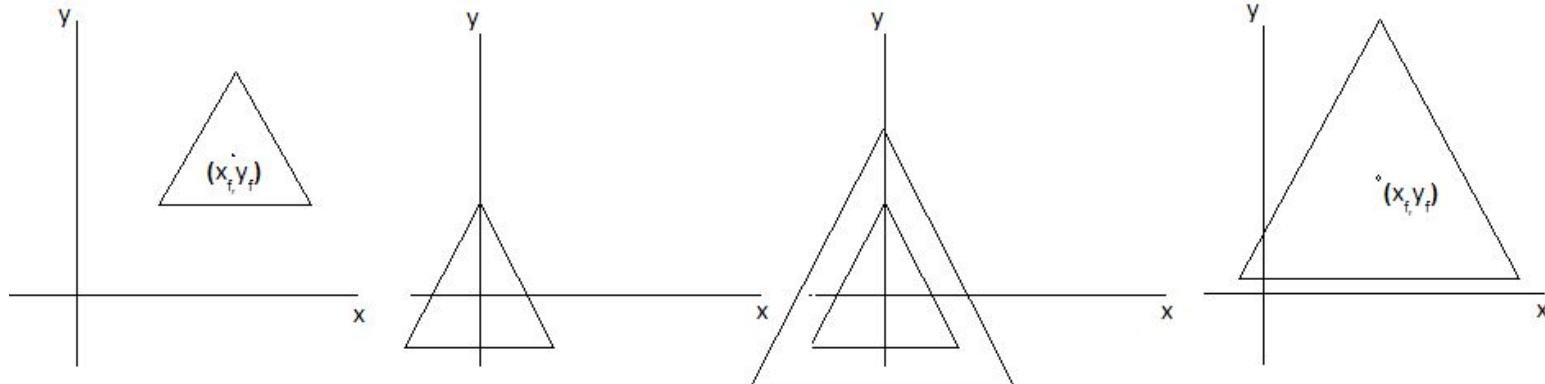
Steps:

1. The fixed point (x_f, y_f) along with the the object is translated to origin
2. The object is scaled about origin
3. The fixed point along with the rotated object is translated back to its original position (x_f, y_f)



Fixed Point Rotations

An object is required to be scaled about a fixed point (x_f, y_f)



$$CM = (T(tx, yf). (S(sx, sy). T(-xf, -yf)))$$

$$\text{Composite Matrix (CM)} = \begin{pmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{pmatrix}$$
$$P' = \left[\begin{array}{c|c} \text{CM} & \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \end{array} \right]$$

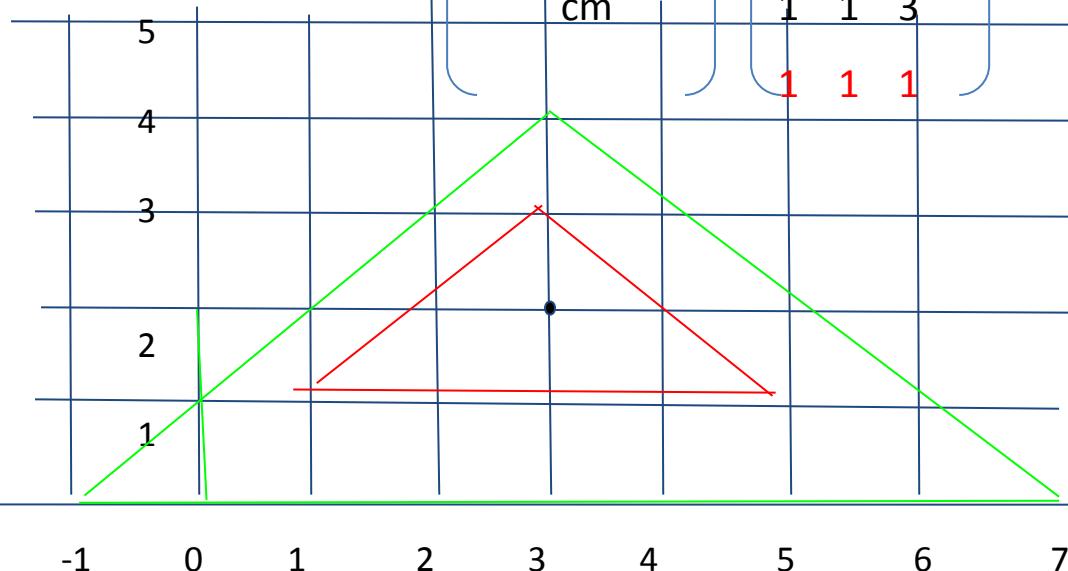
Exercise

Enlarge a triangle A(1,1) B(5,1) C(3,3) about any fixed point P(3,2) by twice its original size $s_x = 2$ $s_y = 2$

$$CM = T(tx = 3, ty = 2) \quad S(sx = 2, sy = 2) \quad T(tx = -3, ty = -2)$$

$$CM = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P' = \begin{pmatrix} 1 & 5 & 3 \\ 1 & 1 & 3 \end{pmatrix} =$$



Rotate a triangle A(1,1) B(5,1) C(3,3) about any fixed point P(3,2) by 90 degrees in anti clockwise direction

Steps:

1. Translate the fixed point P(3,2) to origin along with vertices of triangle to be rotated
2. Rotate the triangle by 90 in ccw
3. Translate the fixed point back to its original location along with rotated vertices of triangle

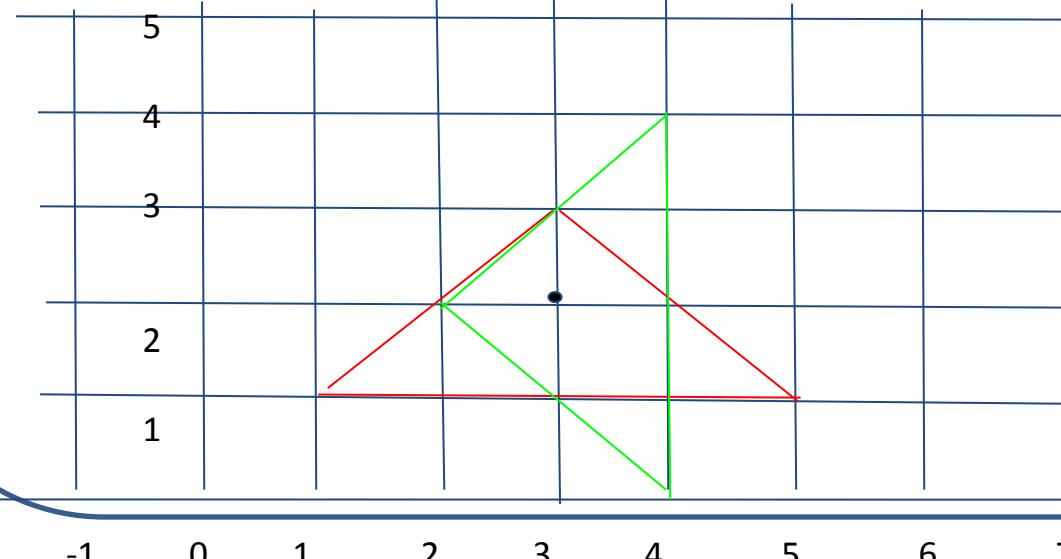
$$CM = T(tx = 3, ty = 2)$$

$$R \quad T(tx = -3, ty = -2)$$

$$CM = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} \cos 90 & -\sin 90 & 0 \\ \sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P' = cm$$

The final vertices of the triangle after performing the desired transformation are A'(4,0), B'(4,4), C'(2,2)

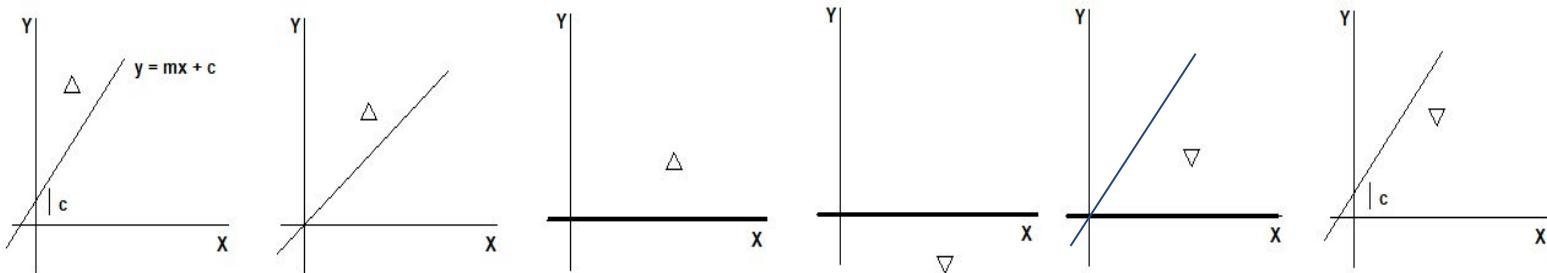


2D Rotations about line $y = mx + c$

An object is required to be reflected about a line $y = mx + c$

Steps:

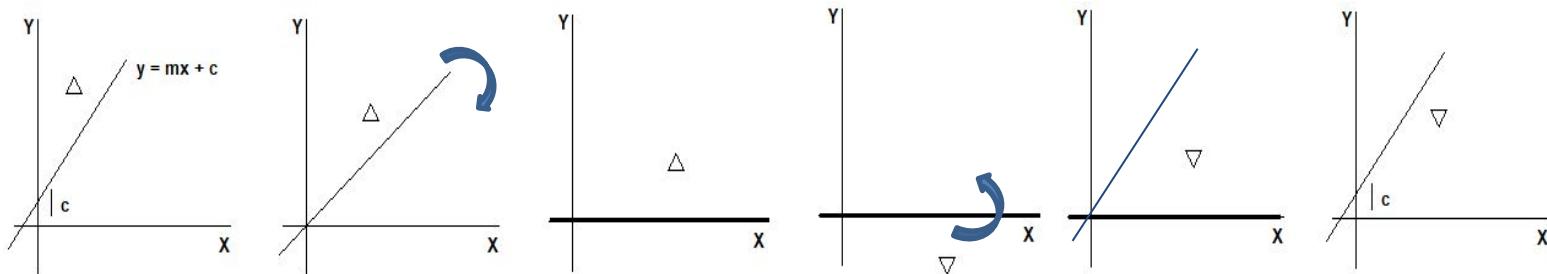
1. The line is translated so that it passes thru origin
2. The line is rotated along with the object to be reflected so that the line is aligned with one of the coordinate axes
3. The object is reflected about that major coordinate axis
4. The line is rotated back along with the reflected object by the same angle with which it was rotated earlier to align the line with the coordinate axis
5. The line is translated back along with the reflected object to its original position



Fixed Point Rotations

An object is required to be reflected about a line $y = mx + c$

$$CM = T(0,c) \cdot R_{ccw} \cdot R_{fx} \cdot R_{cw} \cdot T(0,-c)$$



$$CM = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -c \\ 0 & 0 & 1 \end{pmatrix}$$
$$P' = CM \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Fixed Point Rotations

A Triangle with vertices A(0,4) B(0,5) C(1,5) is required to be reflected about line

$$y = 2x + 1 \quad CM = T(tx=0, ty=1) \cdot R_{ccw}(\theta) \cdot R_f(\theta) \cdot R_{cw} \cdot T(tx=0, ty=-1)$$

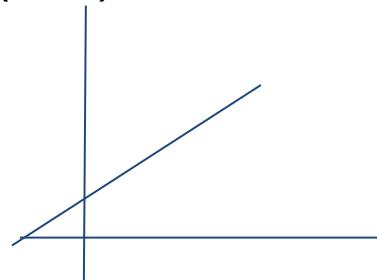
$$y \text{ intercept} = 1 \quad \tan(\theta) = 2 \quad \theta = \tan^{-1}(2) = \text{xxxxx} \quad \cos(\theta)$$

x y

0 1

1 3

2 5



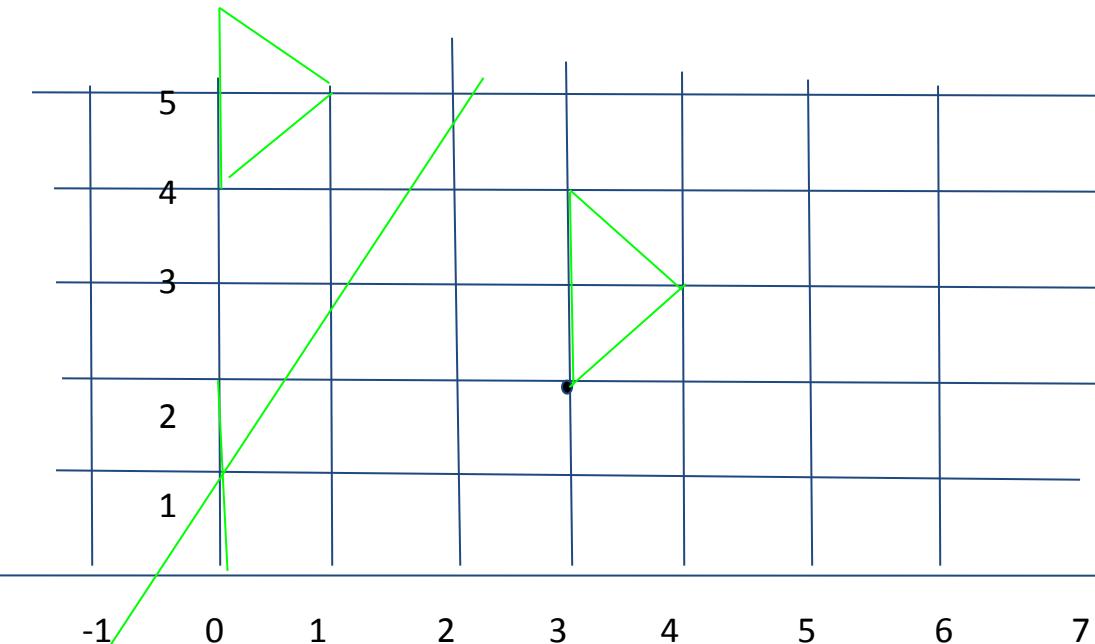
$$CM = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$P' = CM \cdot \begin{pmatrix} A & B & C \\ A' & B' & C' \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 4 & 5 & 5 \\ 1 & 1 & 1 \end{pmatrix}$$

Exercise

A Triangle with vertices A(0,4) B(0,5) C(1,5) about line $y = 2x + 1$

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

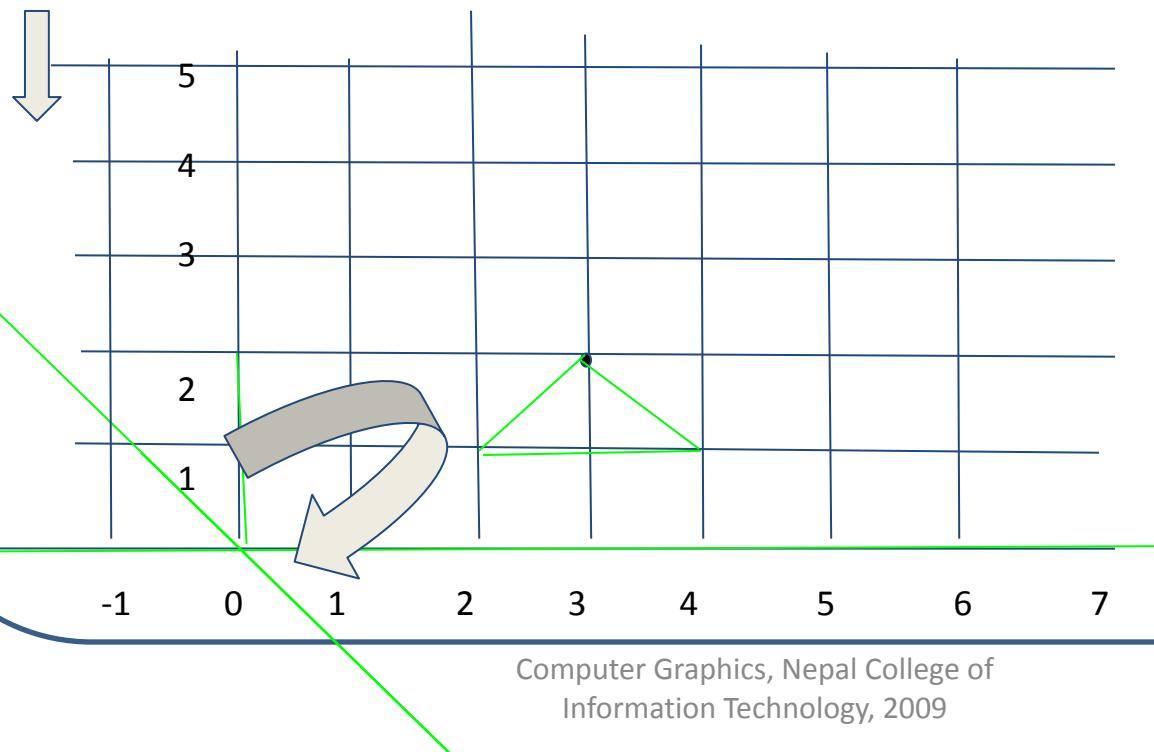


$$\begin{pmatrix} 3 & 3 & 4 \\ 2 & 4 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

Exercise

A Triangle with vertices A(2,3) B(3,4) C(4,3) about line $y = -x + 2$

$$CM = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$



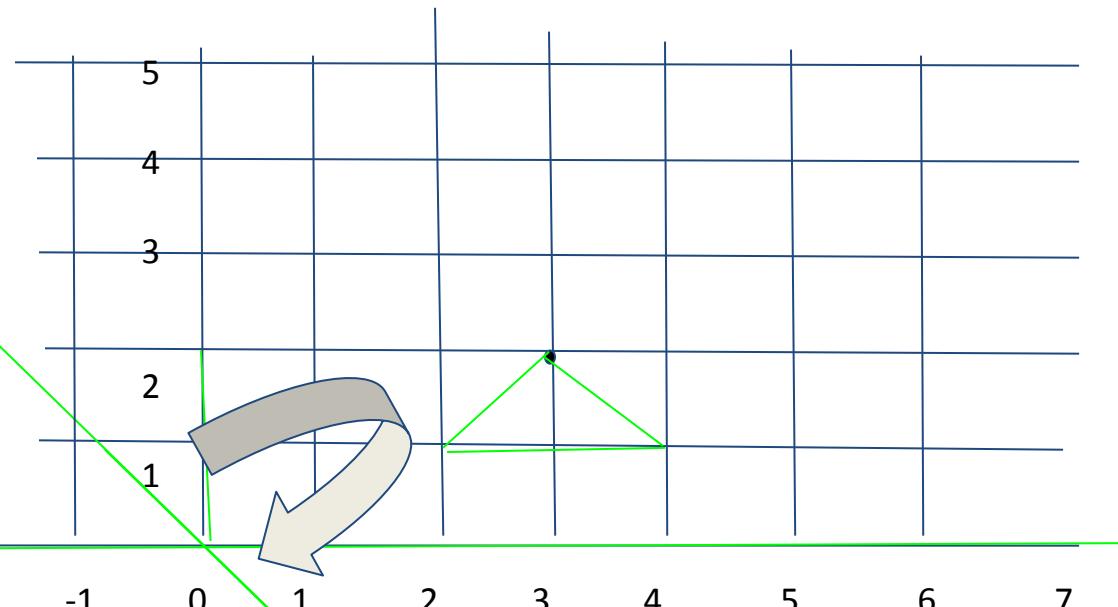
Exercise

A Triangle with vertices A(2,3) B(3,4) C(4,3) is required to be reflected about line $y = -x$

$$\Theta = \tan^{-1}(-3)$$

$$CM =$$

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



Composite Transformation

Matrix Composition: A single matrix formed by combining a number of **transformations** or sequence of **transformations**

The resulting matrix is called as **composite** matrix.

With the matrix representation of transformation equations it is possible to setup a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of individual transformation.

For column matrix representation of coordinate positions we form composite transformation by multiplying matrices in order from right to left.

i. Two Successive Translation are Additive

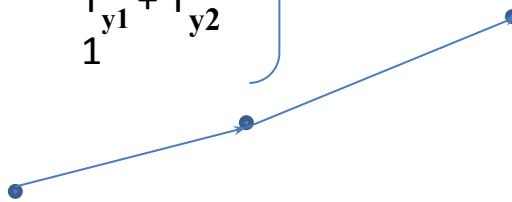
Let two successive translation vectors (t_{x_1}, t_{y_1}) and (t_{x_2}, t_{y_2}) are applied to a coordinate position P then

$$\text{or, } P' = T(t_{x_2}, t_{y_2}) \cdot \{T(t_{x_1}, t_{y_1}) \cdot P\} \quad \text{and} \quad P' = \{T(t_{x_2}, t_{y_2}) \cdot T(t_{x_1}, t_{y_1})\} \cdot P$$

Here the composite transformation matrix for this sequence of translation is

$$\begin{pmatrix} 1 & 0 & T_{x_2} \\ 0 & 1 & T_{y_2} \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & T_{x_1} \\ 0 & 1 & T_{y_1} \\ 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & T_{x_1} + T_{x_2} \\ 0 & 1 & T_{y_1} + T_{y_2} \\ 0 & 0 & 1 \end{pmatrix}$$



Composite Transformation

ii. Two successive Scaling operations are Multiplicative

Let (s_{x_1}, s_{y_1}) and (s_{x_2}, s_{y_2}) be two successive vectors applied to a coordinate position P then the composite scaling matrix thus produced is

$$\text{or, } P' = S(s_{x_2}, s_{y_2}) \cdot \{S(s_{x_1}, s_{y_1}) \cdot P\} \quad \text{and} \quad P' = \{S(s_{x_2}, s_{y_2}) \cdot S(s_{x_1}, s_{y_1})\} \cdot P$$

Here the composite transformation matrix for this sequence of translation is

$$\begin{pmatrix} s_{x_2} & 0 & 0 \\ 0 & s_{y_2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} s_{x_1} & 0 & 0 \\ 0 & s_{y_1} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{x_2} \cdot s_{x_1} & 0 & 0 \\ 0 & s_{y_2} \cdot s_{y_1} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

iii. Two successive Rotation operations are Additive

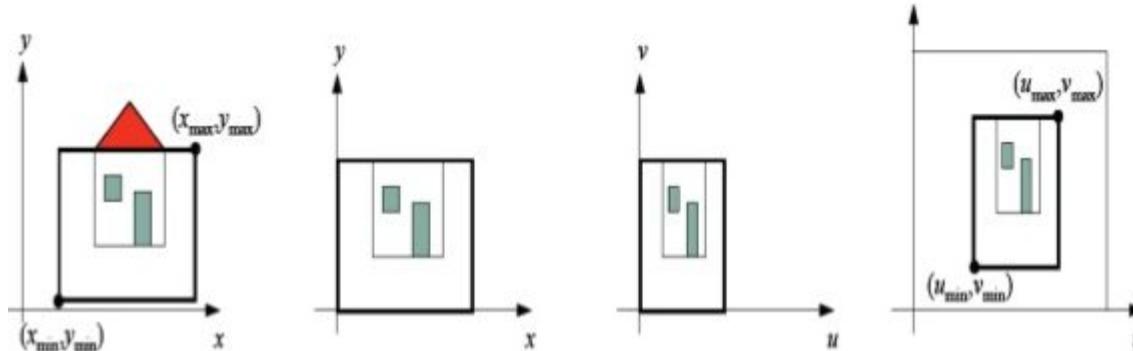
Let $R(\theta_1)$ and $R(\theta_2)$ be two successive rotations applied to a coordinate position P then the composite scaling matrix thus produced is

$$\text{or, } P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\} \quad \text{and} \quad P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$

Here the composite transformation matrix for this sequence of translation is

$$\begin{pmatrix} \cos\theta_2 & -\sin\theta_2 \\ \sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{pmatrix} \begin{pmatrix} \cos\theta_2 \cdot \cos\theta_1 & -\sin\theta_2 \cdot \sin\theta_1 & -\cos\theta_2 \cdot \sin\theta_1 & -\sin\theta_2 \cdot \cos\theta_1 \\ \sin\theta_2 \cdot \cos\theta_1 + \sin\theta_1 \cdot \cos\theta_2 & \cos\theta_2 \cdot \cos\theta_1 & -\sin\theta_2 \cdot \sin\theta_1 & \end{pmatrix} \\ = \begin{pmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) \end{pmatrix} R(\theta_1) = R(\theta_1 + \theta_2)$$

Window and View ports

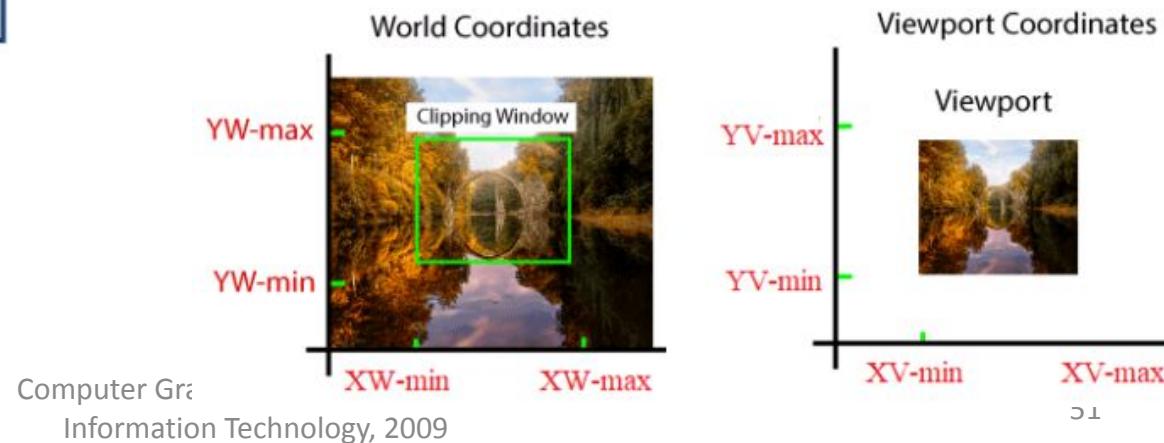
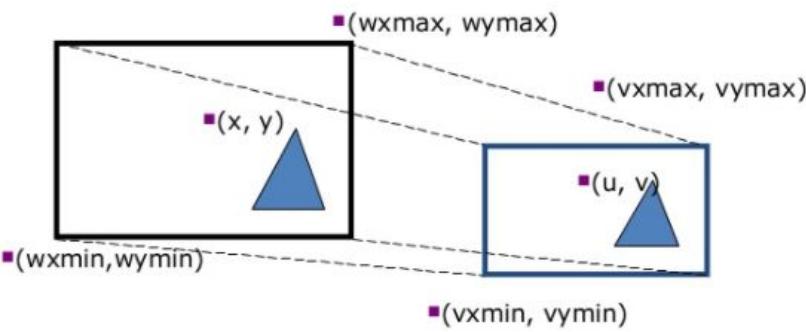


window
in world
coordinates

window
translated
to origin

window
scaled to size
of viewport

translated by
 (u_{\min}, v_{\min})
to final position



Window and View ports

A rectangular area specified in **world coordinates** is called a **window**.

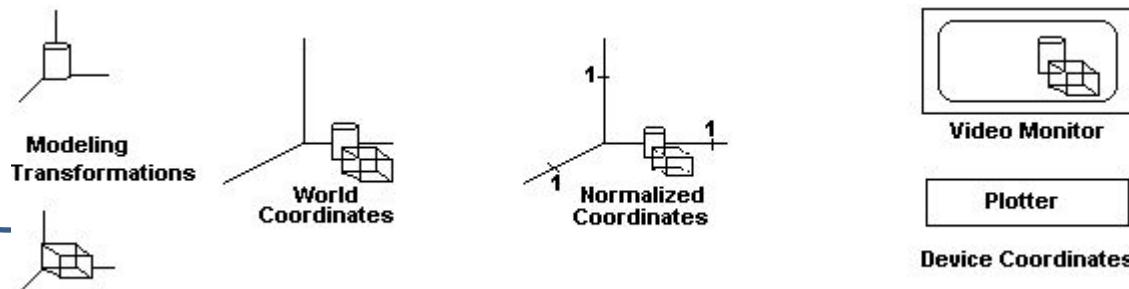
A rectangular area on the **display device** to which a **window is mapped** is called a **view port**.

The **window** defines **what** is to be viewed; the **view port** defines **where** it is to be displayed.

Often windows and view ports are rectangles in standard position with rectangle edges parallel to coordinate axes

For practical applications we need a transformation to translate and scale window to any size by moving it to specified rectangular area on screen

The mapping of a part of world coordinate scene to device coordinate is referred to as viewing transformation.

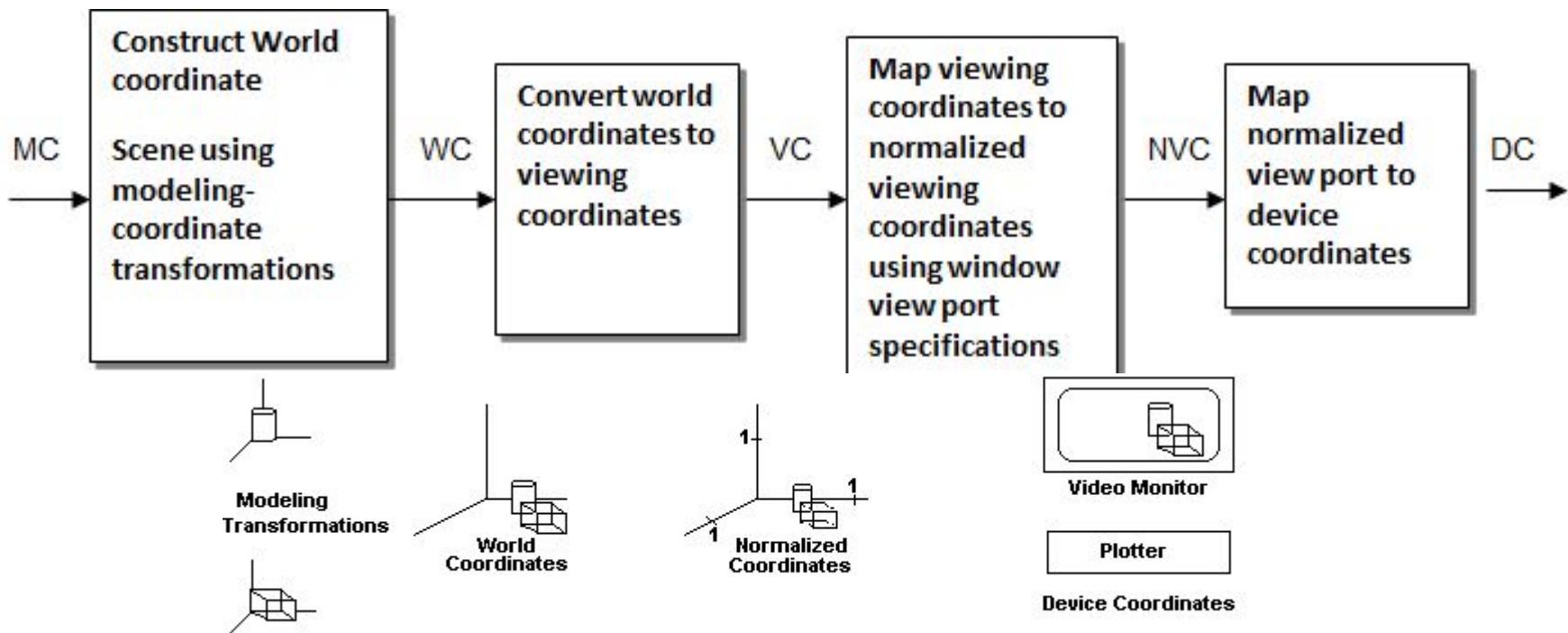


2-D Viewing Transformation Pipeline

A world-coordinate area selected for display is called a window, defines **what** is to be viewed. An area on a display device to which a window is mapped is called a viewport defines **where** it is to be displayed.

Viewing Transformation: The mapping of a part of a world-coordinate scene to device coordinates

Sometimes the two-dimensional viewing transformation is simply referred to as the **window-to-viewport transformation** or the **windowing transformation**. BUT, in general, viewing involves more than just the transformation from the window to the viewport.



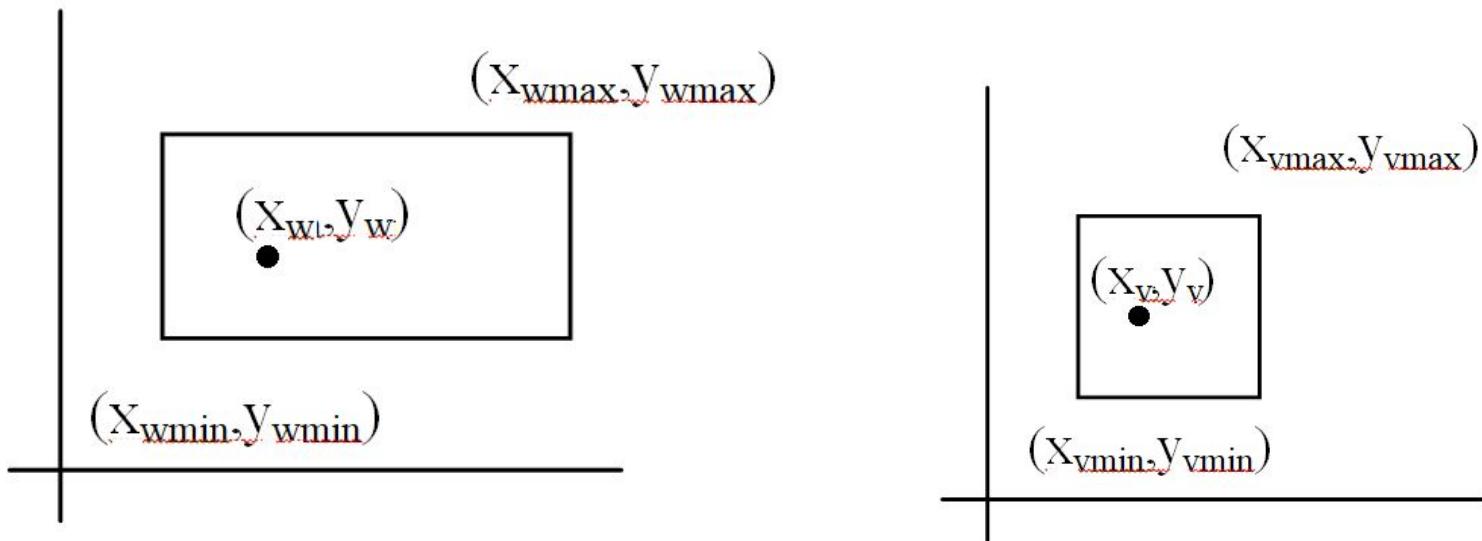
Window to Viewport Transformation

For displaying one has to **convert world coordinates into screen coordinates**

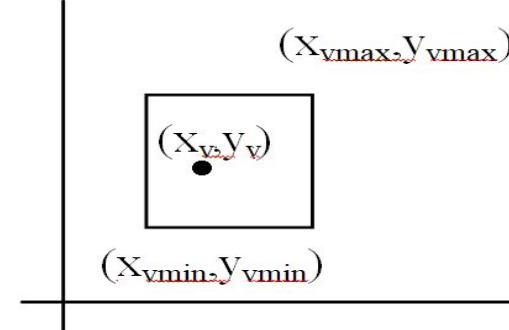
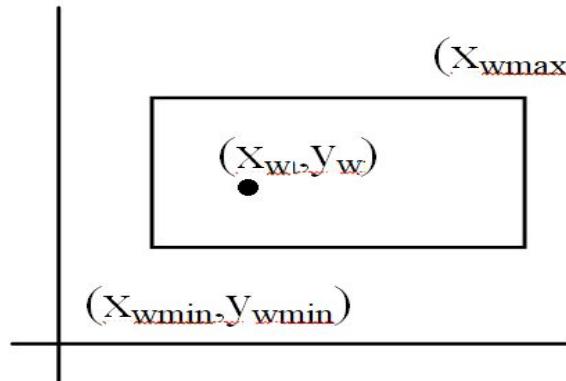
This transformation is called **viewing transformation**

A point in position (x_w, y_w) in window is mapped into position (x_v, y_v) in the viewport

To maintain same **relative placement** in view port as in window we require,



Window to Viewport Transformation



$$x_v - x_{vmin} = x_w - x_{wmin}$$

$$\frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$y_v - y_{vmin} = y_w - y_{wmin}$$

$$\frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

solving equations for view port position (x_v , y_v)

$$x_v = x_{vmin} + \frac{(x_w - x_{wmin})(x_{vmax} - x_{vmin})}{(x_{wmax} - x_{wmin})} = x_{vmin} + (x_w - x_{wmin}) \cdot s_x$$

$$y_v = y_{vmin} + \frac{(y_w - y_{wmin})(y_{vmax} - y_{vmin})}{(y_{wmax} - y_{wmin})} = y_{vmin} + (y_w - y_{wmin}) \cdot s_y$$

Window to Viewport Transformation (Matrix Form)

Window to View port Transformation (Viewing Transformation)

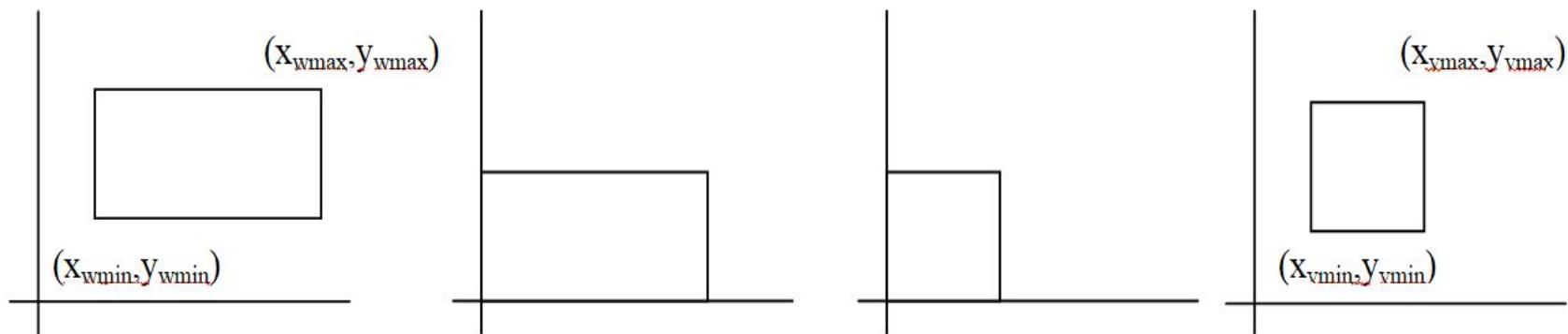
To transform a window to the view port we have to perform the following steps:

Step1: The object together with its window is translated until the lower left corner of the window is at the origin

Step2: The object and window are scaled until the window has the dimensions of the view port

Step3: Again translate to move the view port to its correct position on the screen

The overall transformation which performs these three steps called the viewing transformation. Let the window coordinates be (x_{wmin}, y_{wmin}) and (x_{wmax}, y_{wmax}) , view port coordinates be (x_{vmin}, y_{vmin}) and (x_{vmax}, y_{vmax})



Window to Viewport Transformation (Viewing Transformation)

Therefore the viewing transformation is as follows:

1. We have to translate the window to the origin by

$$T_x = -x_{w\min} \text{ and } T_y = -y_{w\min}$$

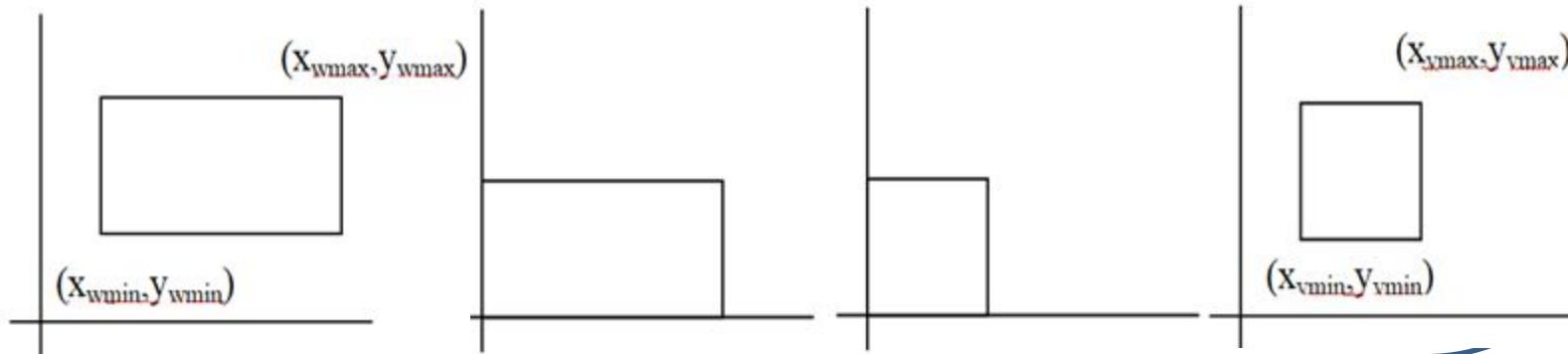
2. Then scale the window such that its size is matched to the view port using

$$S_x = (x_{v\max} - x_{v\min}) / (x_{w\max} - x_{w\min}) \quad S_y = (y_{v\max} - y_{v\min}) / (y_{w\max} - y_{w\min})$$

3. Again translate it by $T_x = x_{v\min}$ and $T_y = y_{v\min}$

All these steps can be represented by the following composite transformation:

$$CM = T_v * S_{wv} * T_w$$



Window to Viewport Transformation (Viewing Transformation)

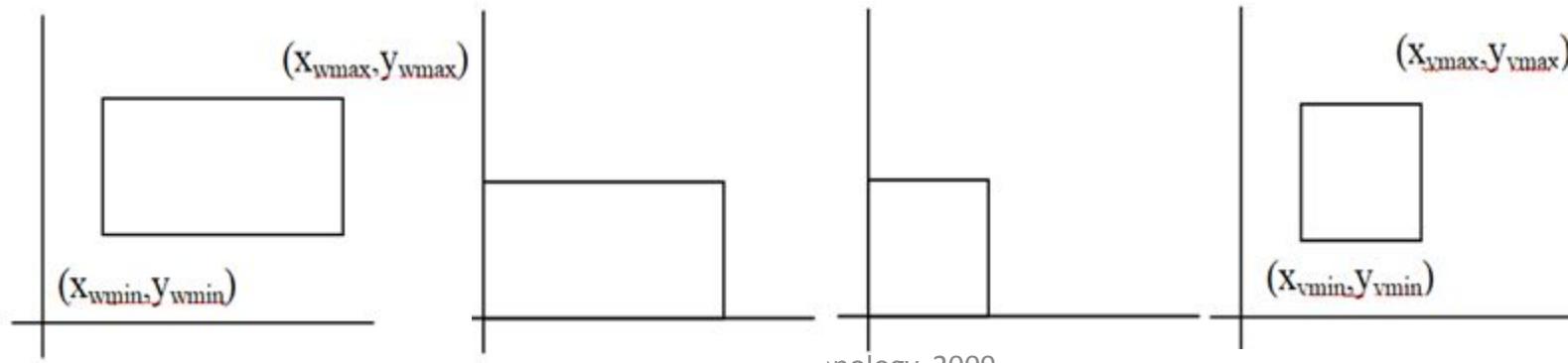
All these steps can be represented by the following composite transformation:

$$CM = T_v * S_{wv} * T_w$$

$$CM = \begin{pmatrix} 1 & 0 & x_{wmin} \\ 0 & 1 & y_{wmin} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_{wmin} \\ 0 & 1 & -y_{wmin} \\ 0 & 0 & 1 \end{pmatrix}$$

Here scaling factors are $s_x = (x_{vmax} - x_{vmin}) / (x_{wmax} - x_{wmin})$ $s_y = (y_{vmax} - y_{vmin}) / (y_{wmax} - y_{wmin})$

$$\begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix} \xrightarrow{CM} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix}$$



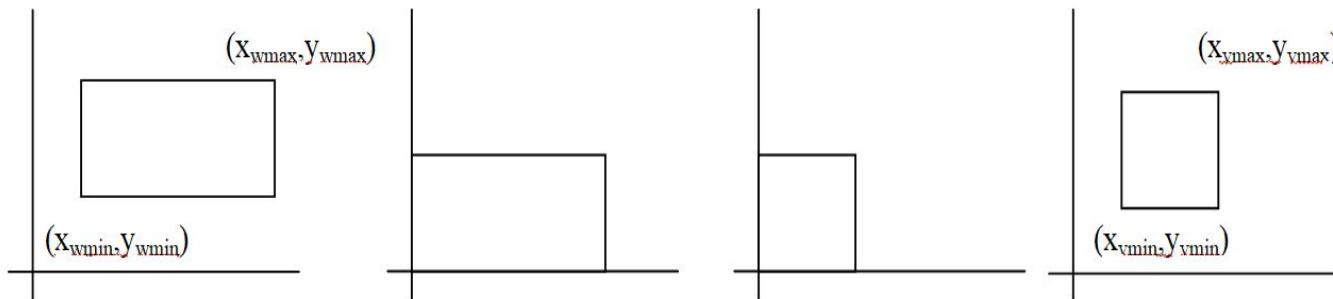
Exercise

A Triangle with vertices A(2,3) B(3,4) C(4,3) in world coordinate space with a window with its lowermost left corner at (10,10) and upper most right corner at (100,100) is required to be mapped to a viewport with its lowermost left corner at (20,20) and upper most right corner at (60,60)

$$x_v = x_{v\min} + (x_w - x_{w\min}) \cdot s_x = (12) \cdot (4)/(9) = 48/9 = 5.33$$

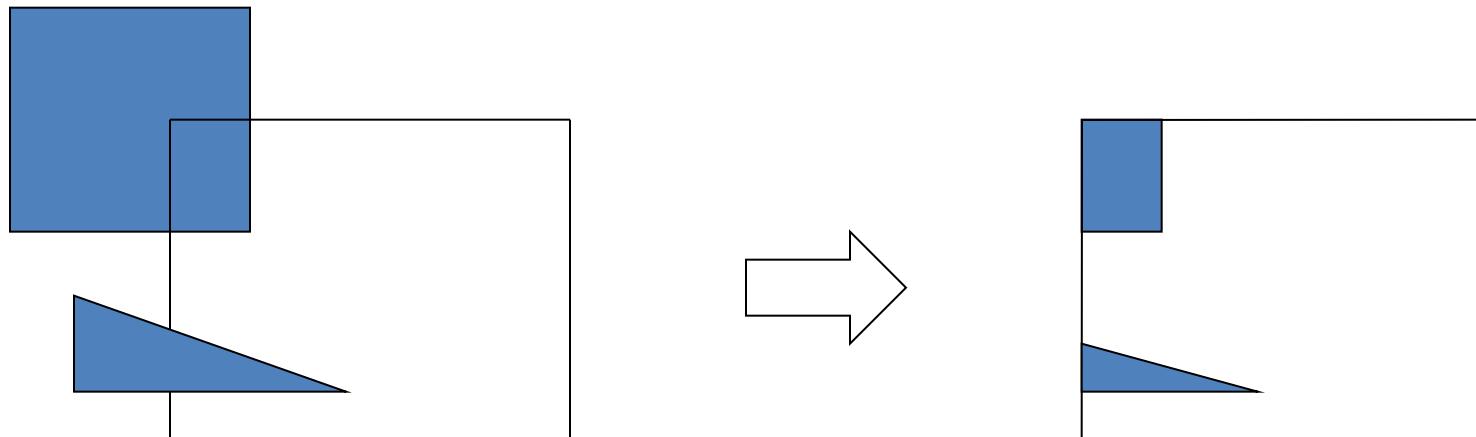
$$y_v = y_{v\min} + (y_w - y_{w\min}) \cdot s_y = (13) \cdot (4)/(9) = 52/9 = 5.78$$

A'B'C'



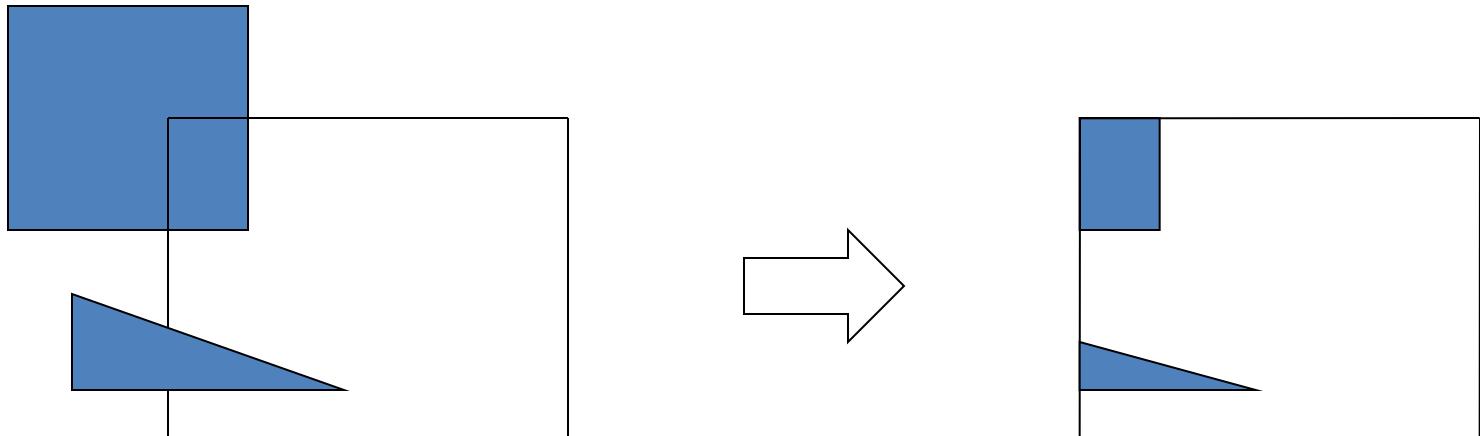
Clipping in Raster World

- Procedure that identifies those operations of picture that are either inside or outside
- The region against which an object is to be clipped is called a clip window.
- Depending on application it can be polygons or even curve surfaces.



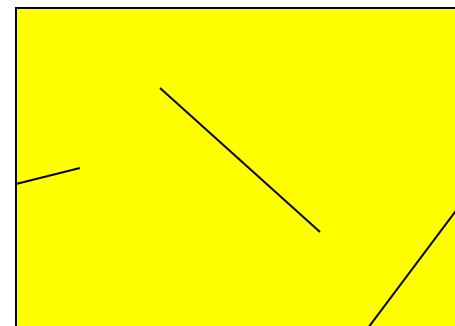
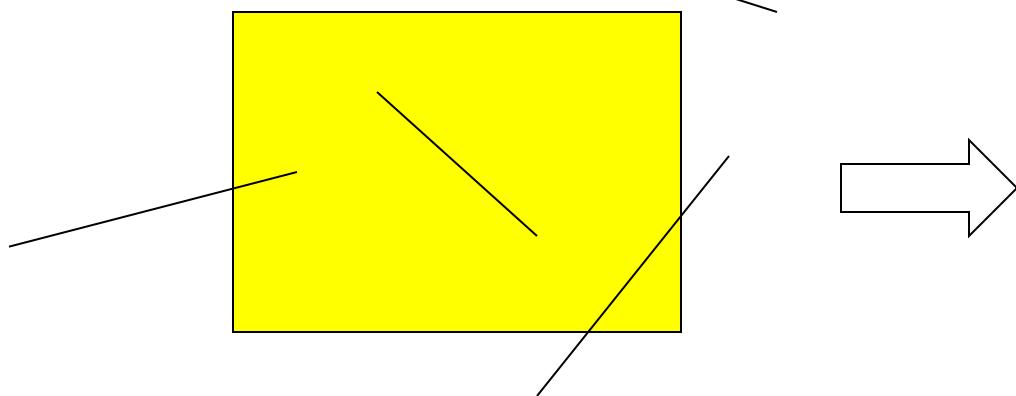
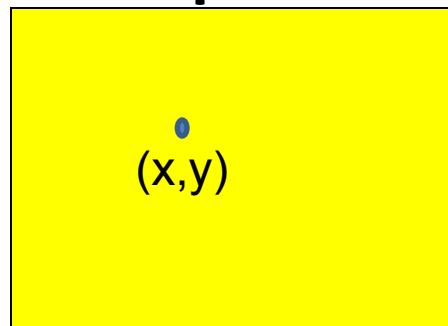
Clipping in Raster World

- Applications
 - i. Extracting parts **of defined scene for viewing**
 - ii. Identifying visible surfaces **in three dimension views**
 - iii. Drawing, painting operations that allow parts **of picture to be** selected for copying, moving, erasing or duplicating **etc.**



Line Clipping

- Point clipping easy: Just **check the inequalities** (x_{\max}, y_{\max})
 - $x_{\min} < x < x_{\max}$
 - $y_{\min} < y < y_{\max}$
- Line clipping more tricky (x_{\min}, y_{\min})



Cohen-Sutherland Line Clipping

- Divide 2D space into 3×3 regions.
- Middle region is the **clipping window**.
- Each region is assigned a 4-bit code.
- Bit 1 is set to 1 if the region is to the **left** of the clipping window, otherwise. Similarly for bits 2, 3 and 4.



4

3

2

1

Top

Bottom

Right

Left

1001

1000

1010

 $(x_{w\max}, y_{w\max})$

0001

0000

0010

 $(x_{w\min}, y_{w\min})$

0101

0100

0110

Cohen-Sutherland Line Clipping

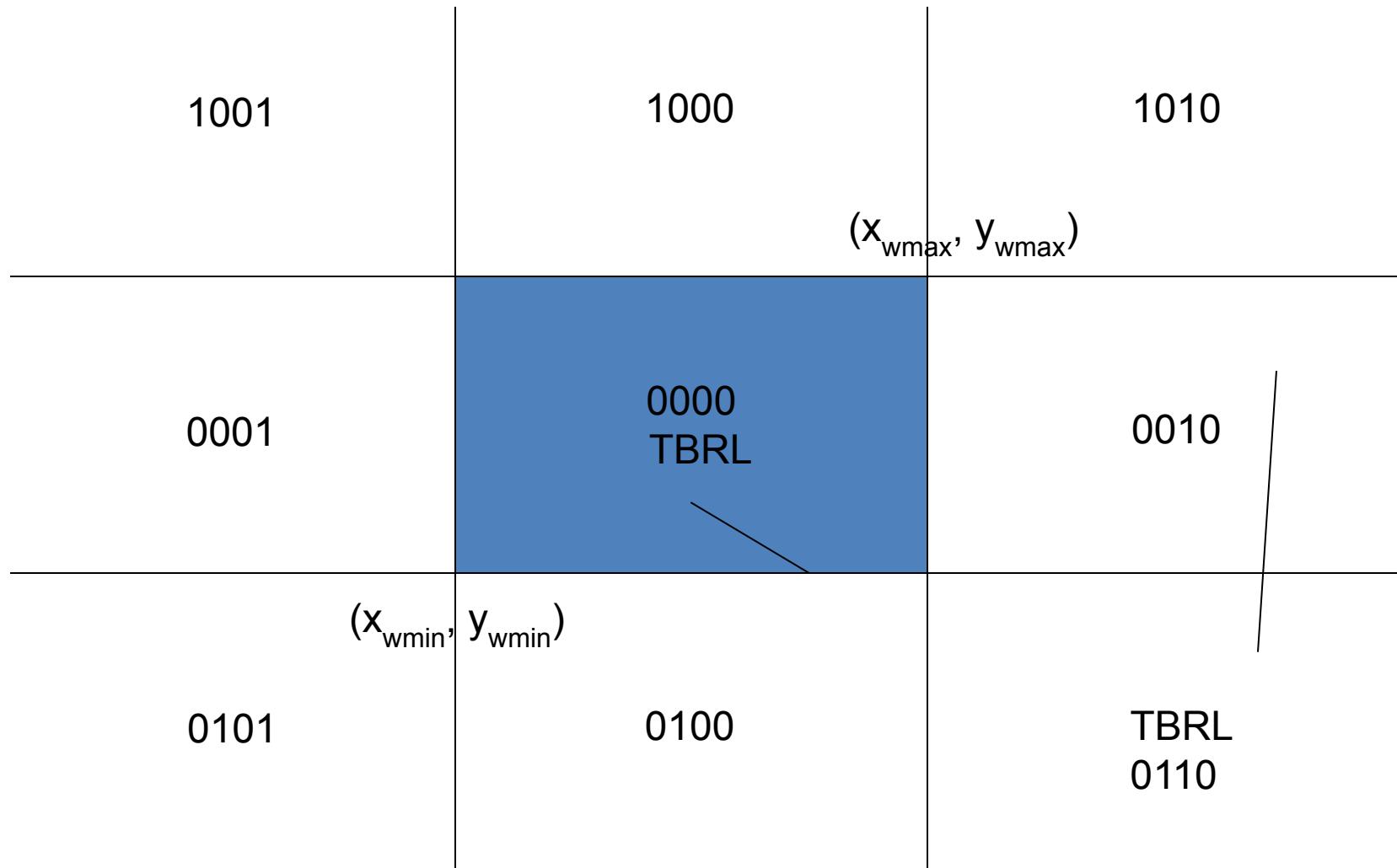
- To clip a line, find out which regions its two endpoints lie in.

Case I

- If they are **both in** region 0000, then it's completely in.

Cohen-Sutherland Line Clipping

TBRL



Cohen-Sutherland Line Clipping

Case II

- If the two region numbers both have a 1 in the same bit position, the line is **completely out**.

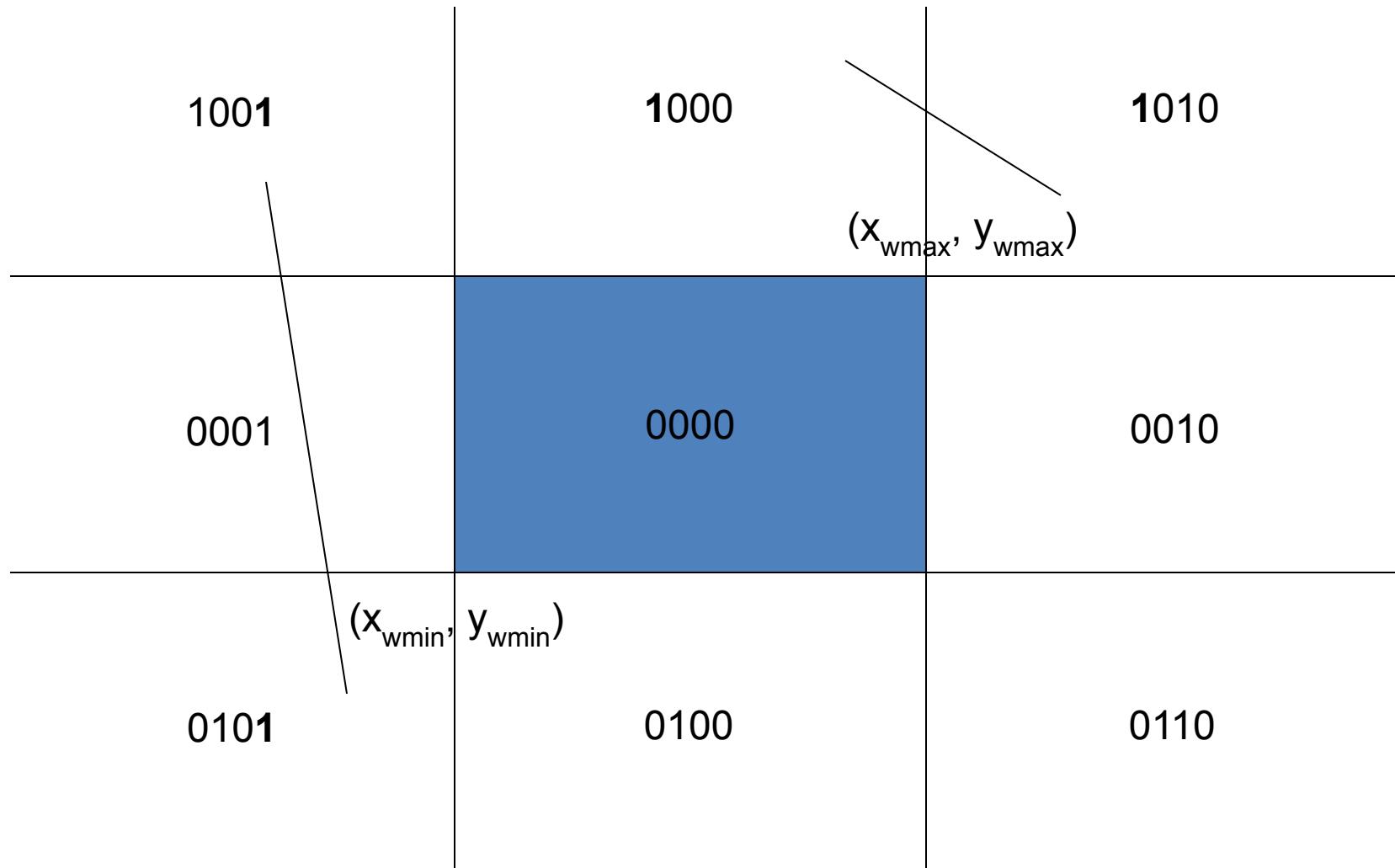
TBRL

0010 A

1010 B

0000 X AND

Cohen-Sutherland Line Clipping

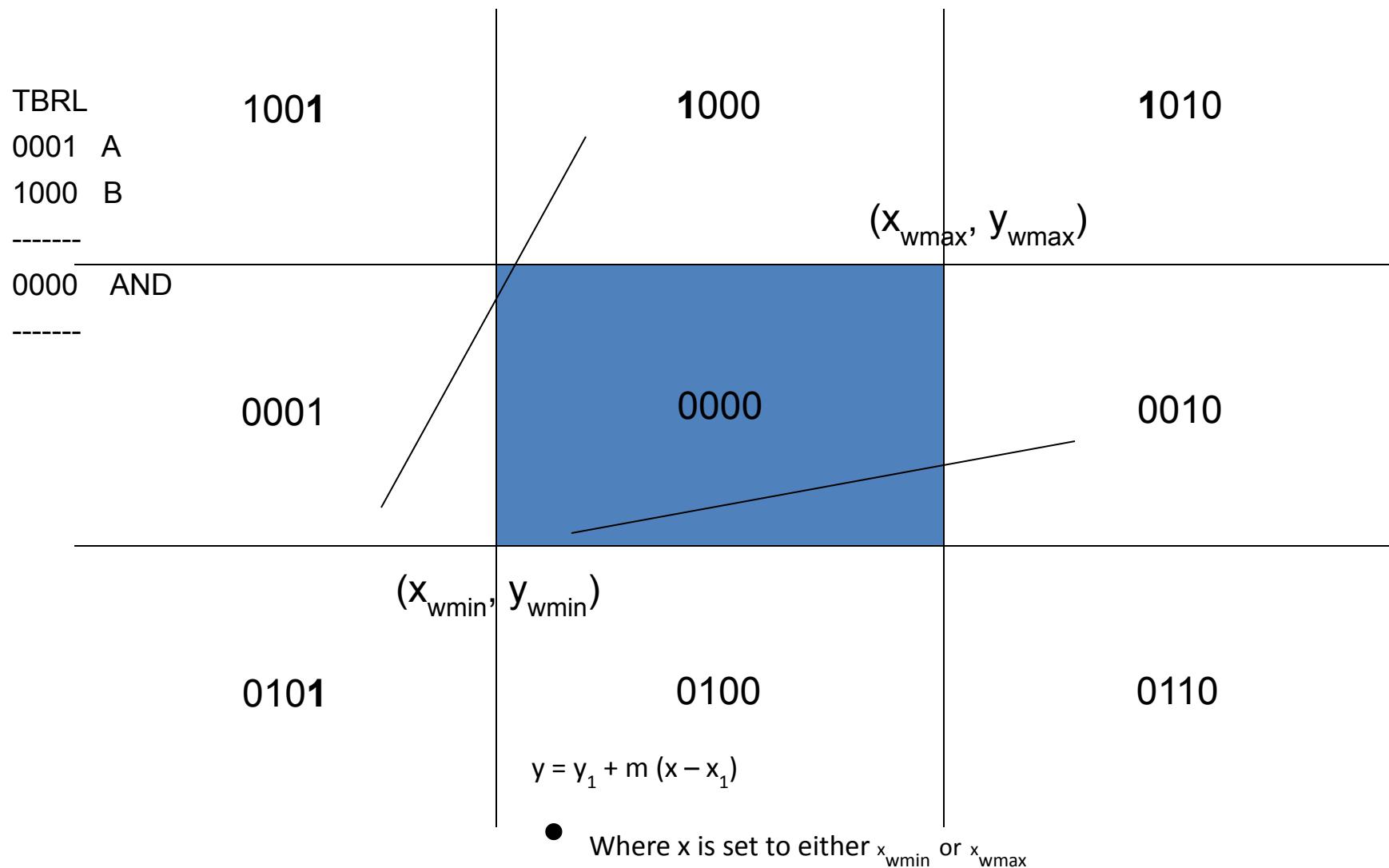


Cohen-Sutherland Line Clipping

Case III

- If lines can not be identified as completely inside or outside we have to **do some more calculations.**
- Here we find the intersection points with a clipping boundary using the slope intercept form of the line equation

Cohen-Sutherland Line Clipping



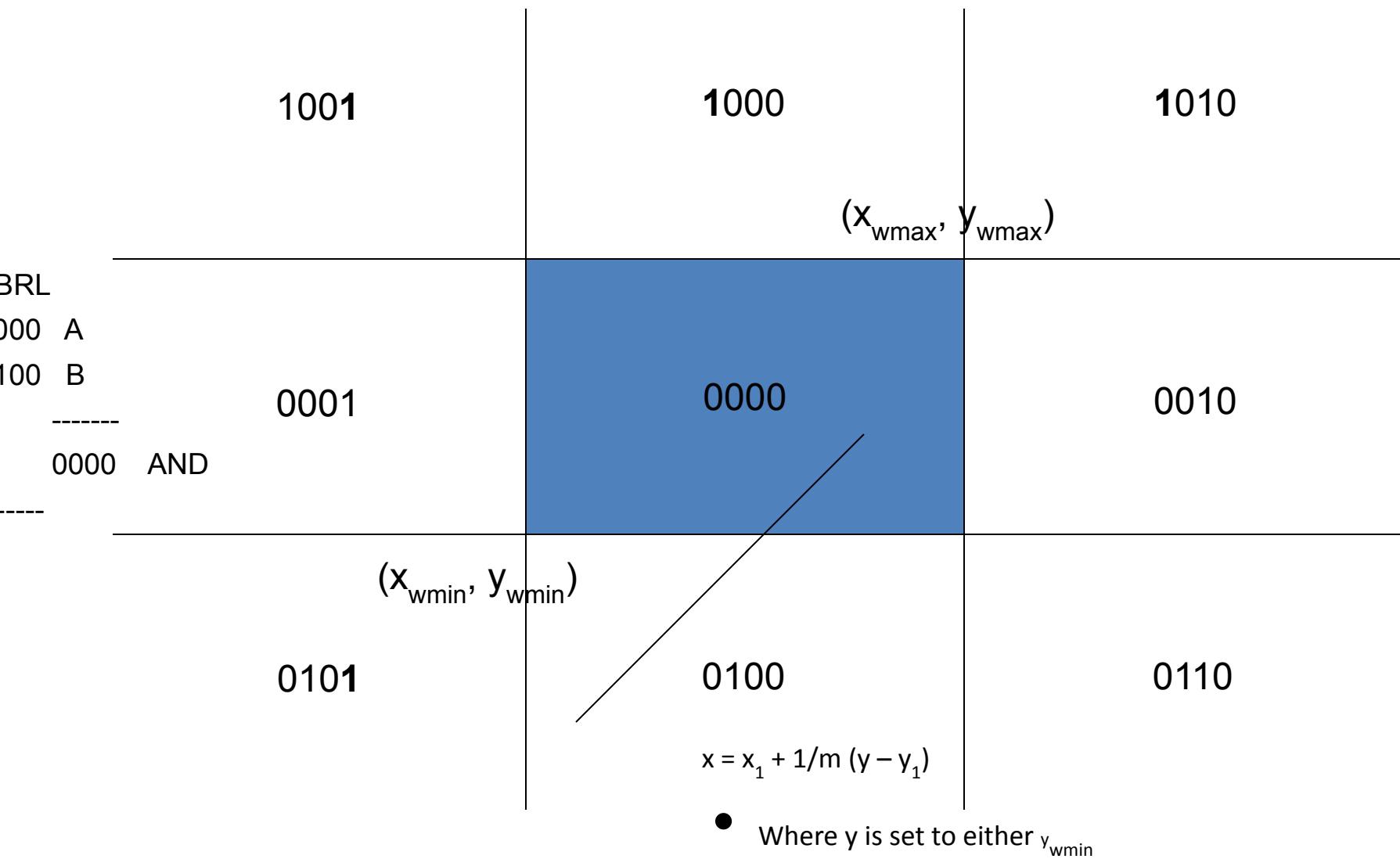
Cohen-Sutherland Line Clipping

- For a line with end point coordinates (x_1, y_1) and (x_2, y_2) the y coordinate of the intersection point with a vertical boundary is

$$y = y_1 + m (x - x_1)$$

- Where x is set to either x_{wmin} or x_{wmax}

Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

- Similarly for the intersection with a vertical boundary

$$x = x_1 + (y - y_1)/m$$

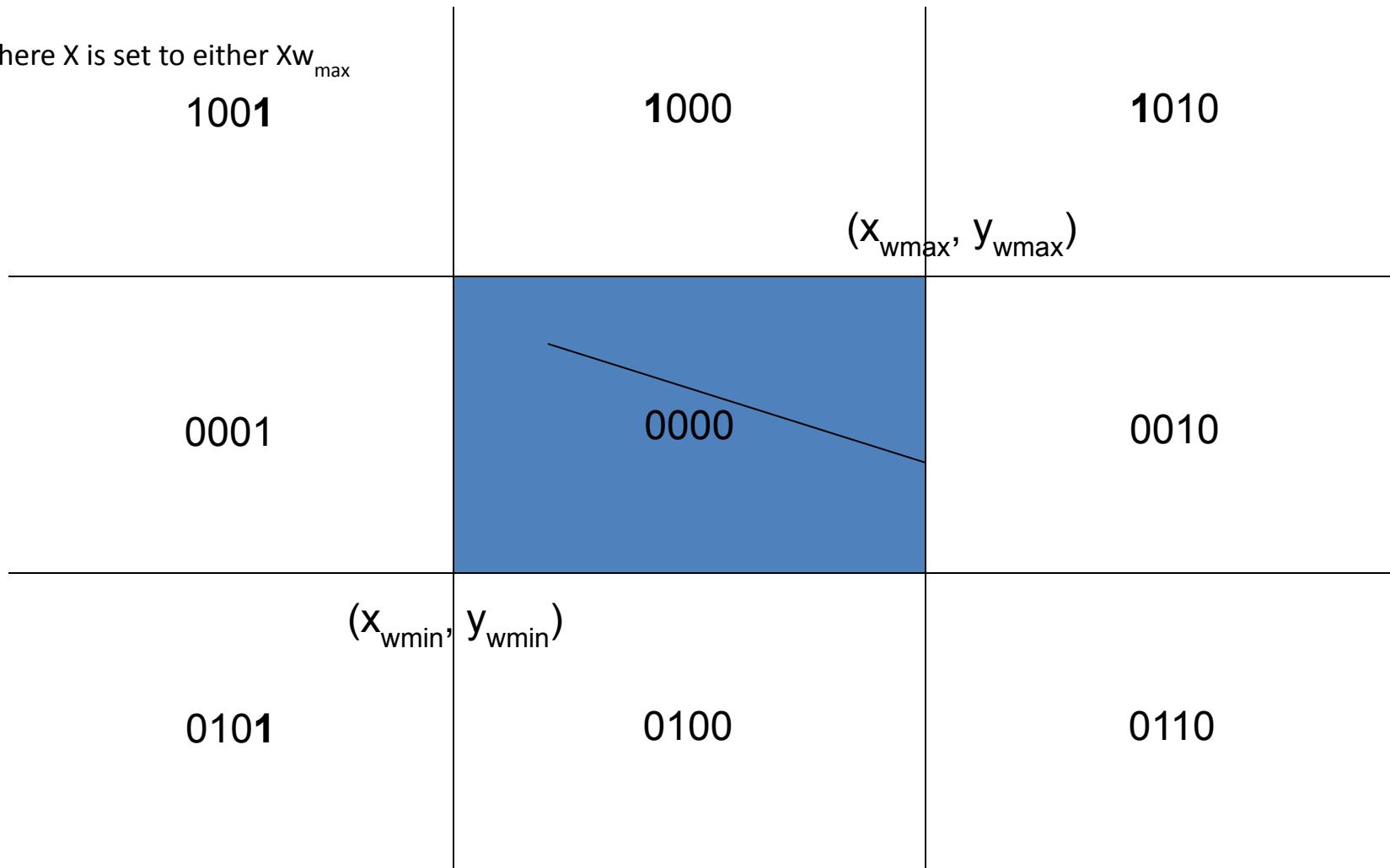
- Where y is set to either $y_{w_{\min}}$ or $y_{w_{\max}}$

Cohen-Sutherland Line Clipping

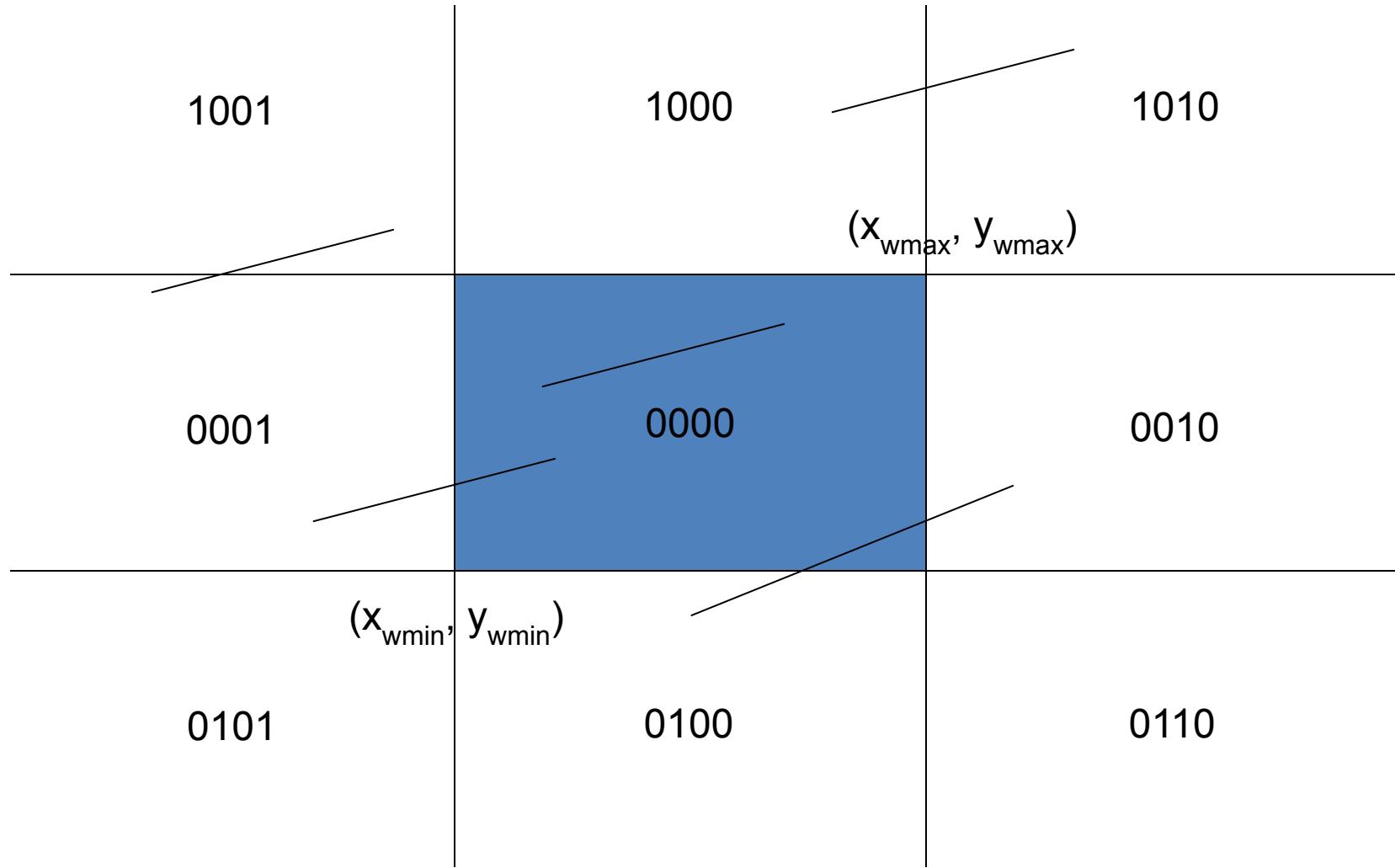
- Similarly for the intersection with a vertical boundary

$$y = y_1 + (x - x_1)m$$

- Where X is set to either $x_{w_{\max}}$



Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping In Detail

Cohen-Sutherland Line Clipping

It is based on a **coding scheme**, makes clever use of bit operations to perform this test efficiently

For each endpoint of a line, a **4 bit binary code** is used

Lower order (**Bit 1**) is set to 1 if the end point is at the **left** side of the window otherwise set to 0

Bit 2, is set to 1 if the end point is at the **right** side of the window otherwise set to 0

Bit 3, is set to 1 if the end point is at the **bottom** of the window otherwise set to 0

Bit 4, is set to 1 if the end point is **above** the window otherwise set to 0

By numbering bit positions in region code as 1 – 4 from right to left coordinate regions can be correlated with bit positions as

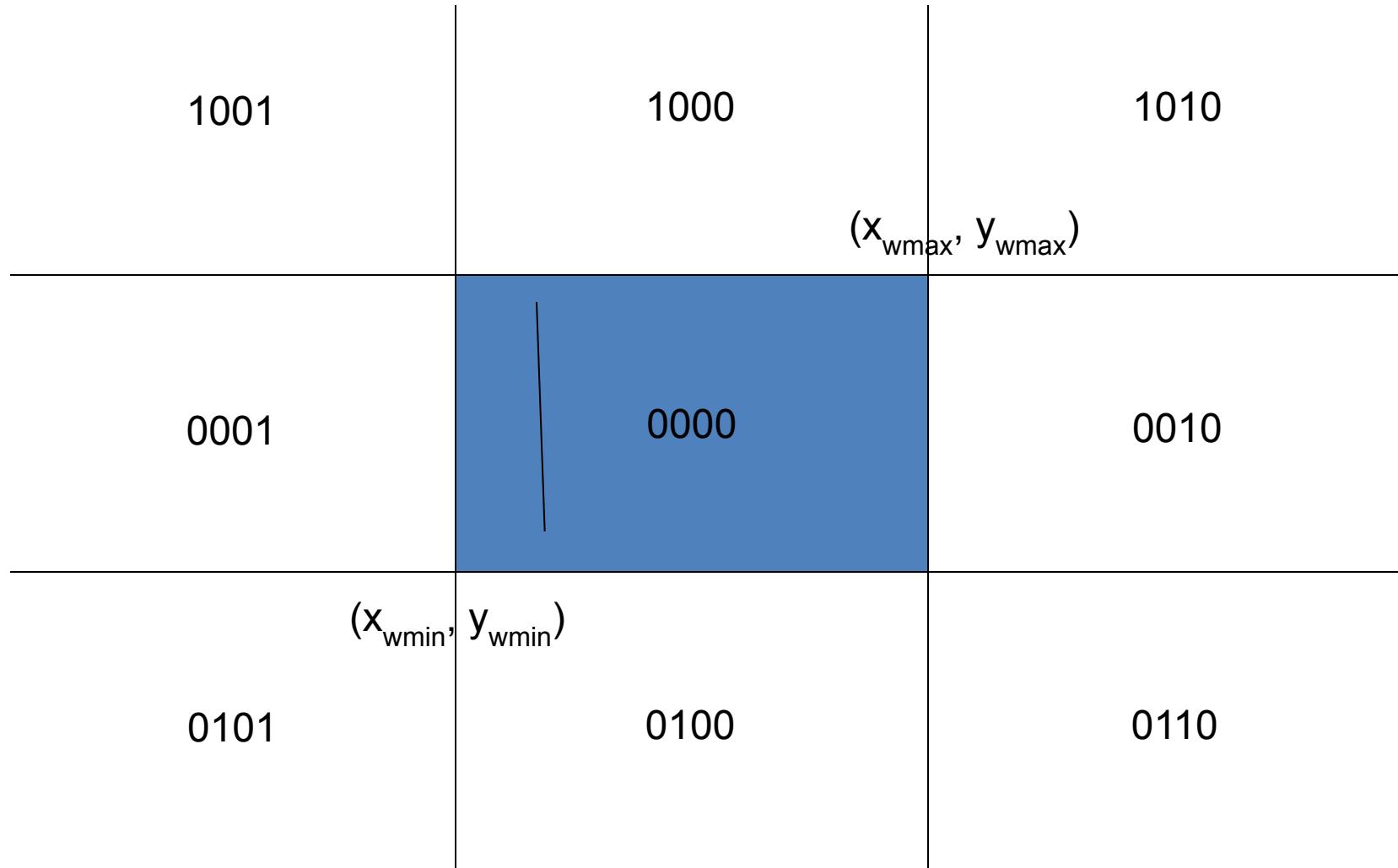
Bit 4	Bit 3	Bit 2	Bit 1
x	x	x	x
up	bottom	right	left

Value of 1 in any bit position indicates that the point is in that relative position otherwise bit position is set to 0.

if point is within clipping rectangle, region code is **0000**.

if point is below and to the left of rectangle then region code is **0101**.

Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

Algorithm:

Step 1: Establish the region codes for all line end points:

Bit 1 is set to 1 if $x < x_{w\min}$ otherwise set it to 0

Bit 2 is set to 1 if $x > x_{w\max}$ otherwise set it to 0

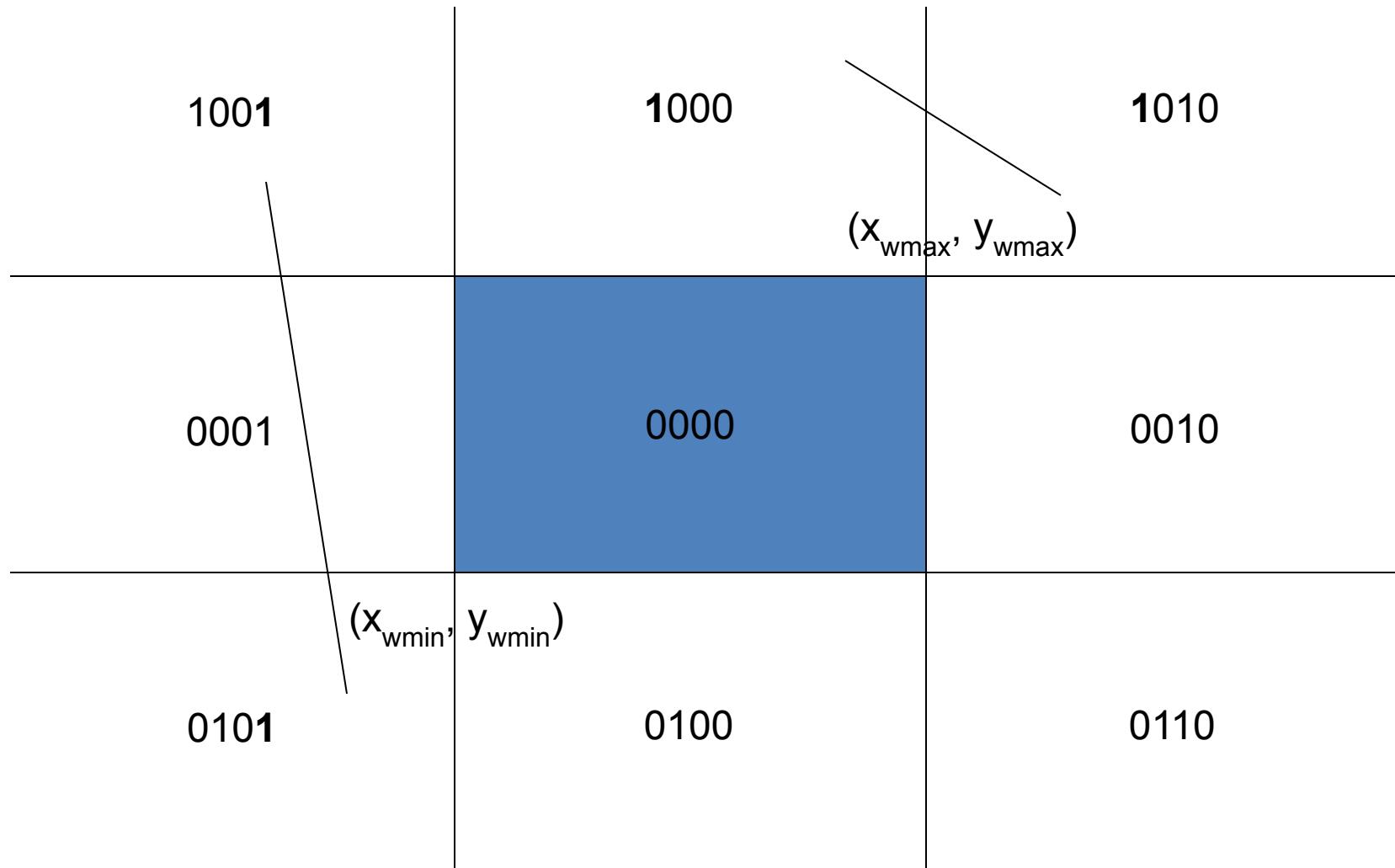
Bit 3 is set to 1 if $y < y_{w\min}$ otherwise set it to 0

Bit 4 is set to 1 if $y > y_{w\max}$ otherwise set it to 0

Step 2: Determine which lines are completely inside the window and which are completely outside , using the following tests:

- If **both end points** of line have **region codes 0000** line is **completely inside** window
- If the **logical AND operation of the region codes** of the two end points is **NOT 0000** then the line is **completely outside** (same bit position of the two end points have 1)

Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

Step 3: if both the tests in step 2 fail then the line is **not completely inside nor outside**. so we need to find out the intersection with the boundaries of the window

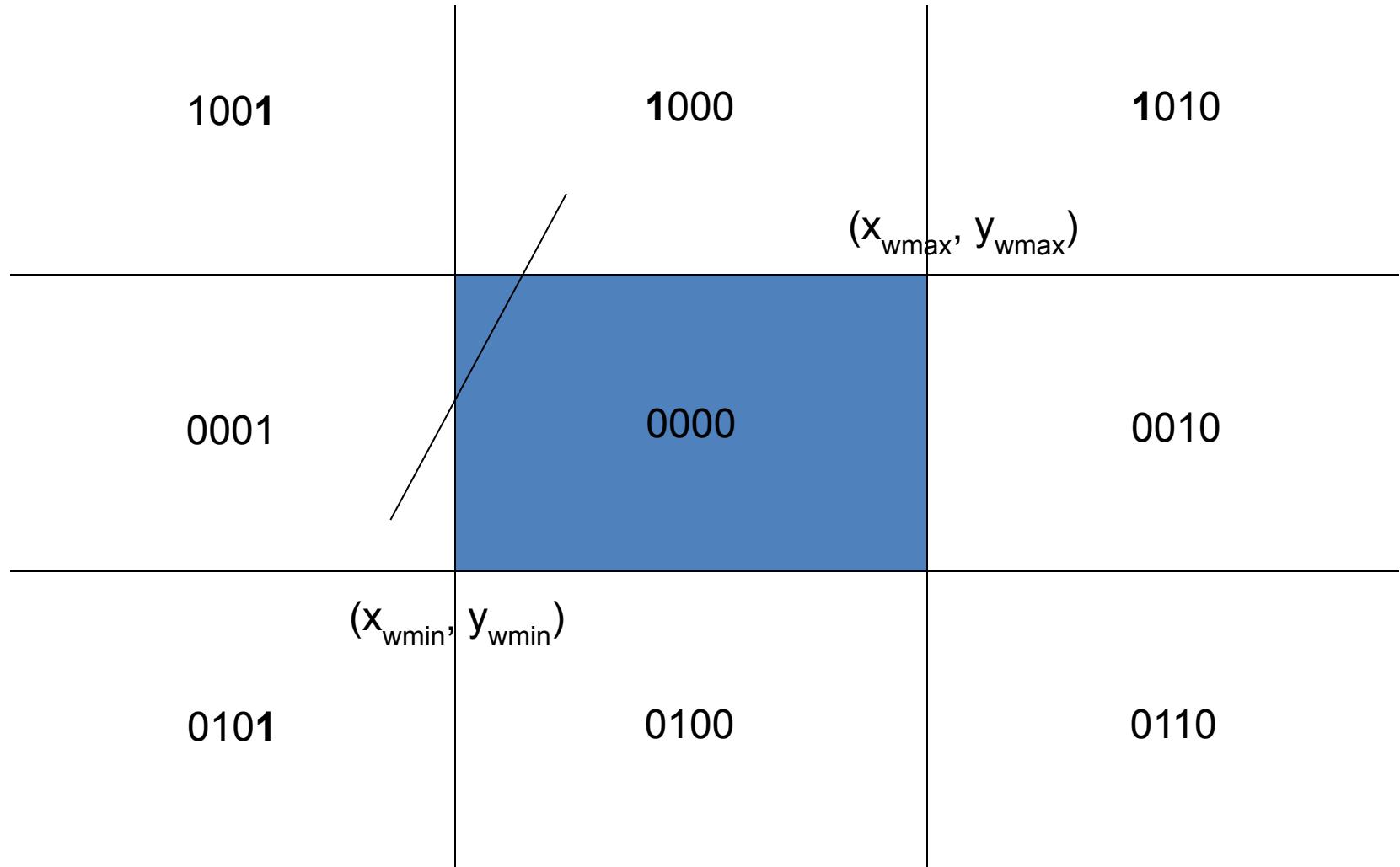
$$\text{slope } (m) = (y_2 - y_1)/(x_2 - x_1)$$

- a. If **bit 1 is 1** then the line intersects with the **left boundary** and $y_i = y_1 + m * (x - x_1)$ where $x = x_{w\min}$
- b. If **bit 2 is 1** then the line intersects with the **right boundary** and $y_i = y_1 + m * (x - x_1)$ where $x = x_{w\max}$
- c. If **bit 3 is 1** then line intersects with the **bottom boundary** and $x_i = x_1 + (y - y_1)/m$ where $y = y_{w\min}$
- d. If **bit 4 is 1** then line intersects with the **upper boundary** and $x_i = x_1 + (y - y_1)/m$ where $y = y_{w\max}$

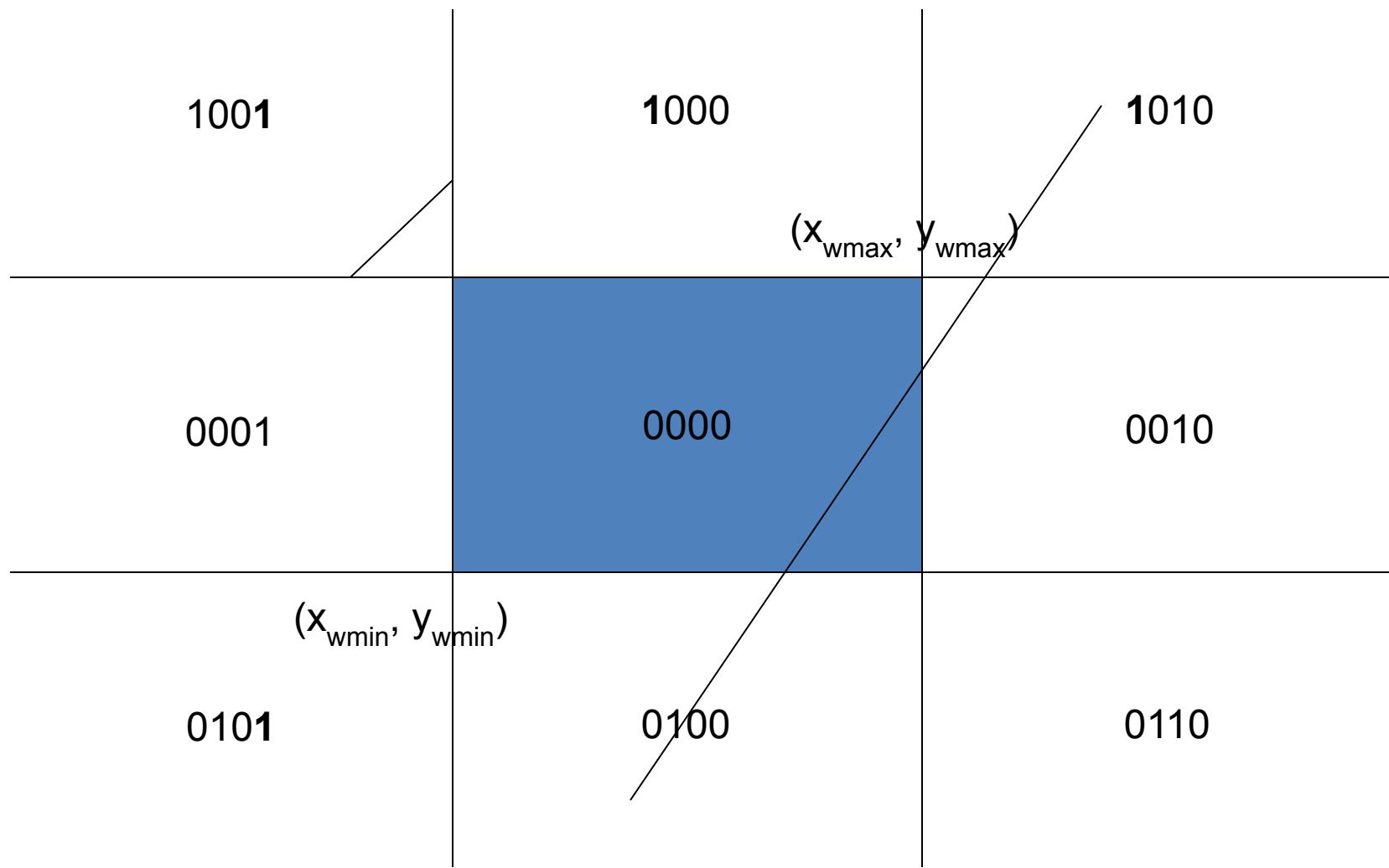
Here, x_i and y_i are the x and y intercepts for that line

Step 4: repeat Step1 to Step3 until the line is completely accepted or rejected

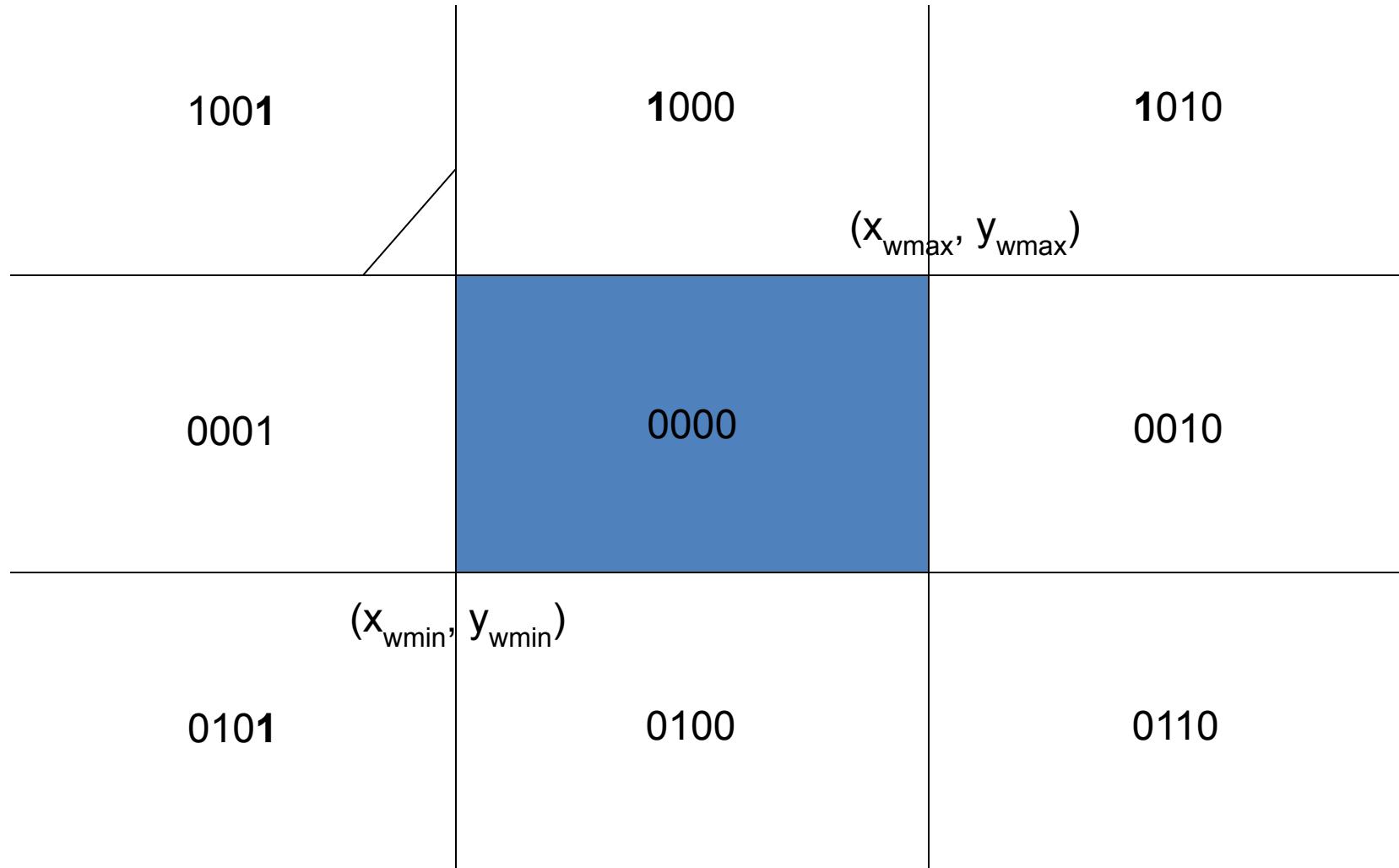
Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(5,20) ,B(60,70) against a clip window with its lower left corner at (10,10) and upper right corner at (100,100)

Hints:

- i. Find the region code for each end point of the line
- ii. Find if it is above, below, left or right of the clip window
- iii. Determine the intersection point of the line segment with the boundary

Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(5,20) ,B(60,70) against a clip window with its lower left corner at (10,10) and upper right corner at (100,100)

Step 1: Establish the region codes for all line end points: TBRL

Region code for A

5 < 10 True Bit 1 = 1

5 > 100 False Bit 2 = 0

20 < 10 False Bit 3 = 0

20 > 100 False Bit 4 = 0

Region code for B

Left 60 < 10 False Bit 1 = 0

Right 60 > 100 False Bit 2 = 0

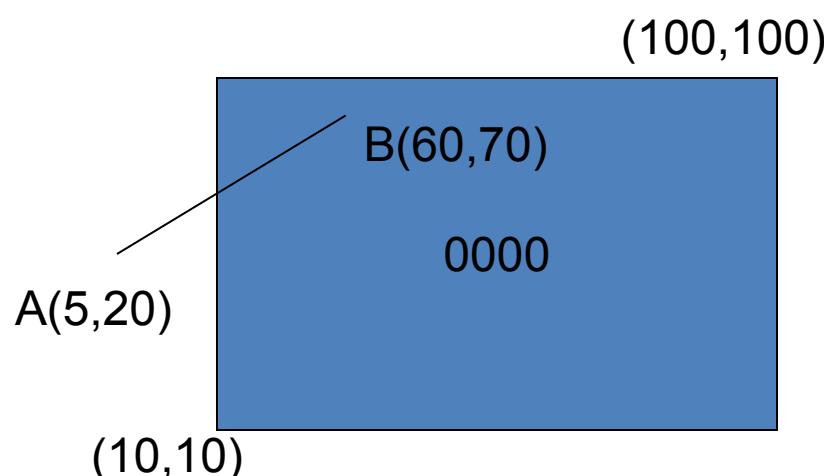
Bottom 70 < 10 False Bit 3 = 0

Top 70 > 100 False Bit 4 = 0

TBRL

So, **Region code for A(0001)**

Region code for B(0000)



Cohen-Sutherland Line Clipping

Region code for A()

Region code for B()

Step 2: Determine which if the line is completely inside the window and or is completely outside:

- a. **both end points** of line **DON'T** have **region codes 0000** so the line is **NOT completely inside window** false
- b. Perform Logial AND operation of region codes of two line end points

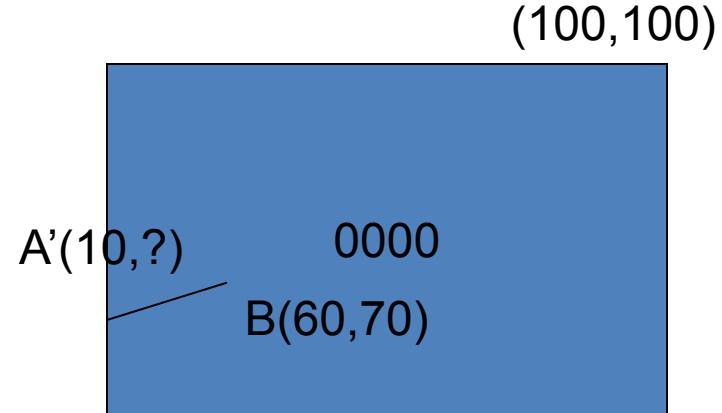
Region code for A 0001

Region code for B 0000

AND

0000

A(5,20)



Logical AND operation of the region codes of the two end points yields **0000** so it is the case of **partial visibility**(The line end point A is towards the left side of clip window and the line end point B is inside the clip window)

Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(5,20) ,B(60,70) against a clip window with its lower left corner at (10,10) and upper right corner at (100,100)

Step 3: Both the tests in step 2 failed so line is **not completely inside nor outside** . so we need to find out the intersection with the boundaries of the window

$$\text{slope } (m) = (y_2 - y_1)/(x_2 - x_1) = 0.91$$

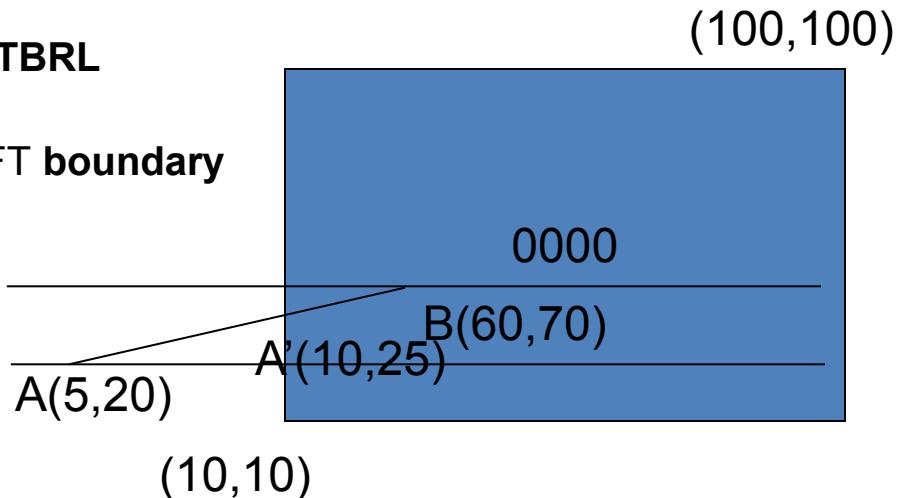
In case of line end point A(5,20) $x_1 = 5$ $y_1 = 20$ **TBRL**

a. **bit 1 is 1** then the line intersects with the **LEFT boundary**

so $x = x_{w\min} = 10$

$$y_i = y_1 + m * (x - x_1)$$

$$y_i = 20 + 0.91 * 5 = 24.54$$



So end point coordinates of line with endpoints A(5,20) ,B(60,70) after clipping : A'(10,25) ,B'(60,70)

Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(105,30) ,B(55,40) against a clip window with its lower left corner at (10,10) and upper right corner at (100,100)

Step 1: Establish the region codes for all line end points:

Region code for A

$105 < 10$ False Bit 1 = 0

$105 > 100$ True Bit 2 = 1

$30 < 10$ False Bit 3 = 0

$30 > 100$ False Bit 4 = 0

Region code for B

Left $55 < 10$ False Bit 1 = 0

Right $55 > 100$ False Bit 2 = 0

Bottom $40 < 10$ False Bit 3 = 0

Top $40 > 100$ False Bit 4 = 0

Region code for A(0010)

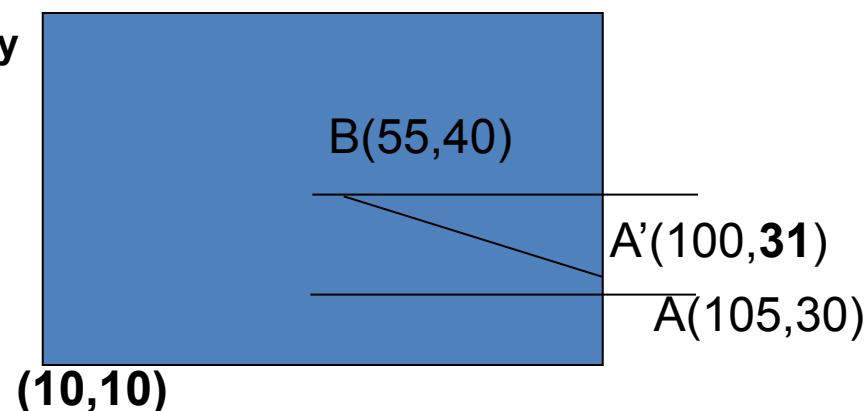
Region code for B(0000)

bit 2 is 1 then the line intersects with the RIGHT boundary

so $x = XWMAX = 100$

$$y_i = y_1 + m * (x - x_1)$$

$$y_i = 30 + m * (100 - 105) = 31$$

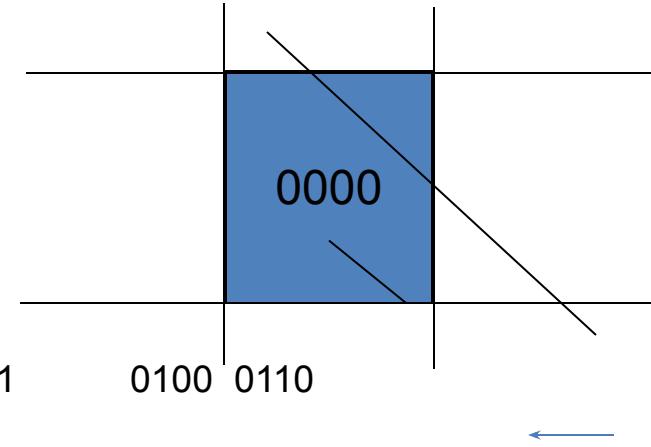


Cohen-Sutherland Line Clipping

Clip a line with end point coordinates A(-1,6) B(5,-8) against a clip window with its lower left corner at (-2,-5) and upper right corner at (4,8) using Cohen-Sutherland algorithm.

Step 1: Establish the region codes for all line end points:

Region code for A			Region code for B		
-1<-2	False	Bit 1 = 0	Left	5<-2	False Bit 1 = 0
-1> 4	True	Bit 2 = 0	Right	5> 4	False Bit 2 = 1
6< -5	False	Bit 3 = 0	Bottom	-8< -5	False Bit 3 = 1
6> 8	False	Bit 4 = 0	Top	-8> 8	False Bit 4 = 0



Region code for A(0000) Region code for B(0110) TBRL

**Step 2: Perform Logical AND operation of region codes of line end points yields 0000 partial visibility
bit 2 ,3 and bit is then line intersects with right then bottom boundary**

Step 3: Find intersection points of the line with window boundaries

$$\text{so } x = x_{w\max} = 4$$

$$y = y_{w\min} = -5$$

$$y_i = y_1 + (x - x_1)m = -8 + 2.32 = -5.7$$

$$x_i = 4 + 0.7/-2.33$$

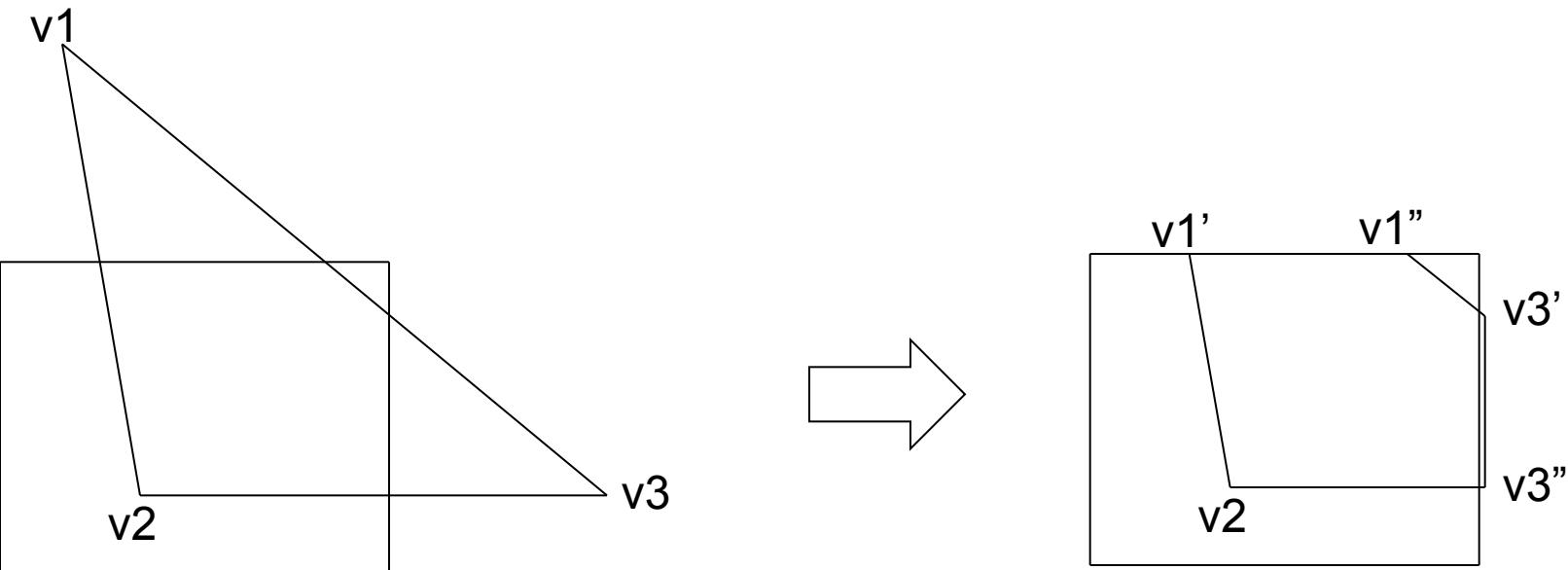
$$x_i = 4$$

$$x_i =$$

$$5. - \quad -5.6$$

$$5.4 \quad -5$$

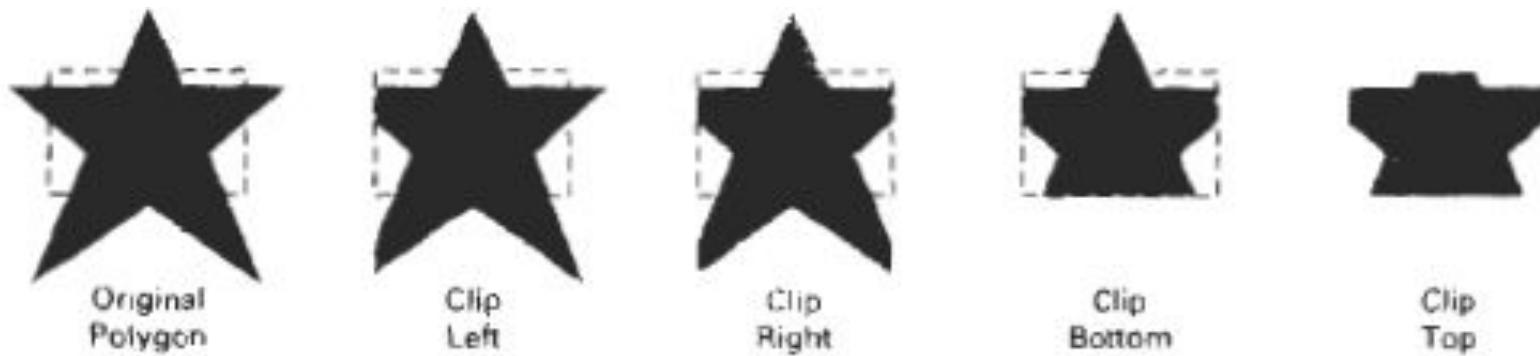
Polygon Fill-Area Clipping



Note: Need to consider each of 4 edge boundaries

Sutherland-Hodgman Polygon Clipping

- Process polygon boundary as a whole against each (clip) window edge
- Take initial set of polygon vertices, clip the polygon against, left , right, bottom and top boundary clipper.
- At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper

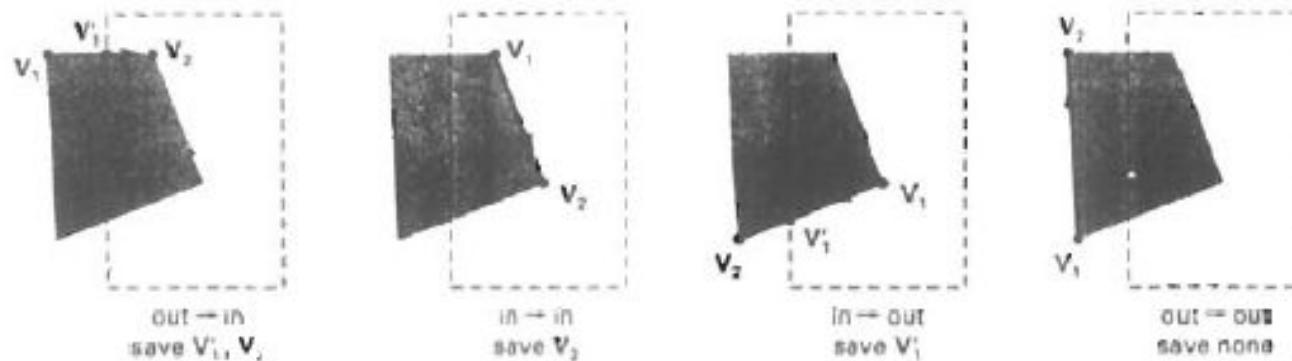


Sutherland-Hodgman Polygon Clipping

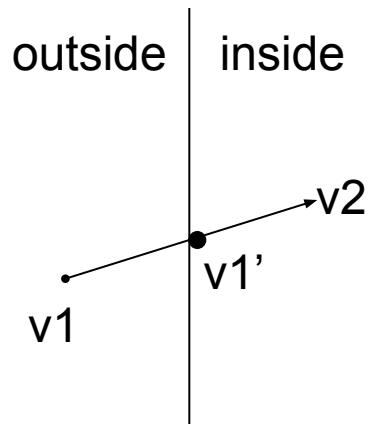
Four possible cases when processing vertices in sequence around perimeter of a polygon:

As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

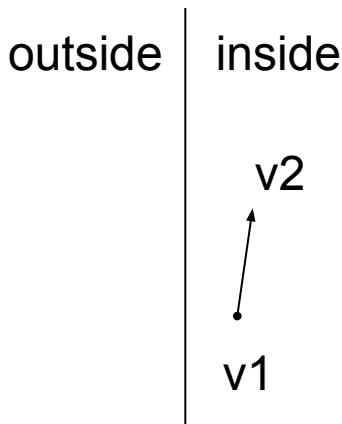
- **Case 1:** If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list.
- Case 2: If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.
- **Case 3:** If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.
- **Case 4:** If both input vertices are outside the window boundary, nothing is added to the output list.
- Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.



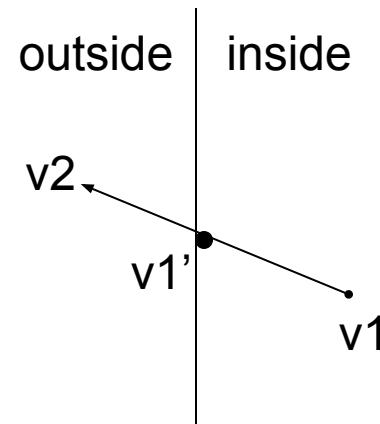
Sutherland-Hodgman Polygon Clipping: Four possible scenarios at each clipper



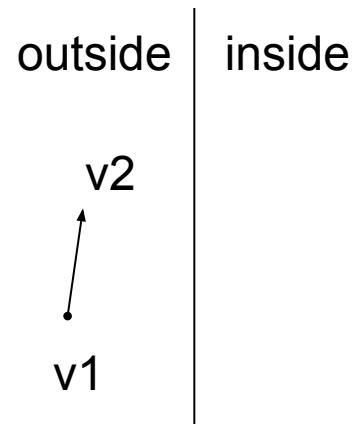
Outside to inside:
Output: v_1' and v_2



Inside to inside:
Output: v_2

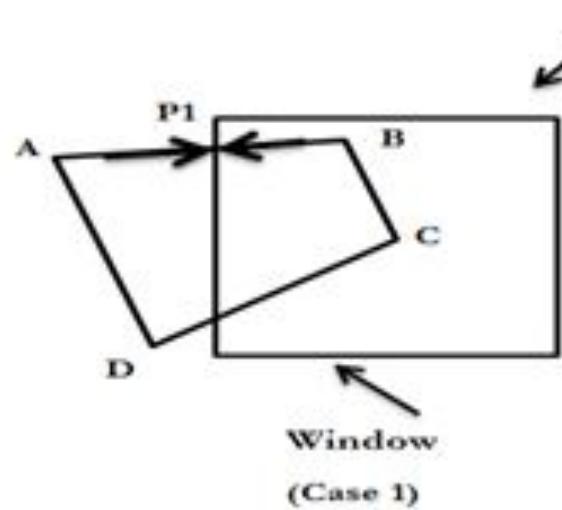


Inside to outside:
Output: v_1'

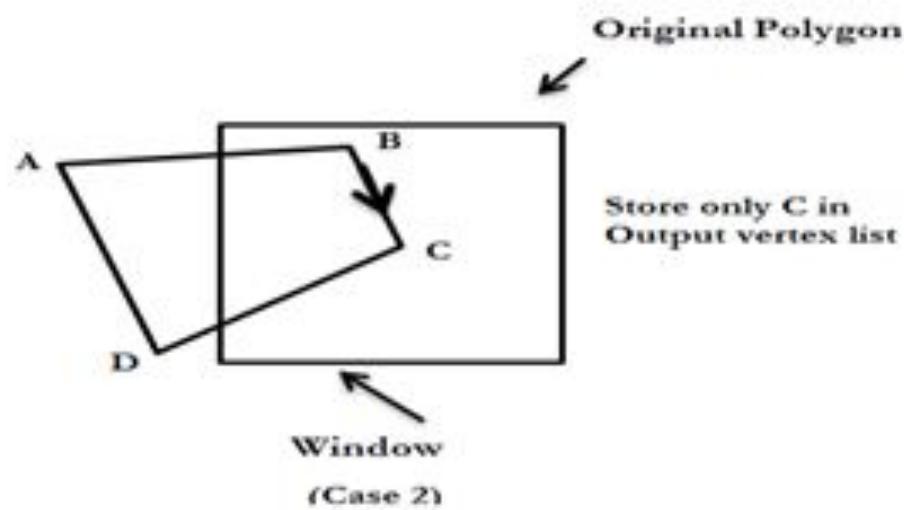


Outside to outside:
Output: nothing

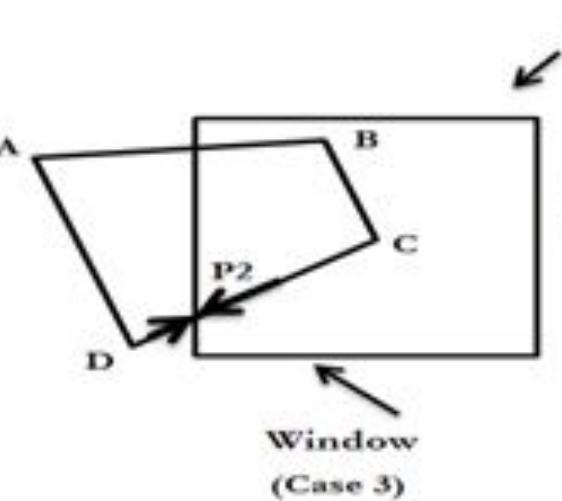
Sutherland-Hodgman Polygon Clipping



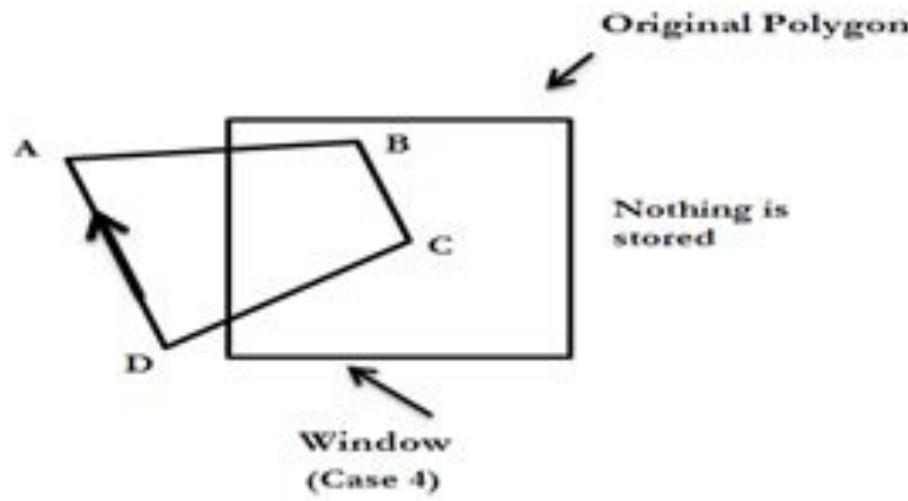
Store P1 & B in Output vertex list



Store only C in Output vertex list

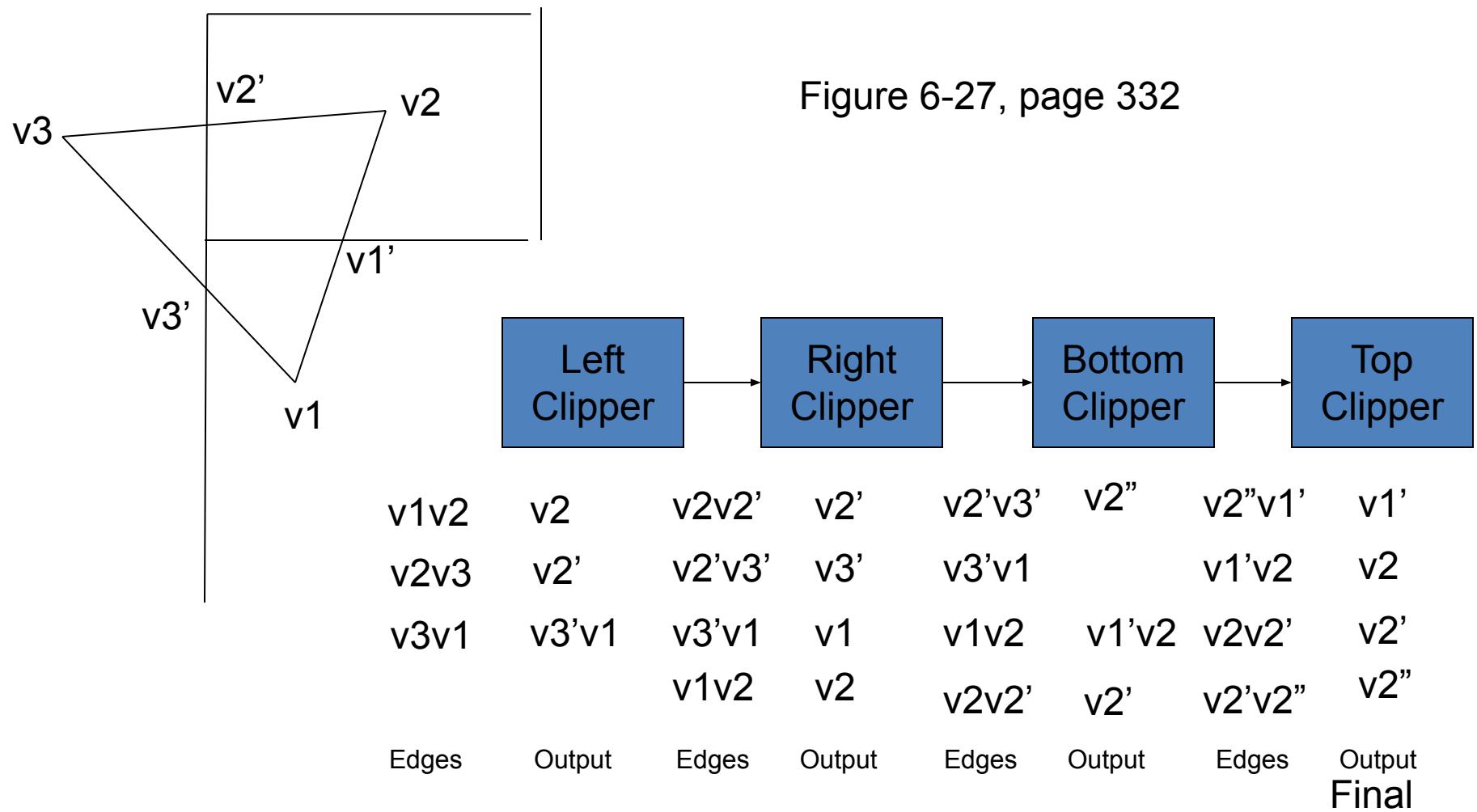


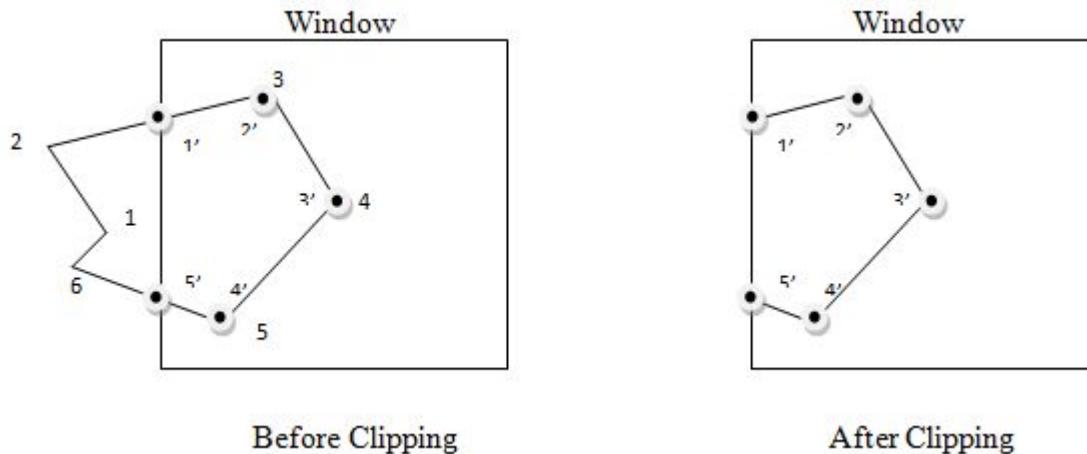
Store P2 only in Output vertex list



Nothing is stored

Sutherland-Hodgman Polygon Clipping





Steps

Process area in figure against left window boundary .

- Vertices 1,2 are outside of boundary
- Moving along to vertex 3 which is inside calculate the intersection . Save both intersection point and vertex 3
- Vertices 4 and 5 are determined to be inside save them both
- Vertex 6 is outside so find intersection point so save the intersection point.

Required setting up storage for an output list vertices as a polygon is clipped against each window boundary

The final set of vertices of the clipped polygon are 1' 2'=3 3'=4 4'=5 5'

Filled Area Primitives

Standard output primitive in graphics packages is solid color ,patterned polygon area

Polygons are easier to process due to linear boundaries

Two basic approaches to area filling on a raster system

- 1. Determine the overlap intervals for scan lines that cross the area.
Typically useful for filling polygons, circles, ellipses**

- 2. Start from a given interior position and paint outwards from
this point until we encounter the specified boundary conditions
useful for filling more complex boundaries, interactive painting
system.**

Filling rectangles

Two things to consider

- i. which pixels **to fill**
- ii. **with what value to fill**

Move along scan line (from left to right) that intersect the primitive and fill in pixels that lay inside

To fill rectangle with solid color

Set each pixel lying on scan line running from left edge to right with same pixel value, each span from x_{max} to x_{min}

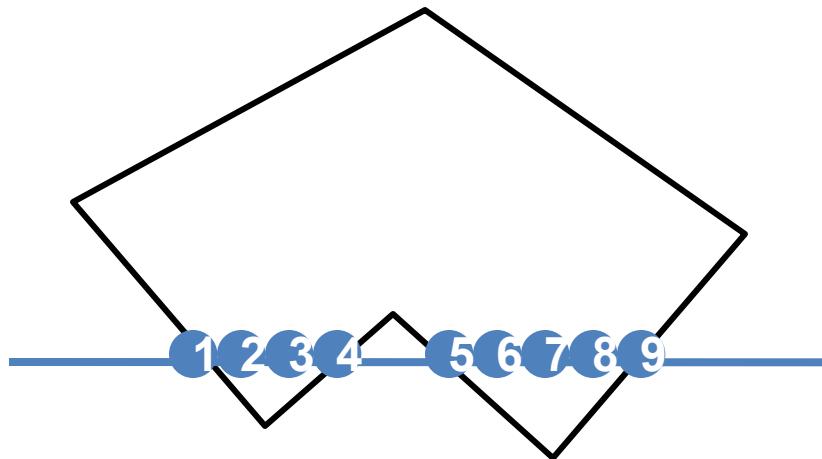
```
for( y from  $y_{min}$  to  $y_{max}$  of rectangle)           /*scan line*/
    for( x from  $x_{min}$  to  $x_{max}$  of rectangle)       /*by pixel*/
        writePixel(x, y, value);
```

Filling Polygons

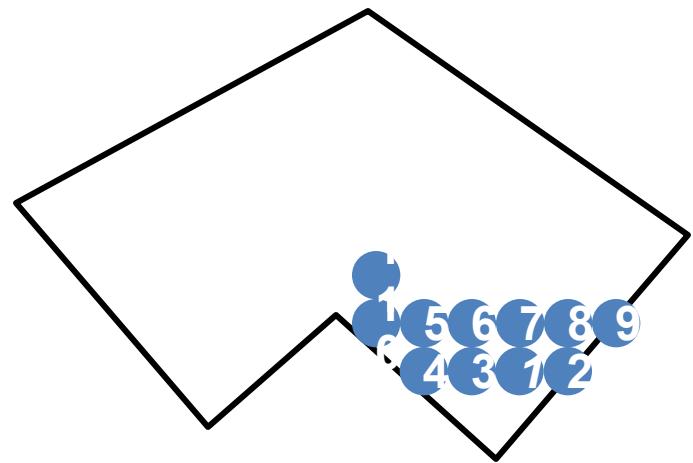
Filling Polygons

Scan-line fill algorithm

Inside-Outside tests



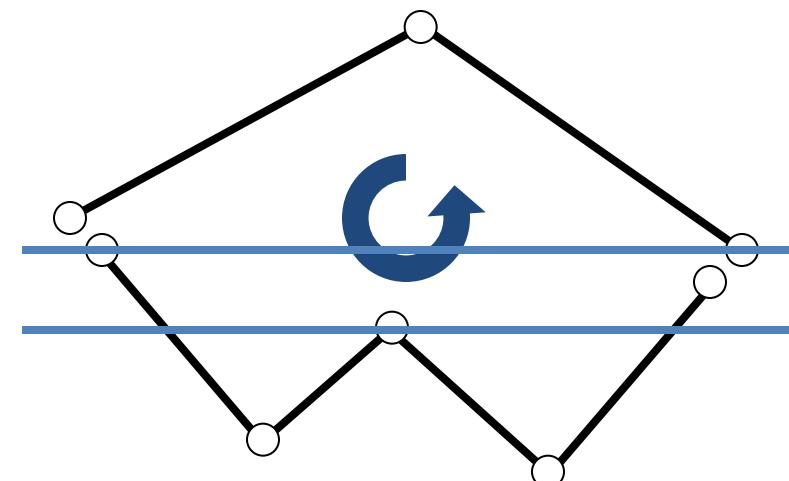
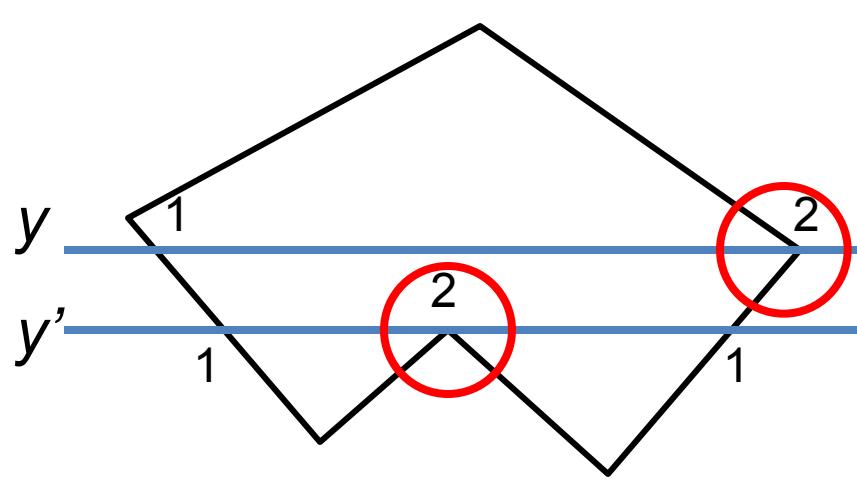
Boundary fill algorithm



Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

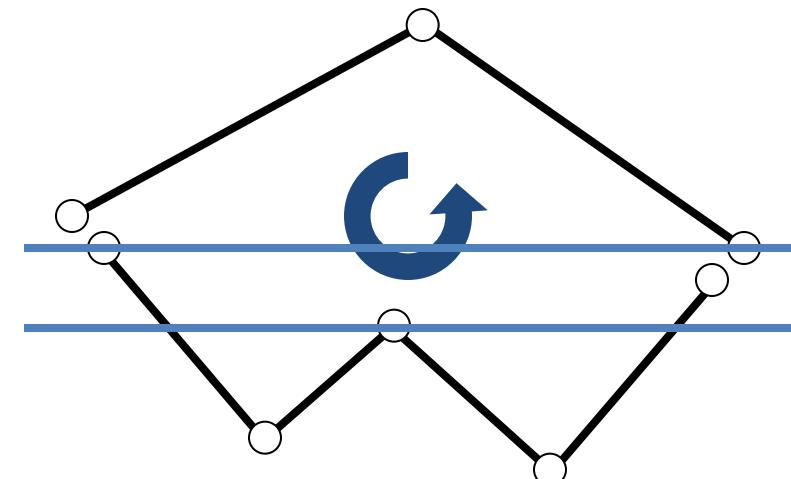
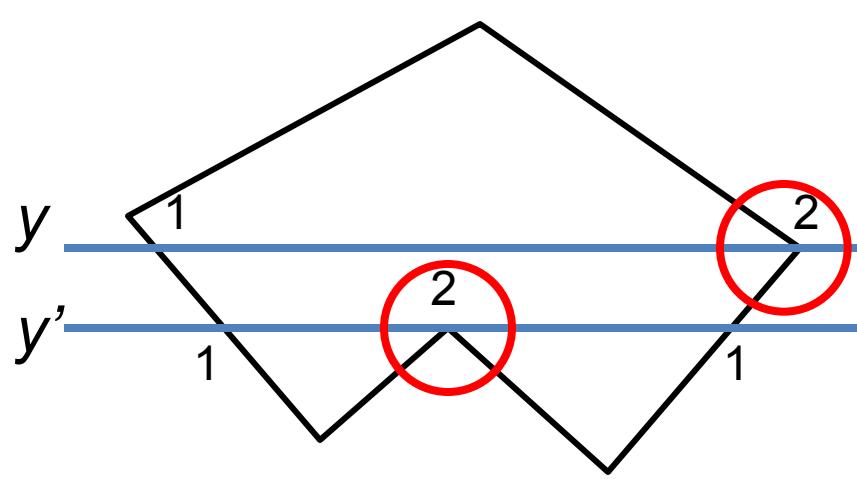
y' : intersection edges are same side



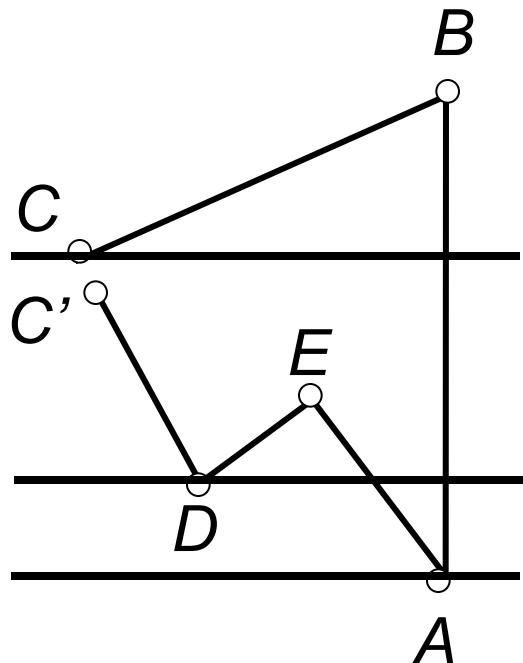
Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

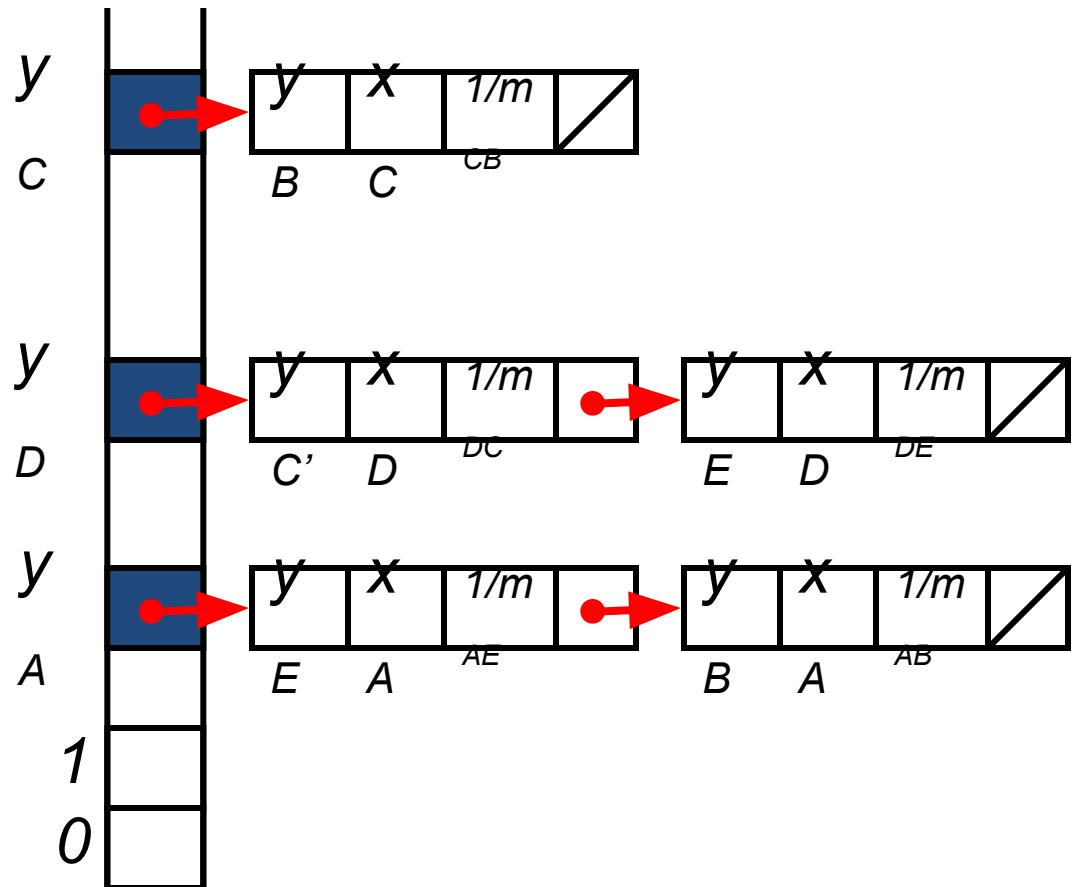
y' : intersection edges are same side



Edge Sorted Table



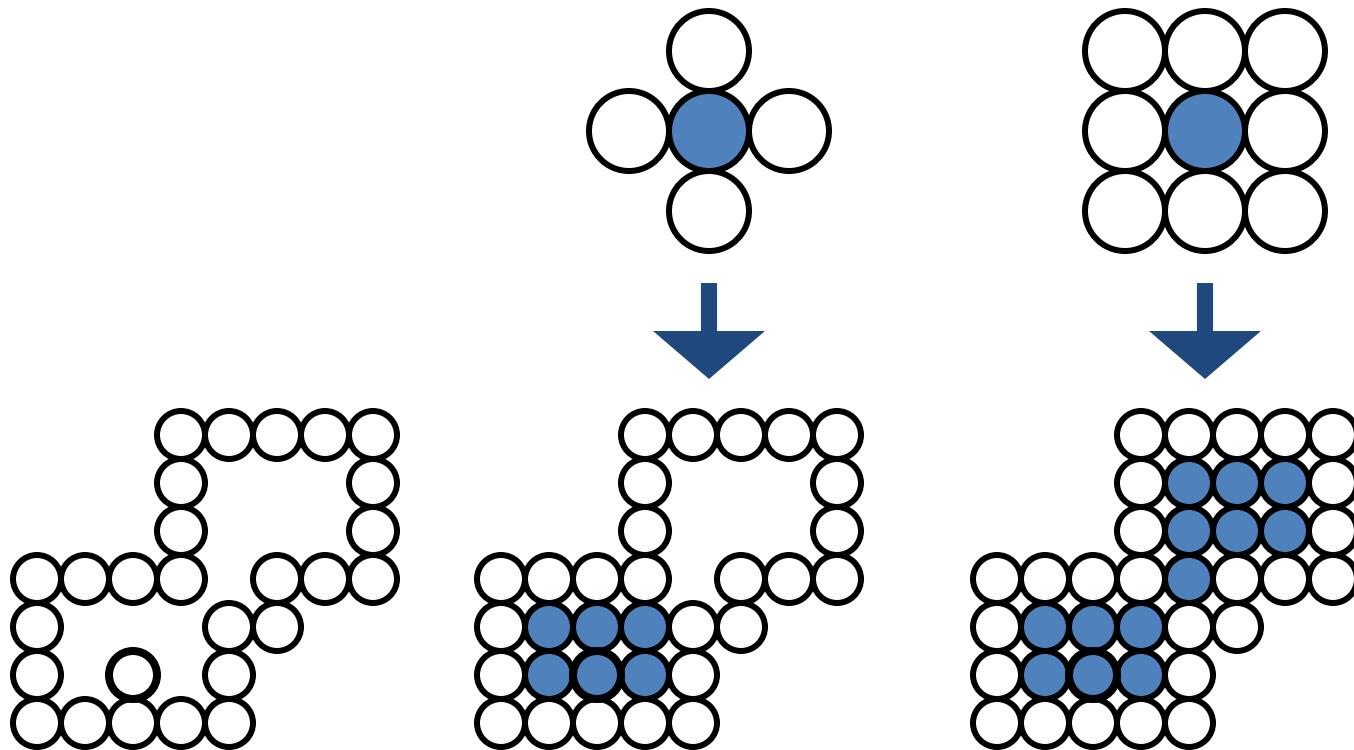
*Scan-Line
Number*



Proceed to Neighboring Pixels

4-Connected

8-Connected



Boundary Fill

Start at a point inside a region and paint the interior outward toward the boundary.

If boundary is specified in a single color the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered

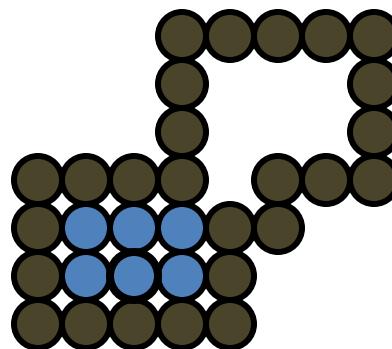
Accepts as input coordinates of an interior point (x, y) a fill color and boundary color.

Starting from (x, y) the procedure tests neighboring position to determine whether they are of the boundary color.

If not they are painted with fill color and their neighbors are tested

Process continues until all pixels up to boundary color for the area have been tested

```
boundaryFill(int x,y,fill_color, boundary_color){
    int color;
    getpixel(x,y,color)
    if(color != boundary_color AND color != fill_color ){
        setpixel( x,y,fill_color)
        boundaryFill( x+1,y,fill_color, boundary_color)
        boundaryFill( x,y+1,fill_color, boundary_color)
        boundaryFill( x-1,y,fill_color, boundary_color)
        boundaryFill( x,y-1,fill_color, boundary_color)
    }
}
```



Flood Fill

Filling an area that is not defined within a single color boundary.

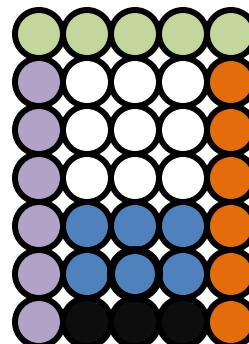
In this case replace a specified interior color instead of searching for boundary color value.

Start from specified interior point (x , y) and reassign all pixel values that are currently set to a given interior color with the desired color.

If the area we want to paint has more than 1 interior color , first assign pixel values so that all interior points have same color.

We can then use 8 or 4 connected approach to move on until all interior points have been repainted.

```
floodFill(int x,y,fill_color, original_color){
    int color;
    getpixel(x,y,color)
    if(color == original_color ){
        setpixel(x,y,fill_color)
        floodFill(x+1,y,fill_color, original_color)
        floodFill(x,y+1,fill_color, original_color)
        floodFill(x-1,y,fill_color, original_color)
        floodFill(x,y-1,fill_color, original_color)
    }
}
```



2D, 3D Clipping

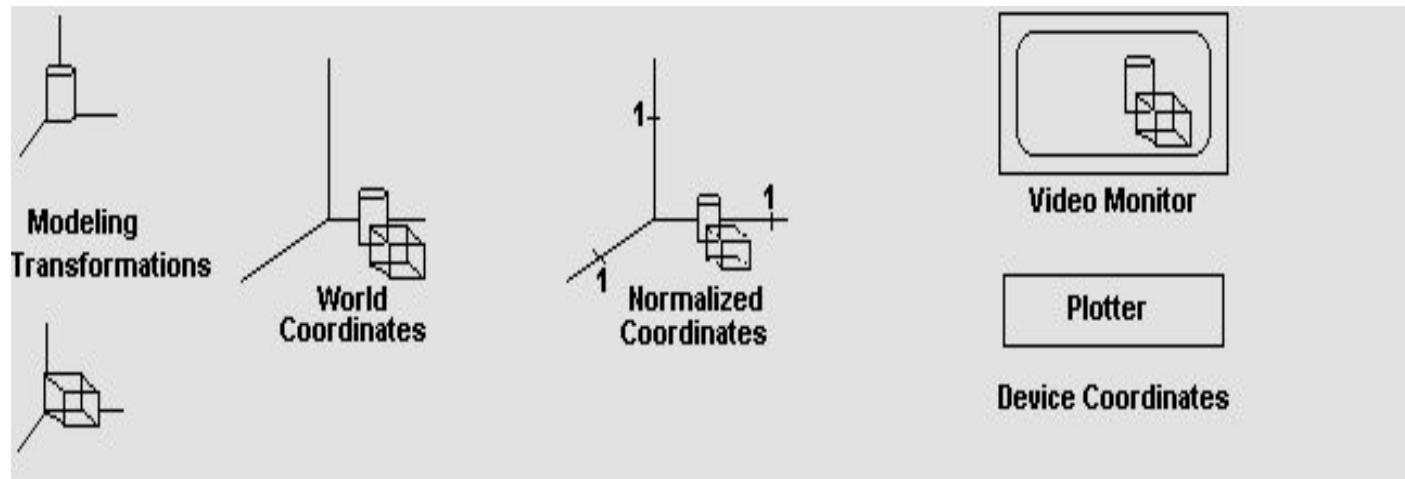
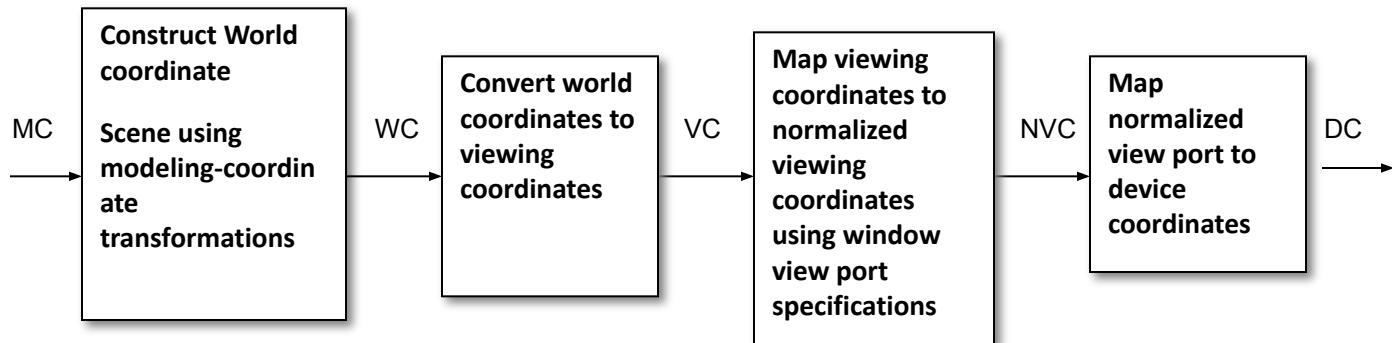
Window and View ports

- A rectangular area specified in **world coordinates** is called a window .
- A rectangular area on the **display device** to which a window is mapped is called a view port .
- The window defines what **is to be viewed**;
the view port defines where **it is to be displayed**.

Window and View ports

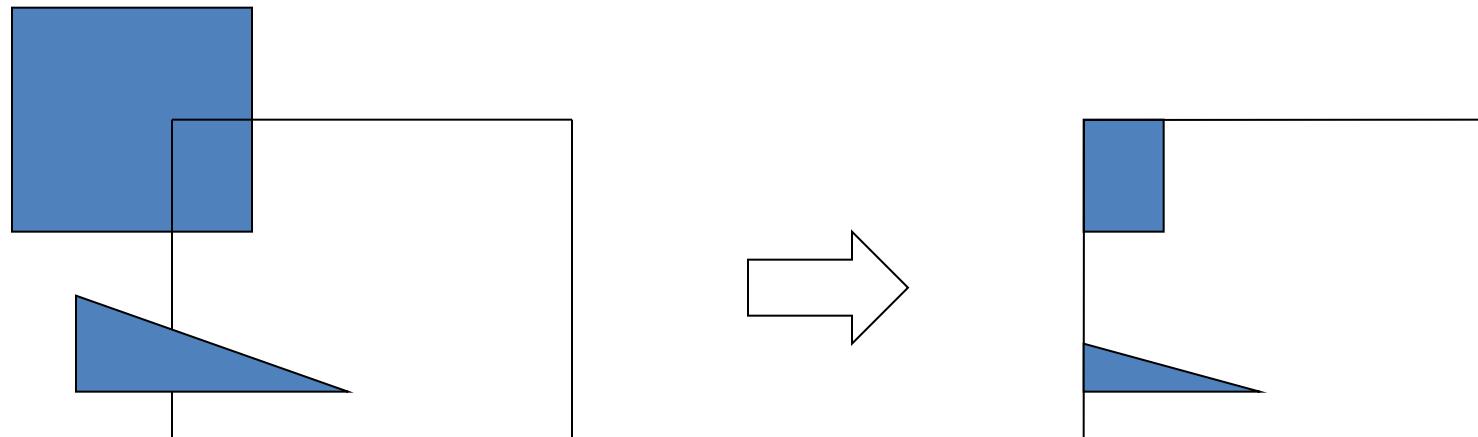
- Often windows and view ports are rectangles in standard position with rectangle edges parallel to coordinate axes
- The **mapping of a part of world coordinate scene to device coordinate** is referred to as viewing transformation.

2D Viewing Pipeline



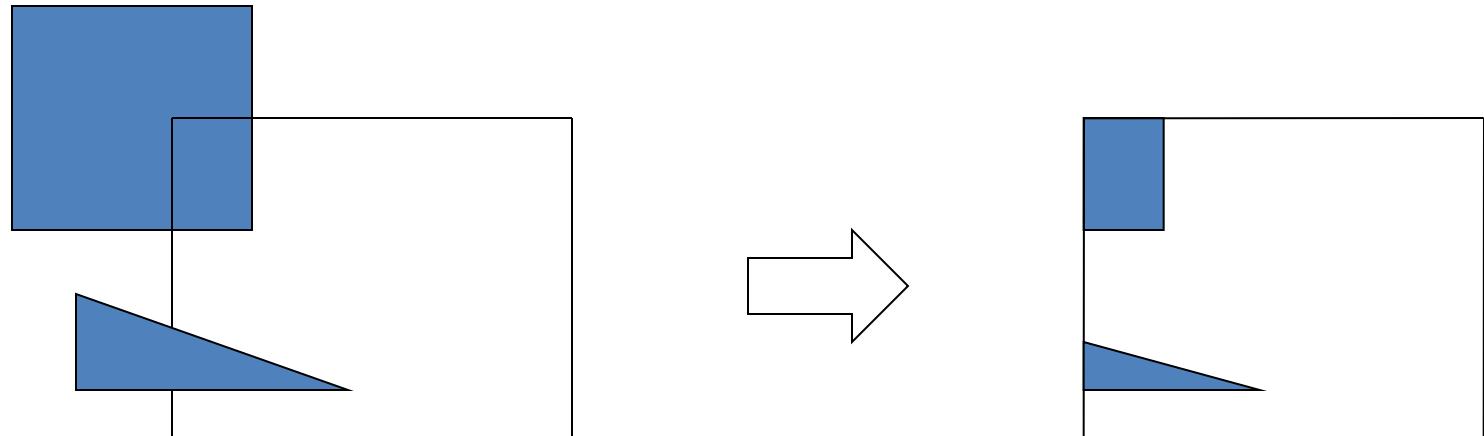
Clipping in Raster World

- Procedure that identifies those operations of picture that are either inside or outside
- The region against which an object is to be clipped is called a clip window.
- Depending on application it can be polygons or even curve surfaces.



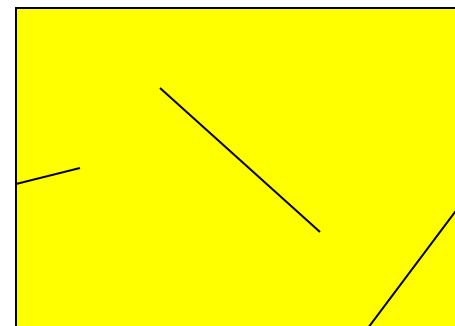
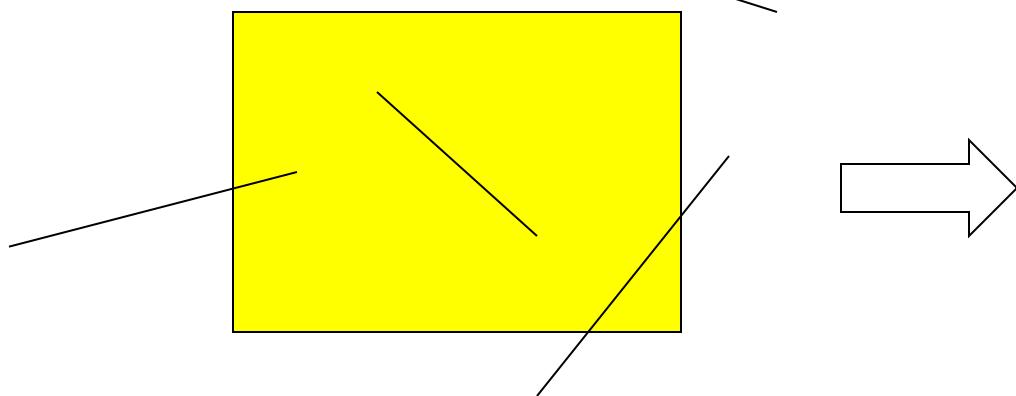
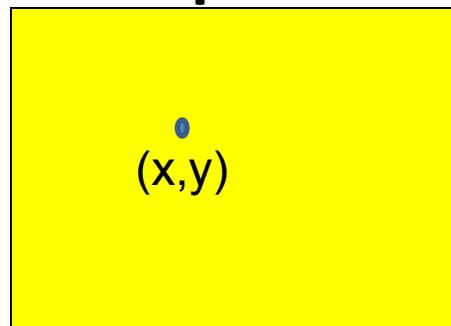
Clipping in Raster World

- **Applications**
 - i. Extracting parts **of defined scene for viewing**
 - ii. Identifying visible surfaces **in three dimension views**
- **iii. Drawing, painting operations that allow parts of picture to be selected for copying, moving, erasing or duplicating etc.**



Line Clipping

- Point clipping easy: Just **check the inequalities** (x_{\max}, y_{\max})
 - $x_{\min} < x < x_{\max}$
 - $y_{\min} < y < y_{\max}$
- Line clipping more tricky (x_{\min}, y_{\min})

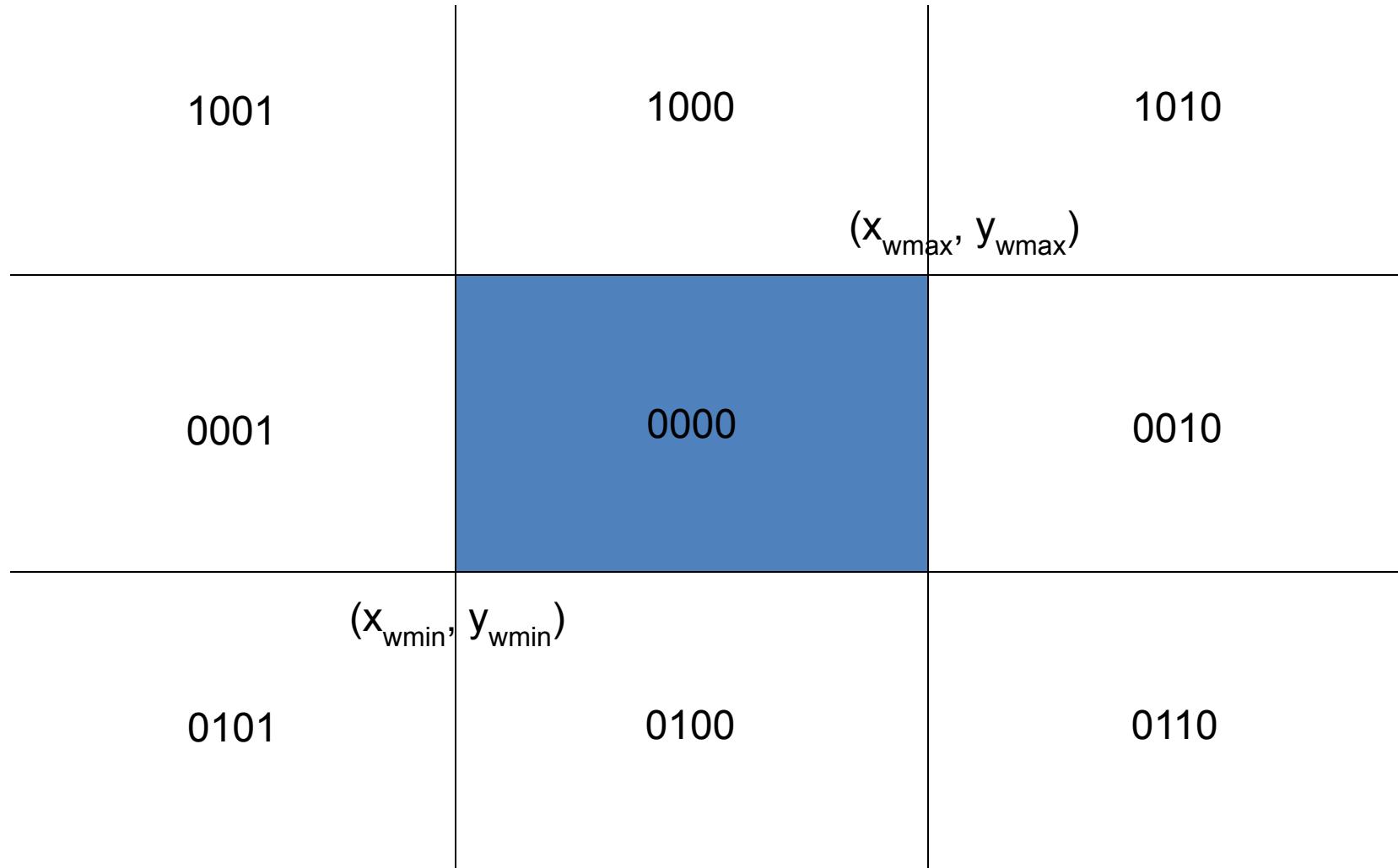


Cohen-Sutherland Line Clipping

- Divide 2D space into 3×3 regions.
- Middle region is the **clipping window**.
- Each region is assigned a 4-bit code.
- Bit 1 is set to 1 if the region is to the **left** of the clipping window, otherwise. Similarly for bits 2, 3 and 4.



Cohen-Sutherland Line Clipping



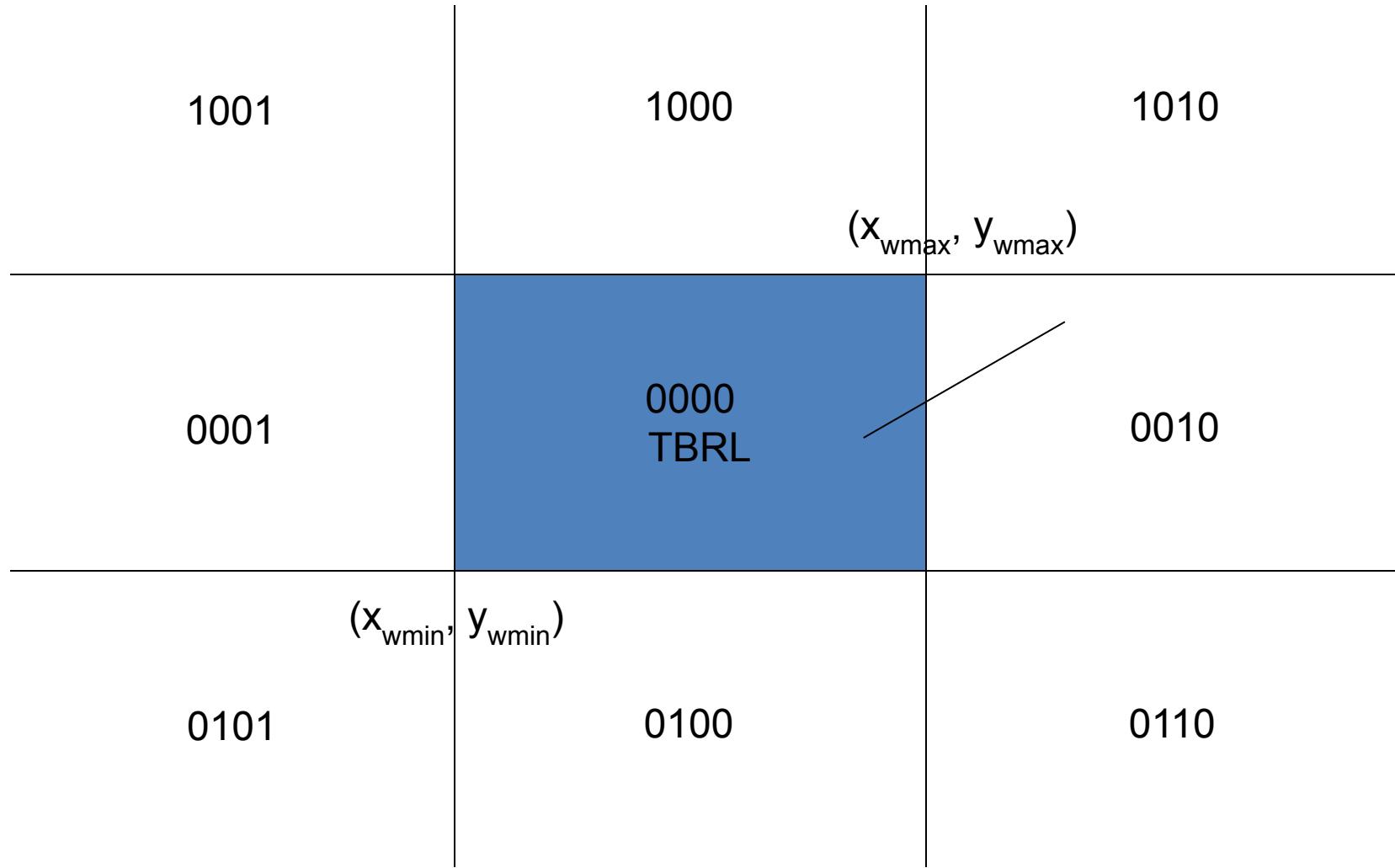
Cohen-Sutherland Line Clipping

- To clip a line, find out which regions its two endpoints lie in.

Case I

- If they are **both in** region 0000, then it's completely in.

Cohen-Sutherland Line Clipping

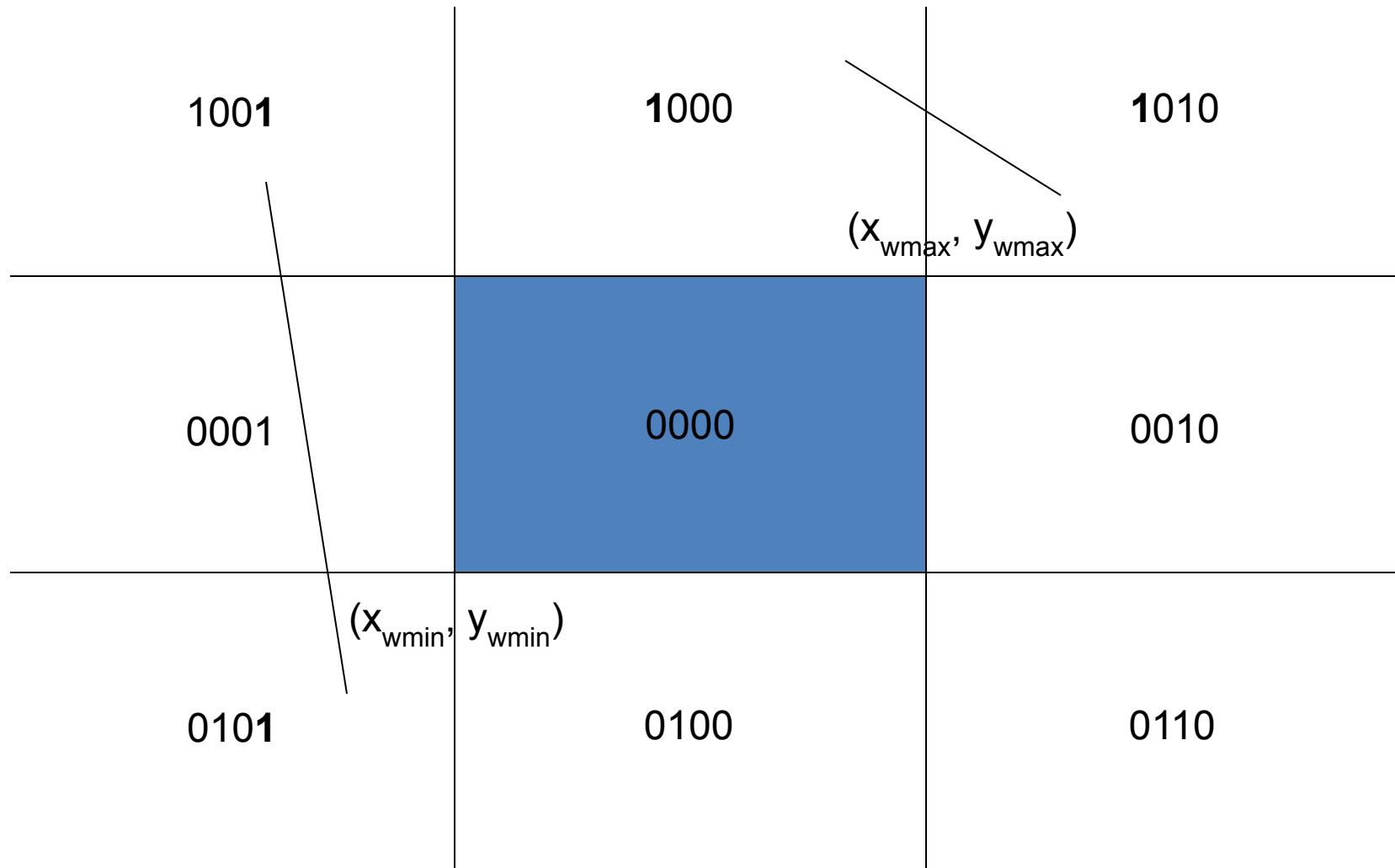


Cohen-Sutherland Line Clipping

Case II

- If the two region numbers both have a 1 in the same bit position, the line is **completely out**.

Cohen-Sutherland Line Clipping

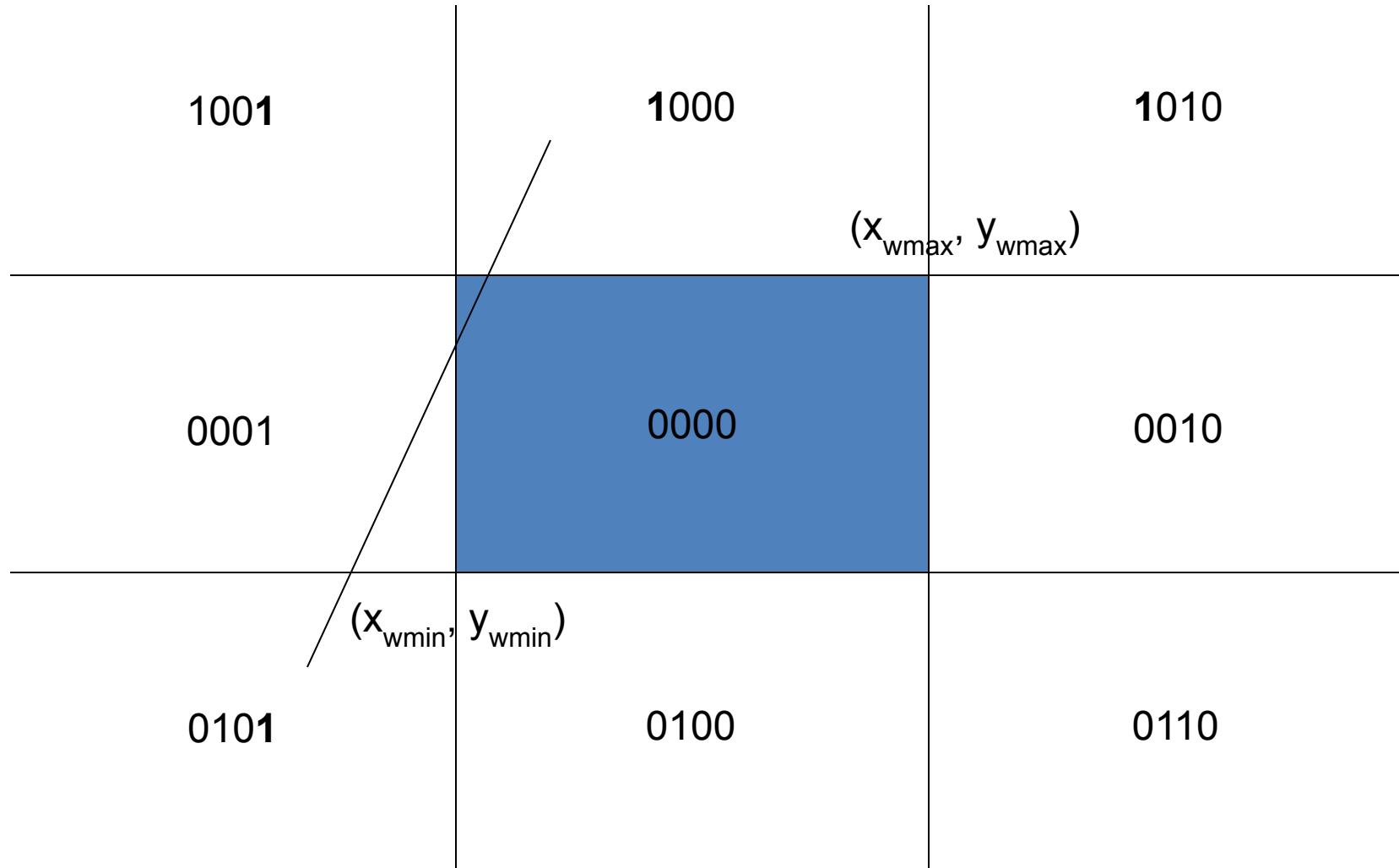


Cohen-Sutherland Line Clipping

Case III

- If lines can not be identified as completely inside or outside we have to **do some more calculations.**
- Here we find the intersection points with a clipping boundary using the slope intercept form of the line equation

Cohen-Sutherland Line Clipping



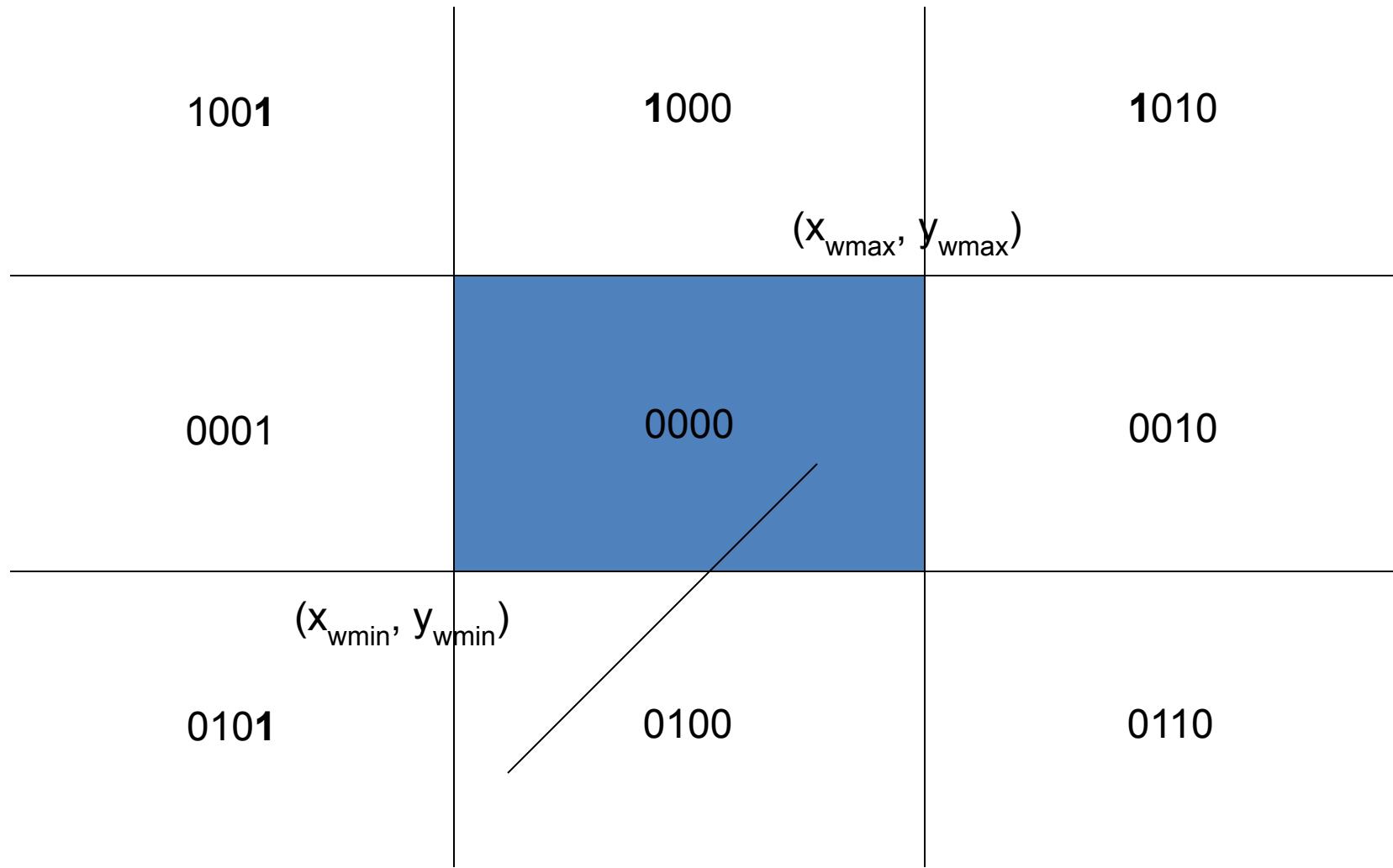
Cohen-Sutherland Line Clipping

- For a line with end point coordinates (x_1, y_1) and (x_2, y_2) the y coordinate of the intersection point with a vertical boundary is

$$y = y_1 + m (x - x_1)$$

- Where x is set to either x_{wmin} or x_{wmax}

Cohen-Sutherland Line Clipping



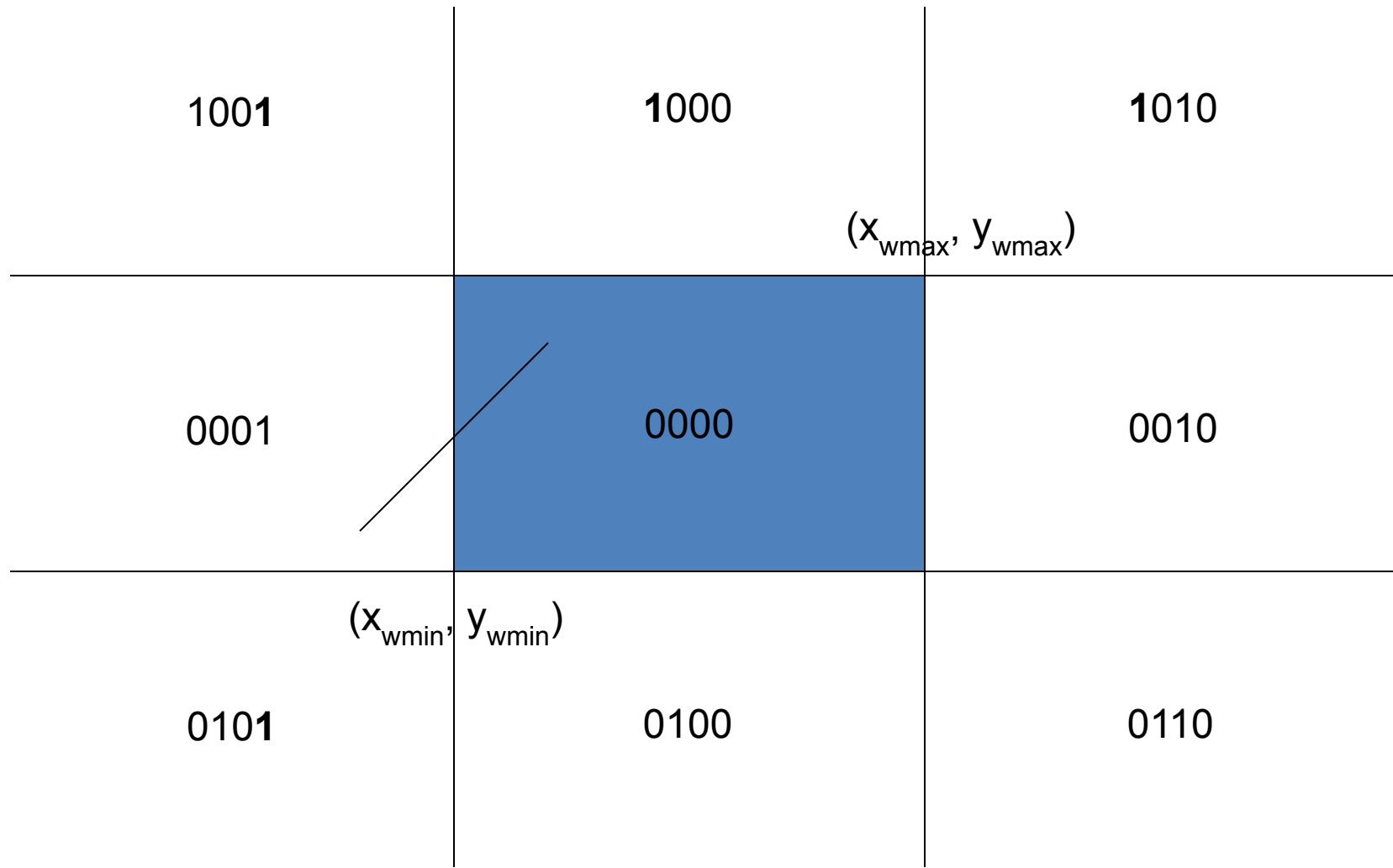
Cohen-Sutherland Line Clipping

- Similarly for the intersection with a vertical boundary

$$x = x_1 + (y - y_1)/m$$

- Where y is set to either $y_{w_{\min}}$ or $y_{w_{\max}}$

Cohen-Sutherland Line Clipping



Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(70,90) ,B(60,85) against a clip window with its lower left corner at (50,80) and upper right corner at (90,120)

Hints:

- i. Find the region code for each end point of the line
- ii. Find if it is above, below, left or right of the clip window
- iii. Determine the intersection point of the line segment with the boundary

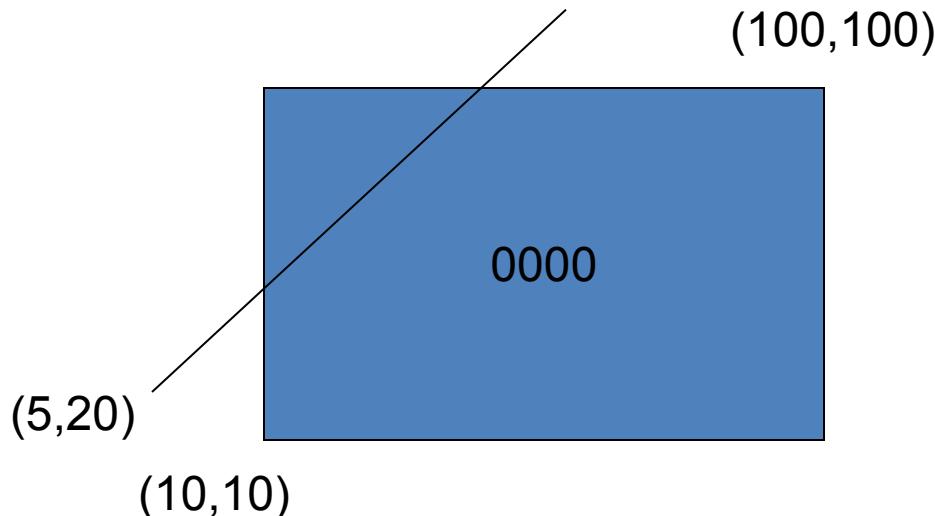
Cohen-Sutherland Line Clipping

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(5,20) ,B(80,120) against a clip window with its lower left corner at (10,10) and upper right corner at (100,100)

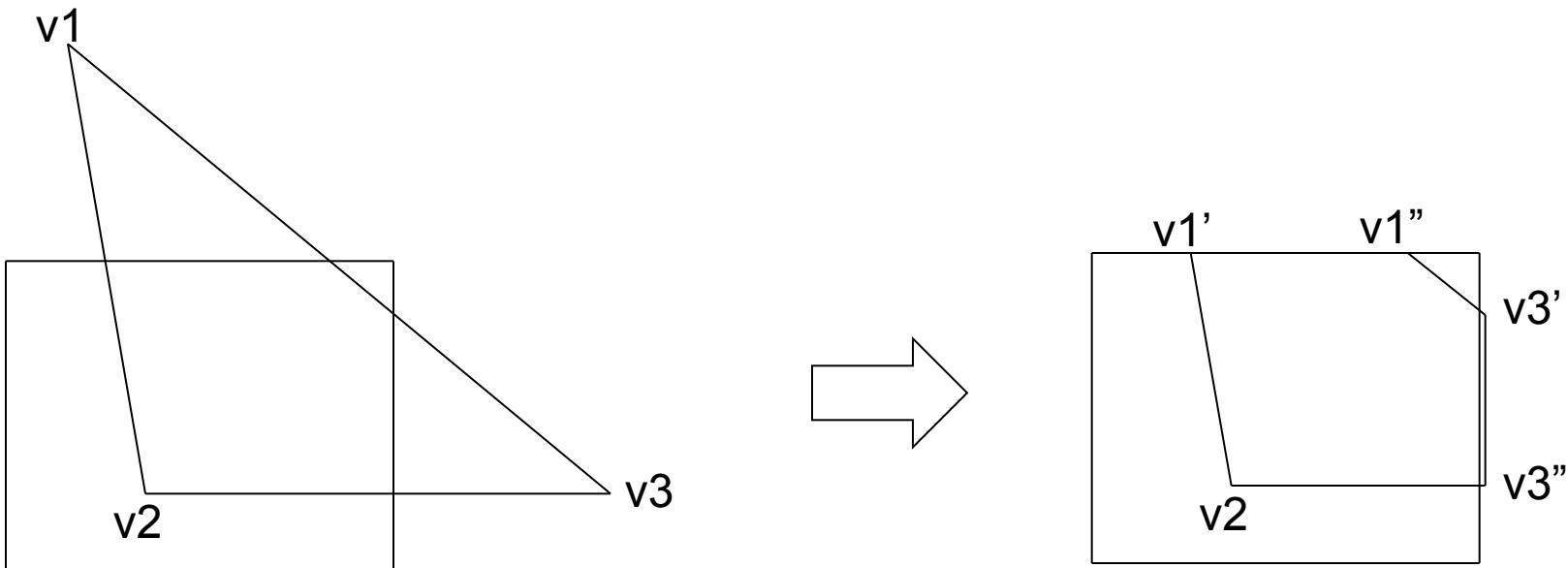
Hints:

- i. Find the region code for each end point of the line
- ii. Find if it is above, below, left or right of the clip window
- iii. Determine the intersection point of the line segment with the boundary

$$x = x_1 + (y - y_1)/m \quad (80, 120)$$



Polygon Fill-Area Clipping

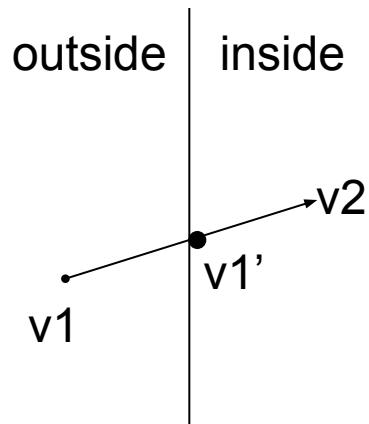


Note: Need to consider each of 4 edge boundaries

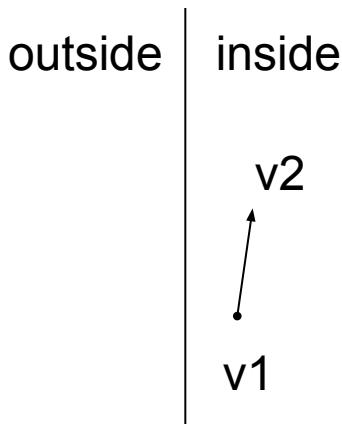
Sutherland-Hodgman Polygon Clipping

- Input each edge (vertex pair) successively.
- Output is a new list of vertices.
- Each edge goes through 4 clippers.
- The rule for each edge for each clipper is:
 - If first input vertex is outside, and second is inside, output the intersection and the second vertex
 - If first both input vertices are inside, then just output second vertex
 - If first input vertex is inside, and second is outside, output is the intersection
 - If both vertices are outside, output is nothing

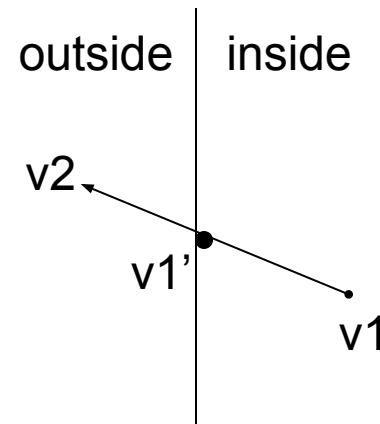
Sutherland-Hodgman Polygon Clipping: Four possible scenarios at each clipper



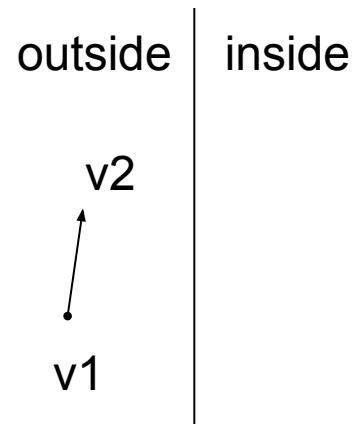
Outside to inside:
Output: v_1' and v_2



Inside to inside:
Output: v_2

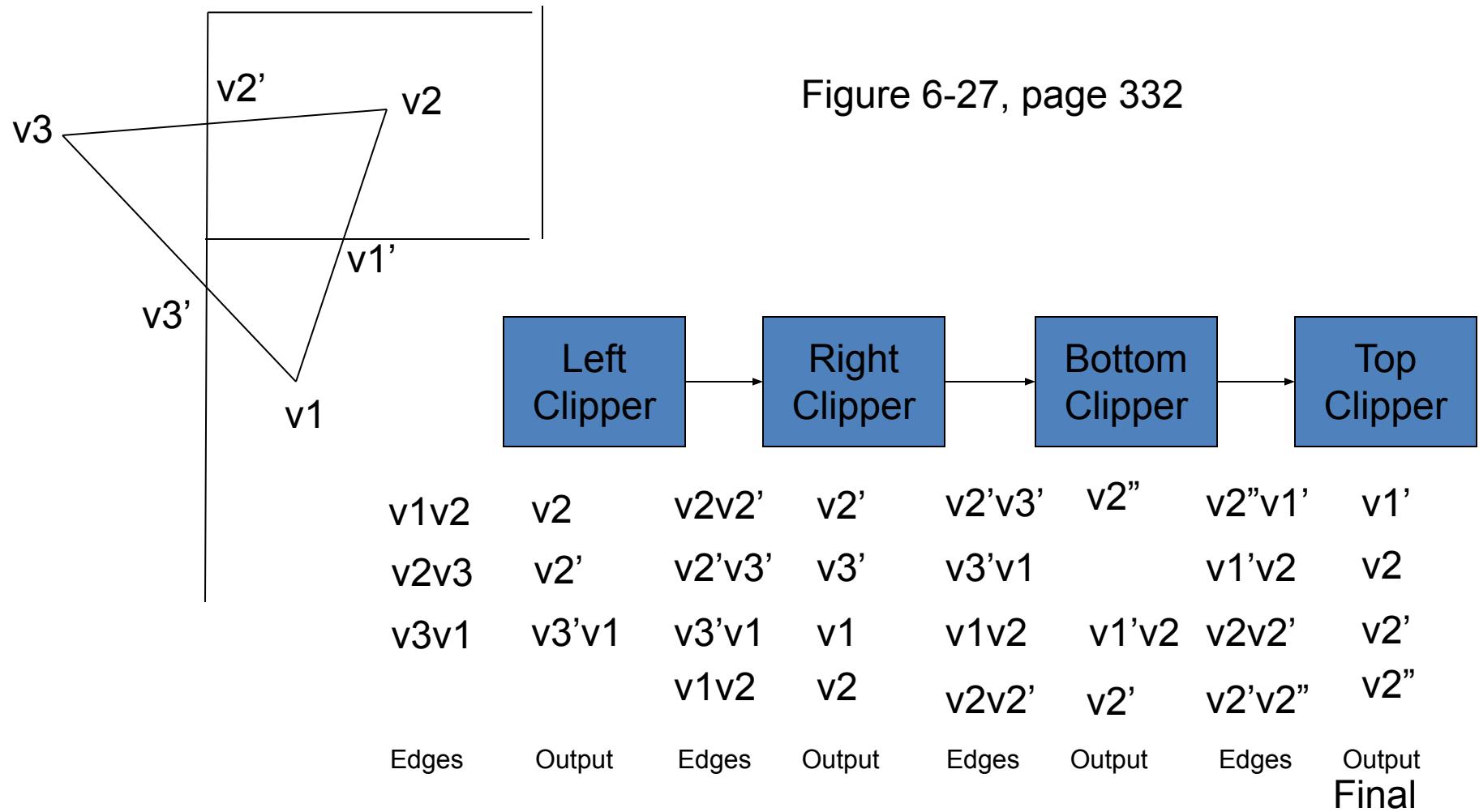


Inside to outside:
Output: v_1'

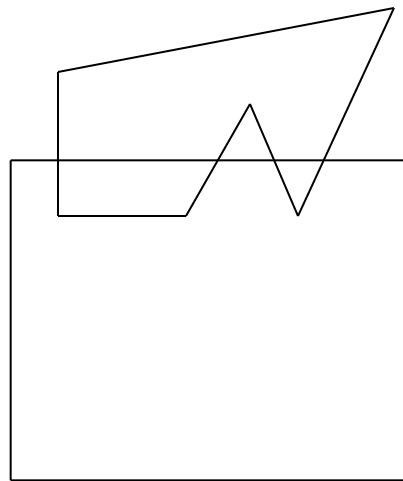


Outside to outside:
Output: nothing

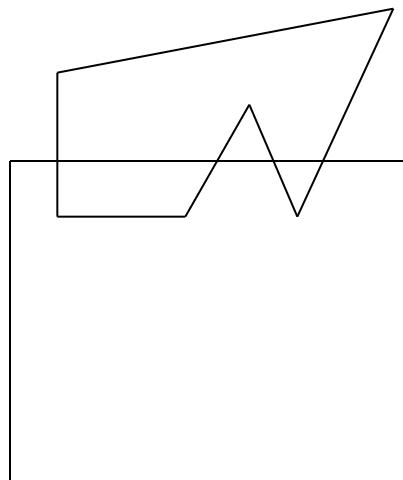
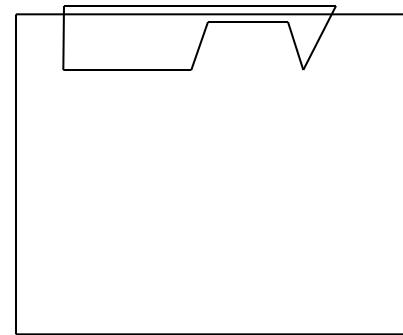
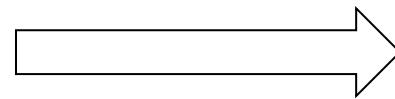
Sutherland-Hodgman Polygon Clipping



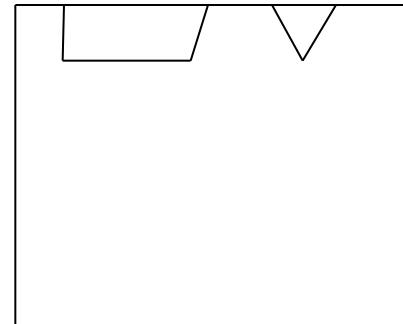
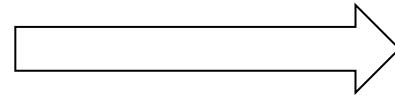
Weiler-Atherton Polygon Clipping



Sutherland-Hodgman



Weiler-Atherton



Clipping in 3D

- Suppose the view volume has been normalized. Then the clipping boundaries are just:

$$xw_{\min} = -1$$

$$xw_{\max} = 1$$

$$yw_{\min} = -1$$

$$yw_{\max} = 1$$

$$zw_{\min} = -1$$

$$zw_{\max} = 1$$

Clipping Homogeneous Coordinates in 3D

- Coordinates expressed in homogeneous coordinates
- After geometric, viewing and projection transformations, each vertex is: (x_h, y_h, z_h, h)
- Therefore, assuming coordinates have been normalized to a $(-1,1)$ volume, a point (x_h, y_h, z_h, h) is inside the view volume if:

$$-1 < \frac{x_h}{h} < 1 \quad \text{and} \quad -1 < \frac{y_h}{h} < 1 \quad \text{and} \quad -1 < \frac{z_h}{h} < 1$$

Suppose that $h > 0$, which is true in normal cases, then

$$-h < x_h < h \quad \text{and} \quad -h < y_h < h \quad \text{and} \quad -h < z_h < h$$

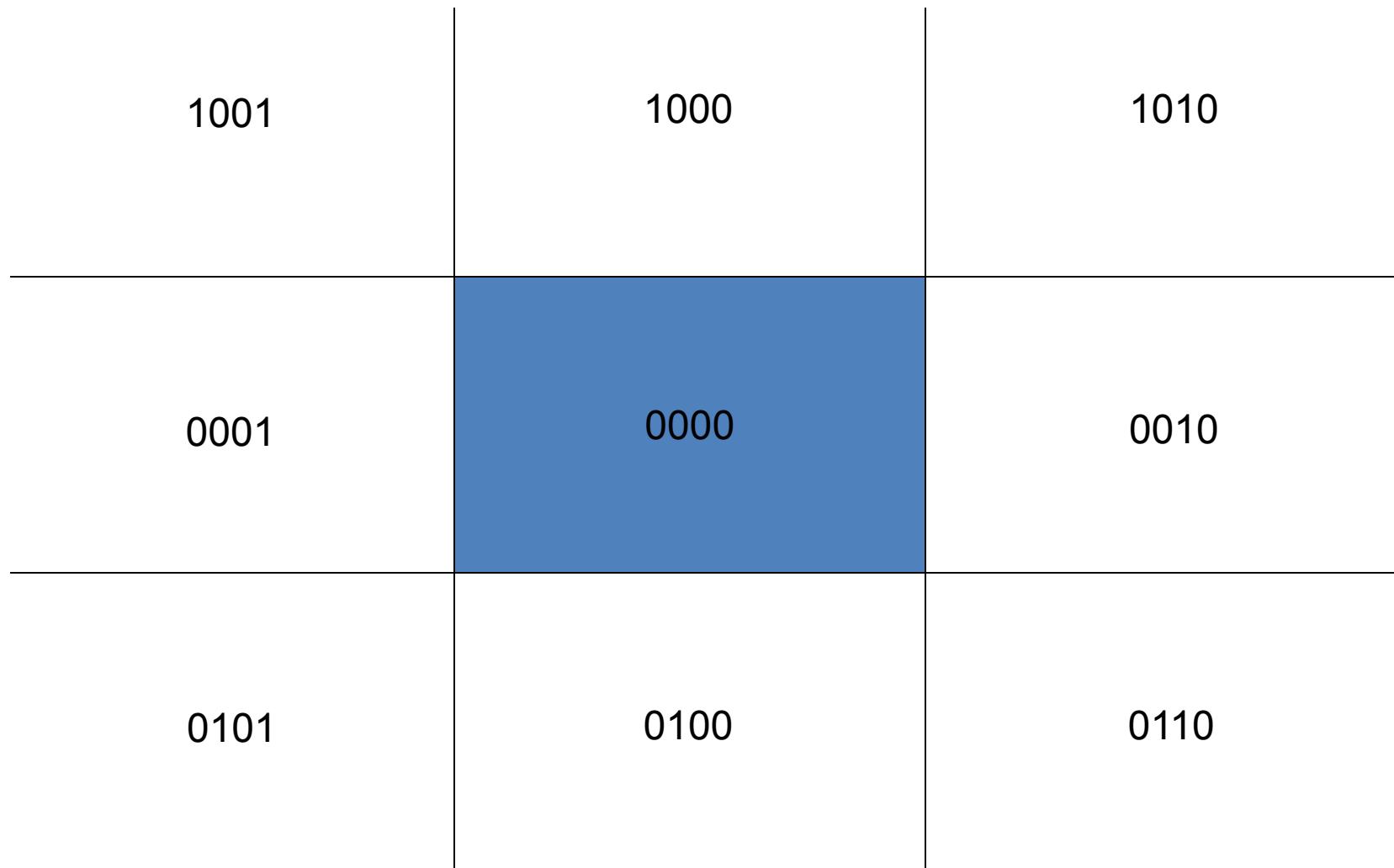
Remember Cohen-Sutherland 2D Line Clipping Region Codes?

- Divide 2D space into 3x3 regions.
- Middle region is the clipping window.
- Each region is assigned a 4-bit code.
- Bit 1 is set to 1 if the region is to the left of the clipping window, 0 otherwise. Similarly for bits 2, 3 and 4.



Cohen-Sutherland Line Clipping Region

Codes in 2D



3D Cohen-Sutherland Region Codes

- Simply use 6 bits instead of 4.

6	5	4	3	2	1
Far	Near	Top	Bottom	Right	Left

Example: If $h + x_h < 0$, then bit 1 is set to 1.

This is because if $-h > x_h$, then the point is to the left of the viewing volume.

Clipping Polygons

- First, perform trivial acceptance and rejection using, for example, its coordinate extents.
- Polygons usually split into triangle strips.
- Then each triangle is clipped using 3D extension of the Sutherland-Hodgman method

Arbitrary Clipping Planes

- Can specify an arbitrary clipping plane: $Ax + By + Cz + D = 0$.
- Therefore, for any point, if $Ax + By + Cz + D < 0$, it is not shown.
- To clip a line against an arbitrary plane,
 - If both end-points are in, then the line is in
 - If both end-points are out, then the line is out
 - If one end-point is in, and one is out, then we need to find the intersection of the line and the plane

Intersection of Line and Plane

- First, given two end-points of a line, P_1 and P_2 , form a parametric representation of the line:

$$P = P_1 + (P_2 - P_1) u, \text{ where } 0 < u < 1$$

Equation of the clipping plane: $N \cdot P + D = 0$, where $N = (A, B, C)$

Substituting, $N \cdot (P_1 + (P_2 - P_1)u) + D = 0$

$$u = \frac{-D - N \cdot P_1}{N \cdot (P_2 - P_1)}$$

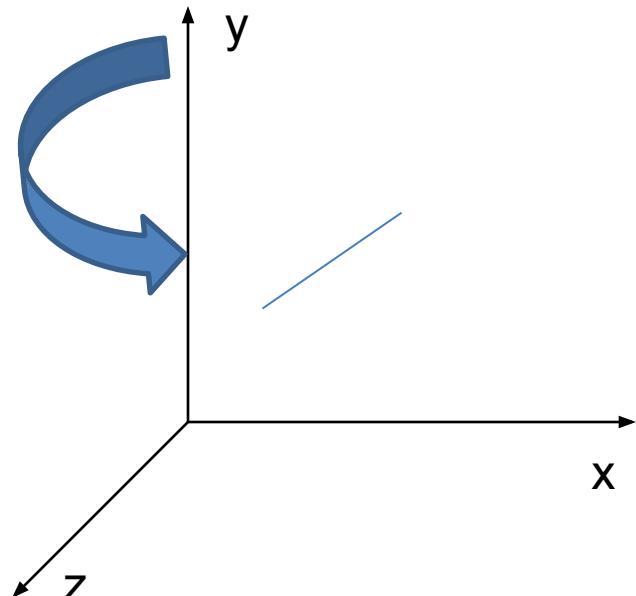
Rotation about z axis ccw

$$z' = z \cos\theta - x \sin\theta$$

$$x' = z \sin\theta + x \cos\theta$$

$$y' = y$$

$x \rightarrow y \rightarrow z \rightarrow x$

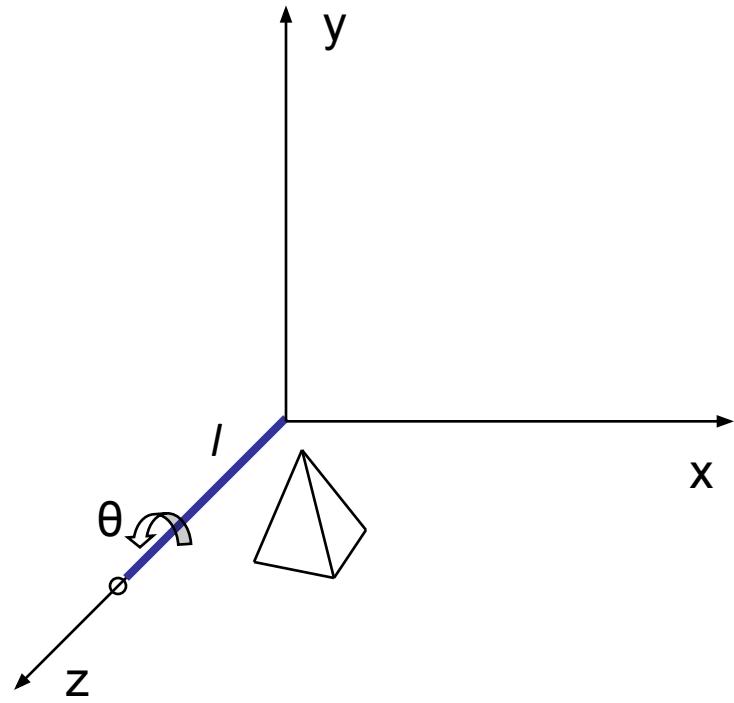
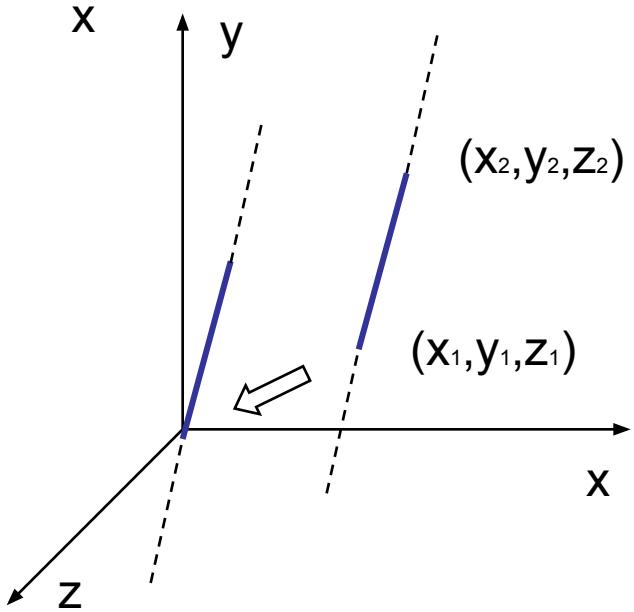
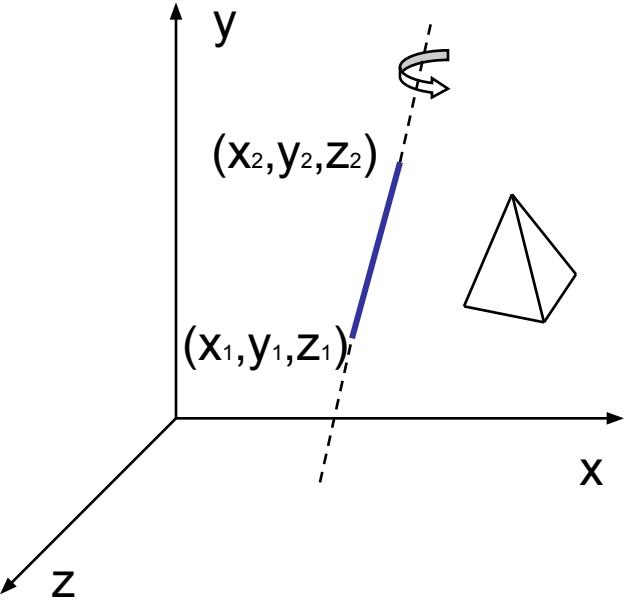


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

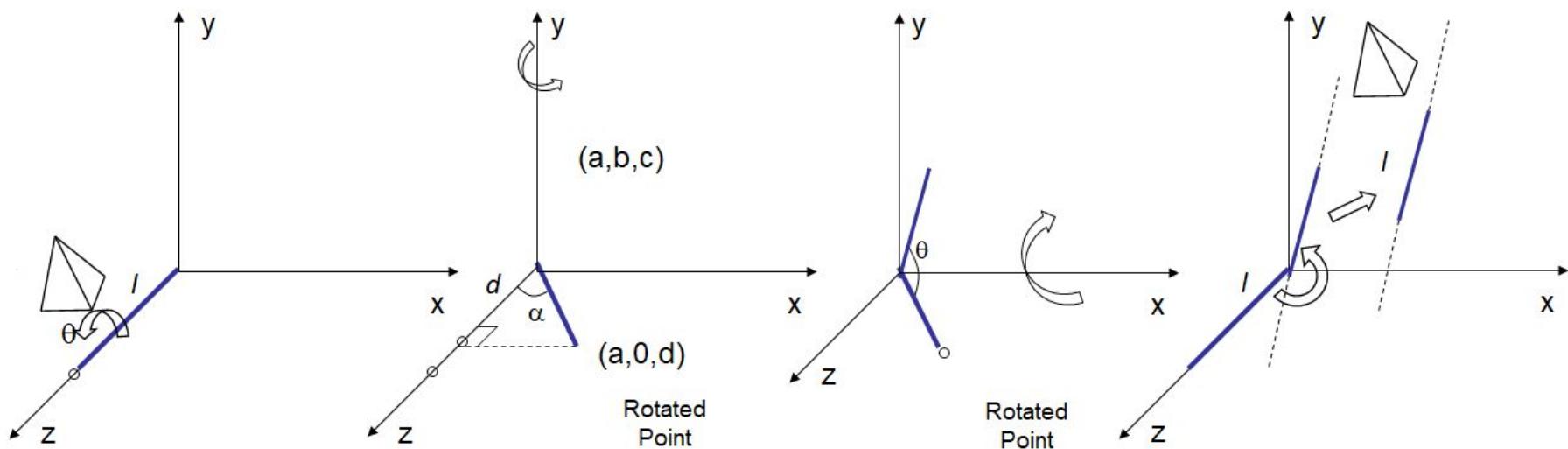
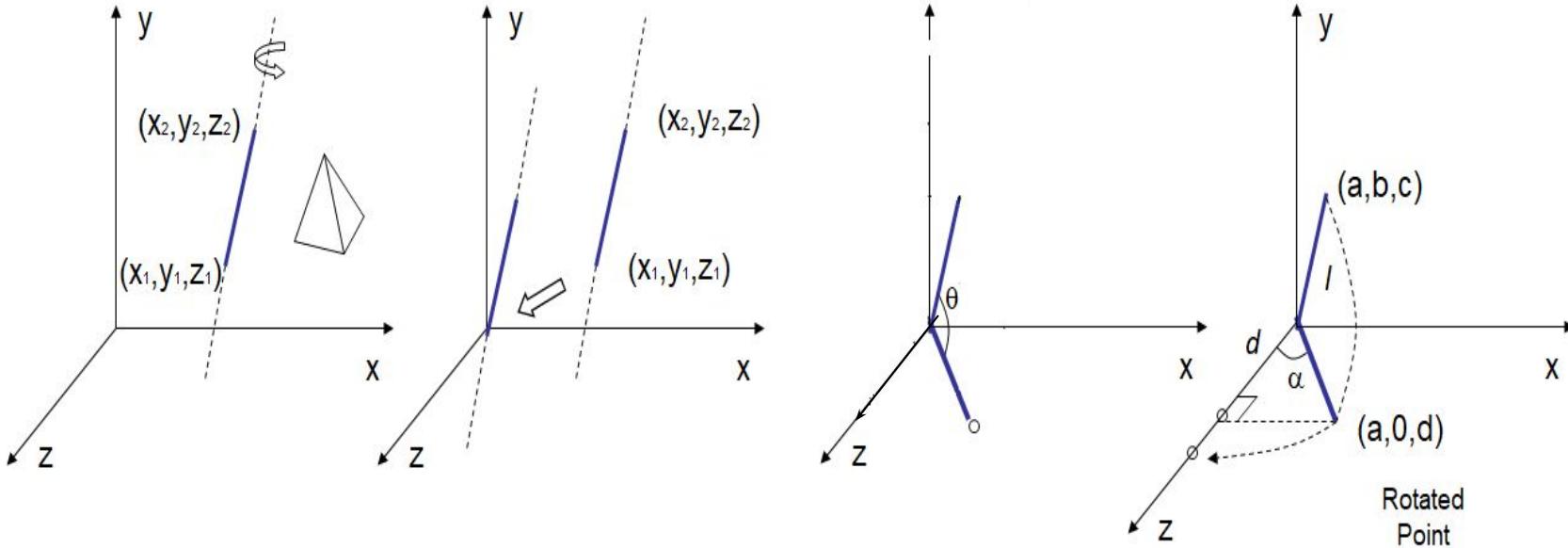
Rotation about an Arbitrary Axis

Steps

1. Translate (x_1, y_1, z_1) to the origin
2. Rotate (x'_2, y'_2, z'_2) on to the z axis
3. Rotate the object around the z-axis
4. Rotate the axis to the original orientation
5. Translate the rotation axis to the original position



Rotation about an Arbitrary Axis

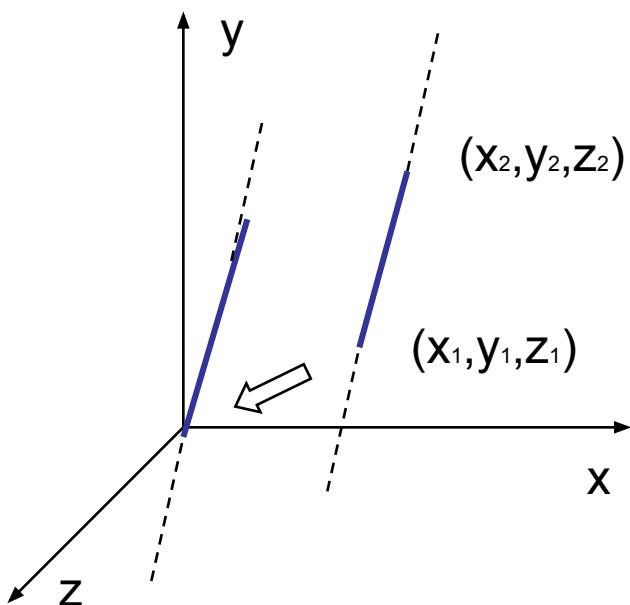


$$cm = T'_{(x_1, y_1, z_1)} \cdot R'_x \cdot R'_y \cdot R_z \cdot R_y \cdot R_x \cdot T_{(-x_1, -y_1, -z_1)}$$

Rotation about an Arbitrary Axis

Step 1

Translate (x_1, y_1, z_1) to the origin

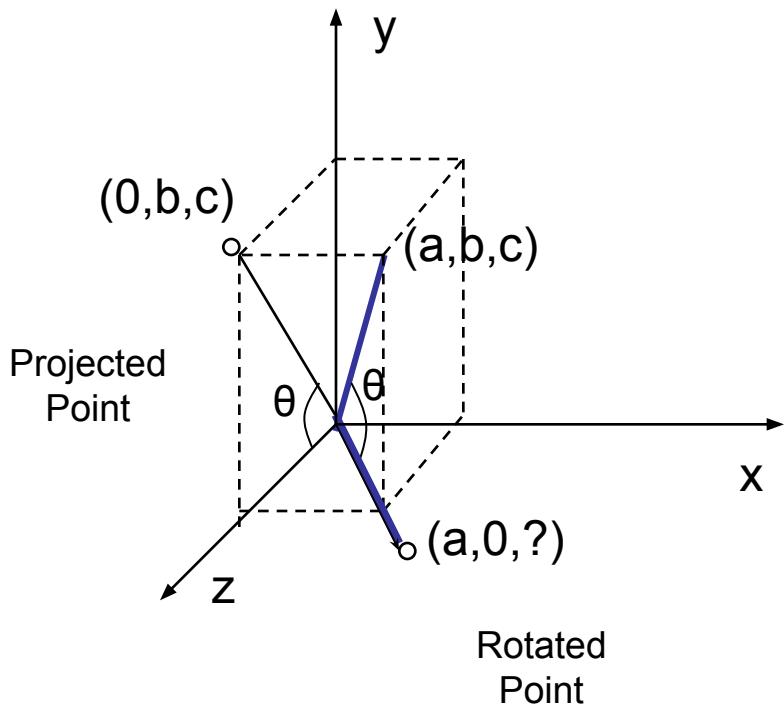


$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

Step 2

Rotate the line about x axis in anti clockwise direction by ' θ ' angle



$$\sin \theta = \frac{b}{\sqrt{b^2 + c^2}} = \frac{b}{d}$$

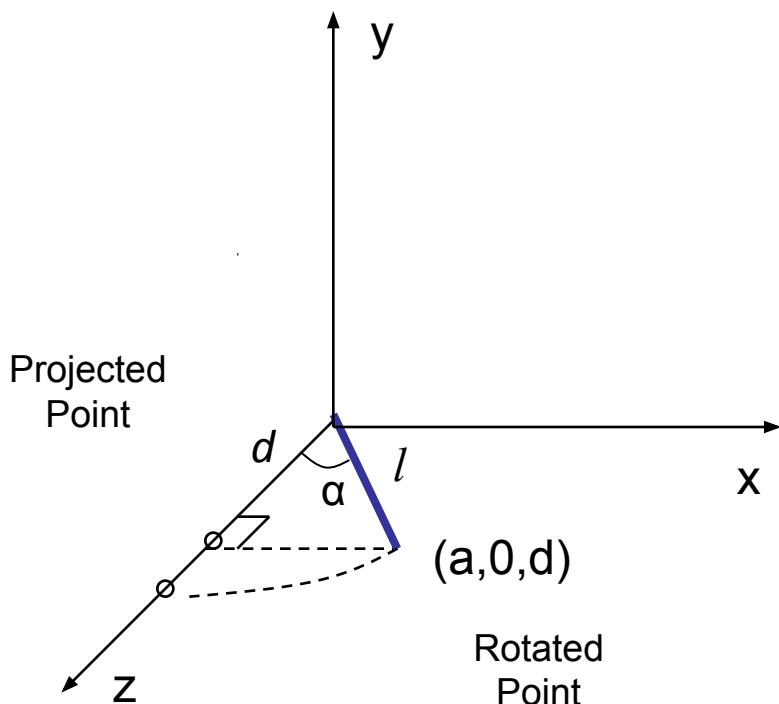
$$\cos \theta = \frac{c}{\sqrt{b^2 + c^2}} = \frac{c}{d}$$

$$[R_{x_\theta}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

Step 3

Rotate the line about y axis in clockwise direction by 'α' angle



$$\sin \alpha = \frac{a}{l}, \quad \cos \alpha = \frac{d}{l}$$

$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

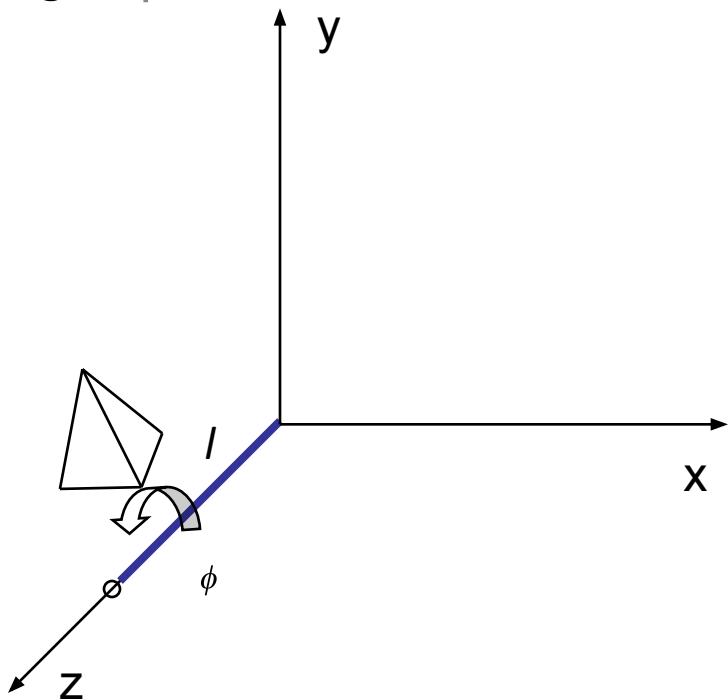
$$d = \sqrt{b^2 + c^2}$$

$$\begin{bmatrix} R_{y_\alpha} \end{bmatrix} = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & -a/l & 0 \\ 0 & 1 & 0 & 0 \\ a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

Step 4

Rotate the object about the line that has been aligned with z axis by the desired angle ' ϕ '



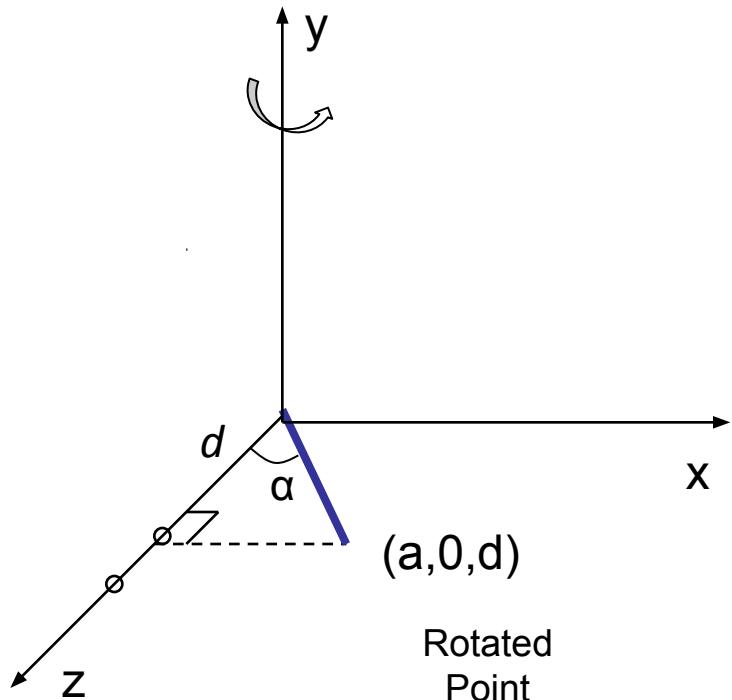
$$[R_{z_\phi}] = \begin{bmatrix} \cos \phi & -\sin \phi & 0 & 0 \\ \sin \phi & \cos \phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

Step 5

(apply reverse transformations to place the arbitrary axis along with the reflected object back to its original position)

Rotate the line about y axis in anti clockwise direction

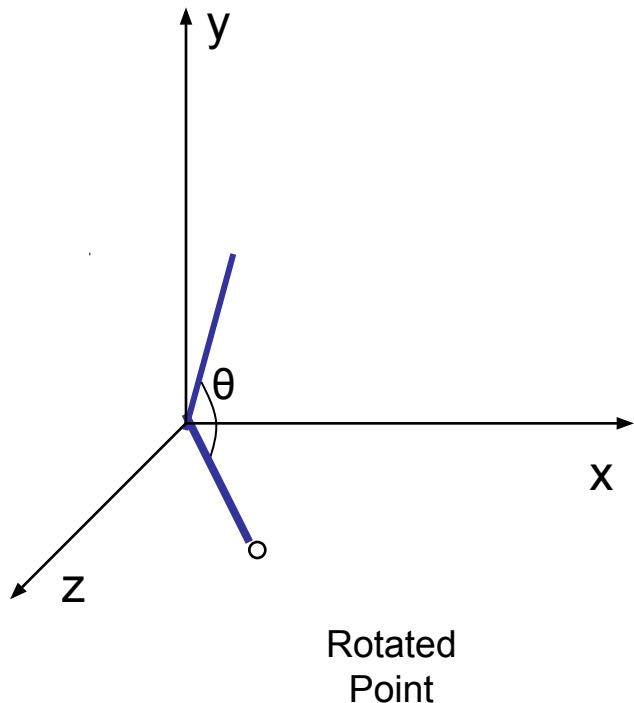


$$\begin{bmatrix} R' \\ y_\alpha \end{bmatrix} = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & a/l & 0 \\ 0 & 1 & 0 & 0 \\ -a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

Step 6

Rotate the line about x axis in clockwise direction

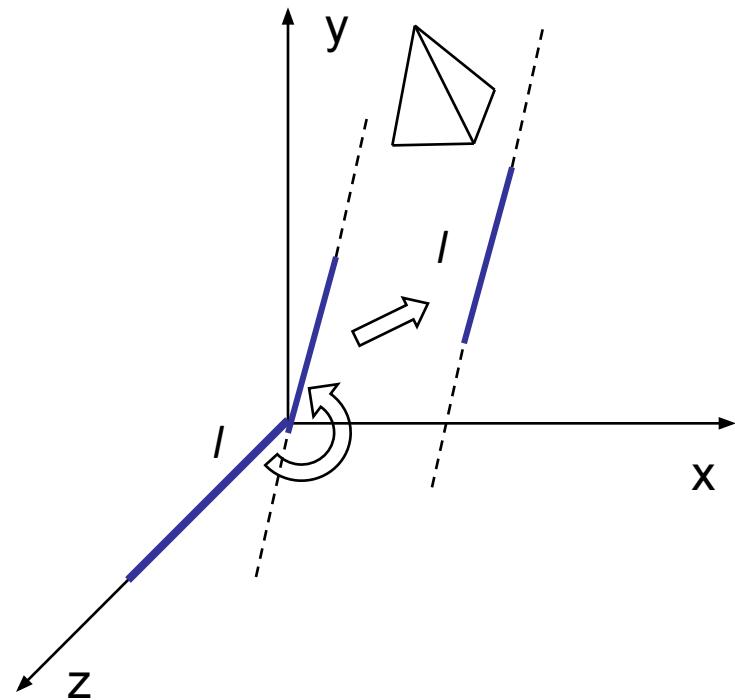


$$\begin{bmatrix} R'_{x_\theta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

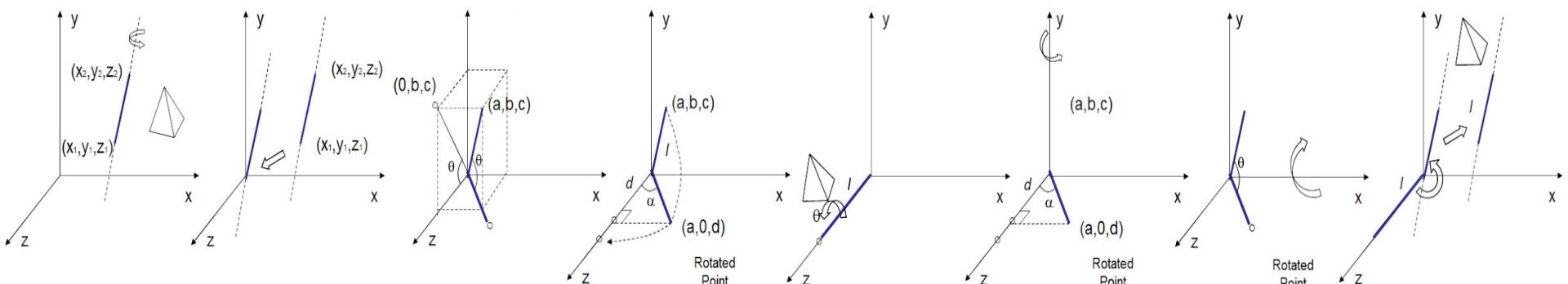
Step 7

Rotate the line about y axis in clockwise direction



$$T' = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis

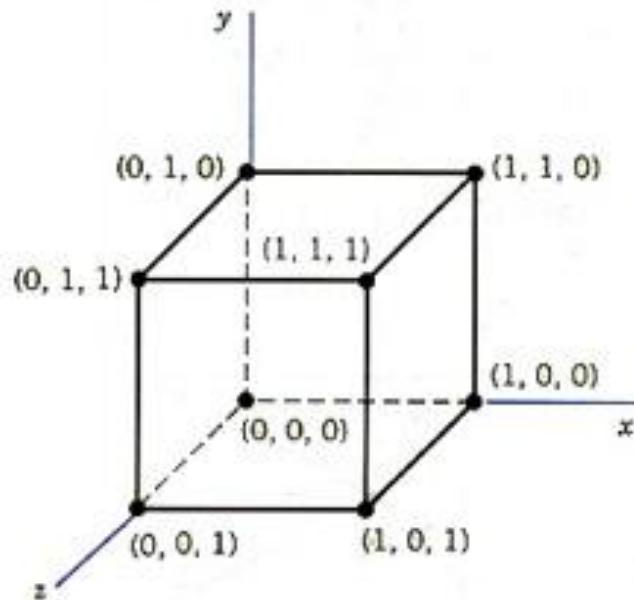


$$CM = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$CM = T' \cdot R'_X \cdot R'_Y \cdot R_Z \cdot R_Y \cdot R_X \cdot T$$

Rotation about an Arbitrary Axis

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

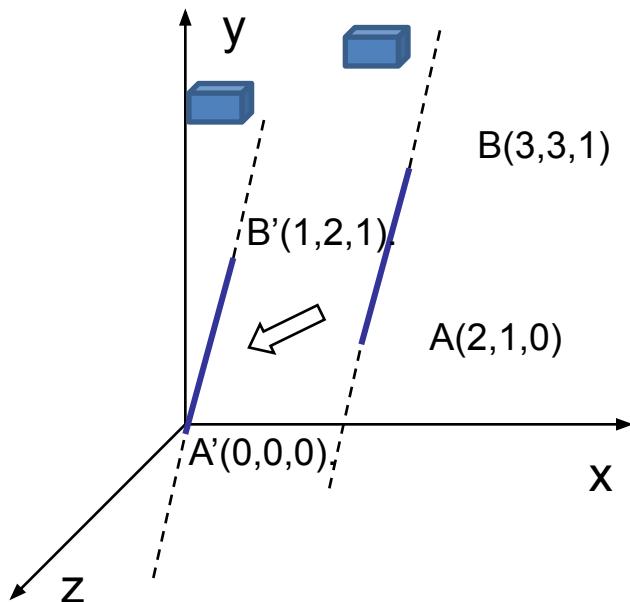


$$cm = T'_{(x_1,y_1,z_1)} \cdot R'_x \cdot R'_y \cdot R_z \cdot R_y \cdot R_x \cdot T_{(-x_1,-y_1,-z_1)}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 1

Translate end point A (2, 1, 0) to the origin

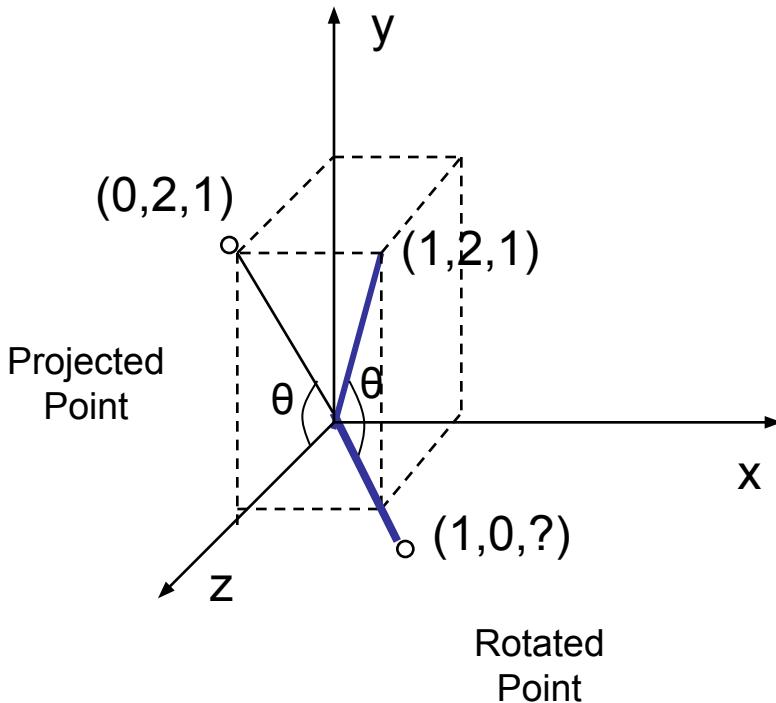


$$T = \begin{bmatrix} 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 2

Rotate the line about x axis in anti clockwise direction by 'θ' angle



$$\sin \theta = \frac{2}{\sqrt{2^2 + 1^2}} = \frac{2}{\sqrt{5}}$$

$$\cos \theta = \frac{1}{\sqrt{2^2 + 1^2}} = \frac{1}{\sqrt{5}}$$

$$[R_{x_\theta}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 3

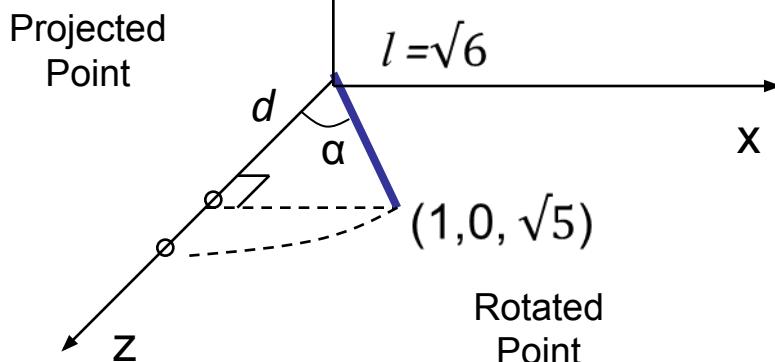
Rotate the line about y axis in clockwise direction by 'α' angle

$$l = \sqrt{6} \quad d = \sqrt{5}$$

$$\sin \alpha = \frac{1}{\sqrt{6}}, \quad \cos \alpha = \frac{\sqrt{5}}{\sqrt{6}}$$

$$l^2 = a^2 + b^2 + c^2 = a^2 + d^2$$

$$d = \sqrt{b^2 + c^2} = \sqrt{5}$$

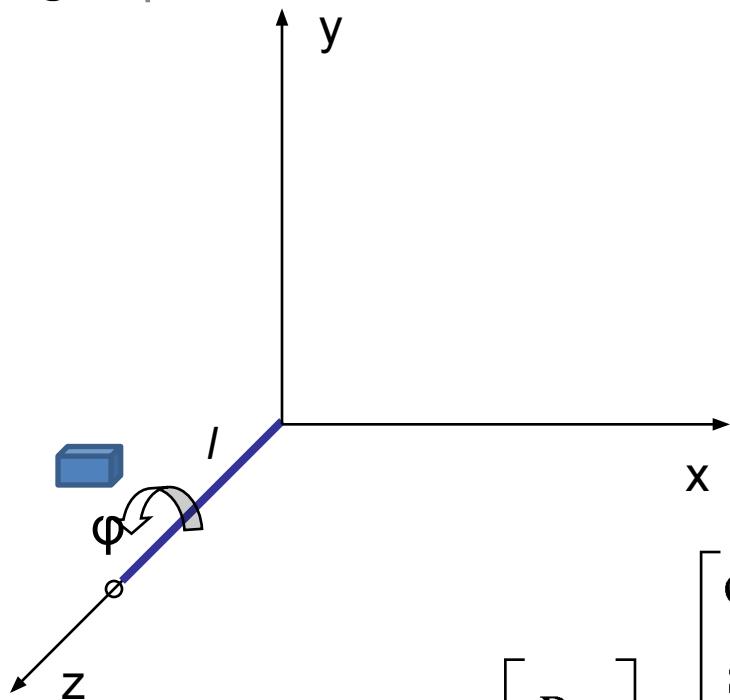


$$\left[R_{y_\alpha} \right] = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & -a/l & 0 \\ 0 & 1 & 0 & 0 \\ a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 4

Rotate the object about the line that has been aligned with z axis by the desired angle ' ϕ '

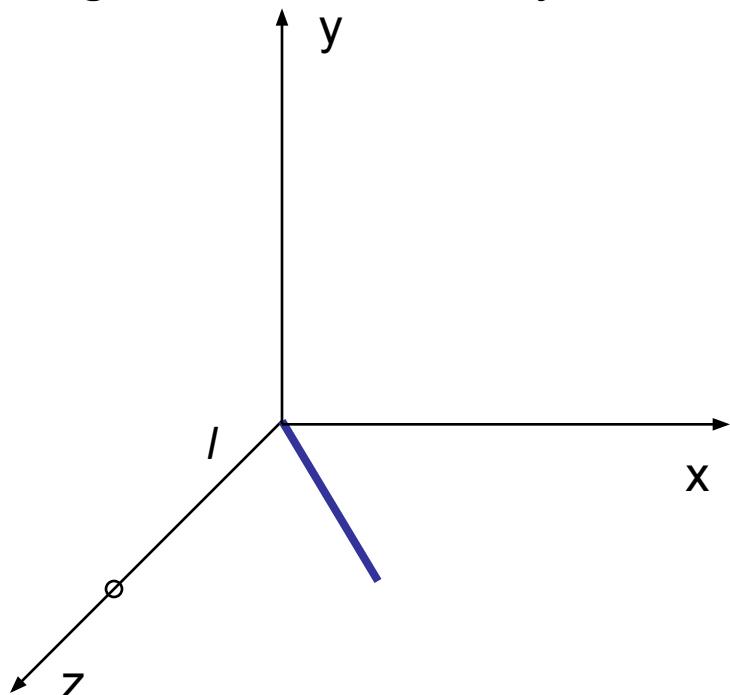


$$\left[R_{z_\phi} \right] = \begin{bmatrix} \cos 90 & -\sin 90 & 0 & 0 \\ \sin 90 & \cos 90 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 5

Rotate the line that has been aligned with z axis by the angle alpha angle in ccw along with the rotated object

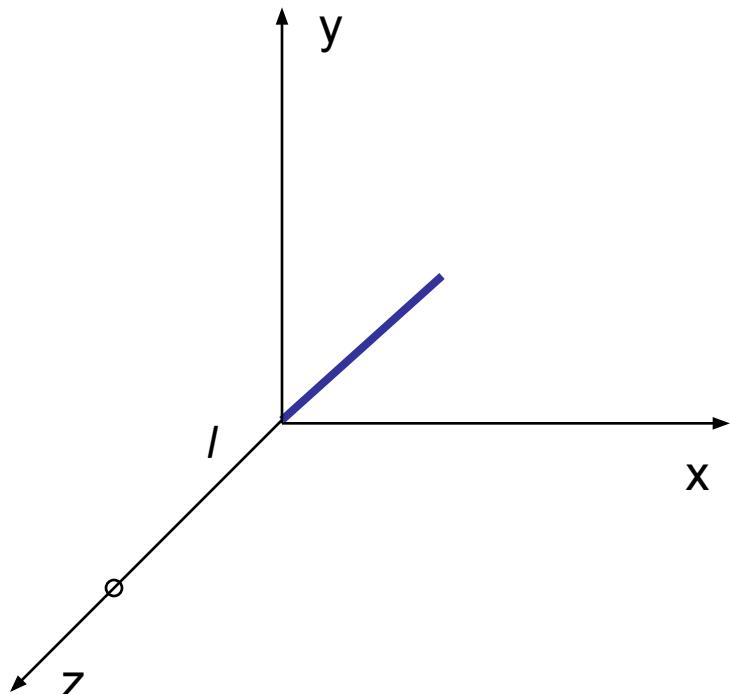


$$\begin{bmatrix} R' \\ y_\alpha \end{bmatrix} = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d/l & 0 & a/l & 0 \\ 0 & 1 & 0 & 0 \\ -a/l & 0 & d/l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 6

Rotate the line about x axis in cw

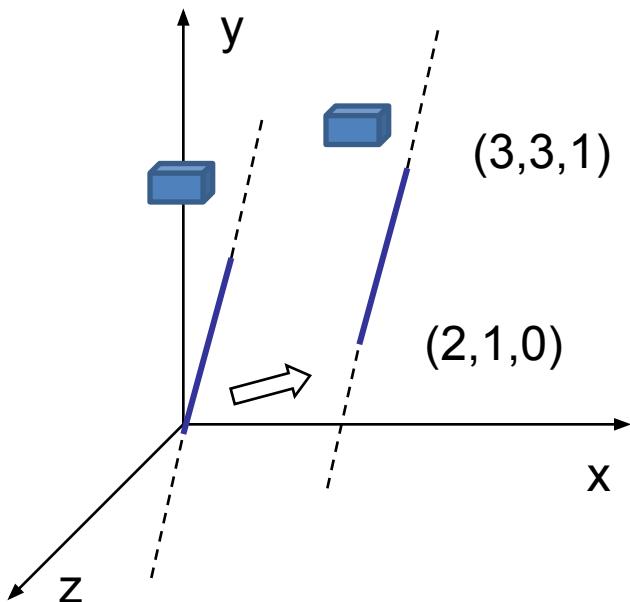


$$\left[R_{x_\theta} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

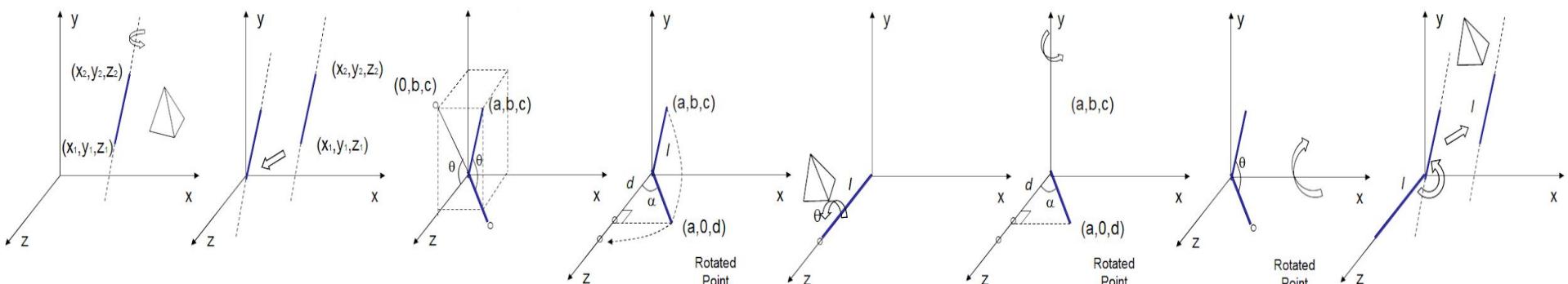
Step 7

Translate end point back to (2, 1, 0)



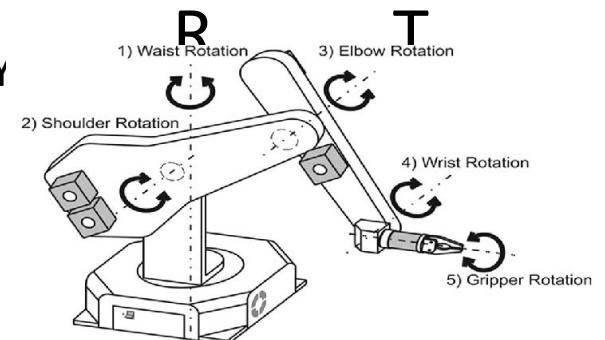
$$T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Axis



$$CM = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Cm = T' \cdot R'_x \cdot R'_y \cdot R_z \cdot R_y$$



Reflection about an Arbitrary Plane

Hints:

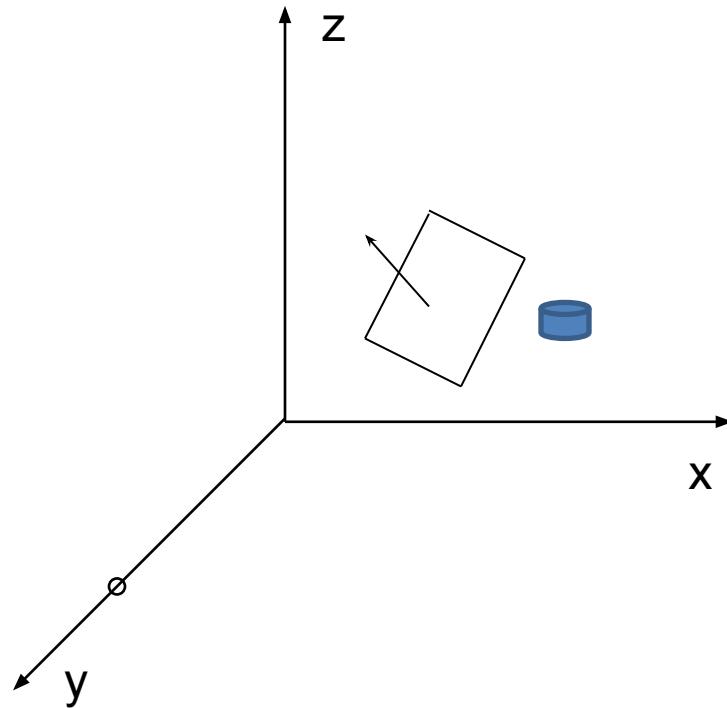
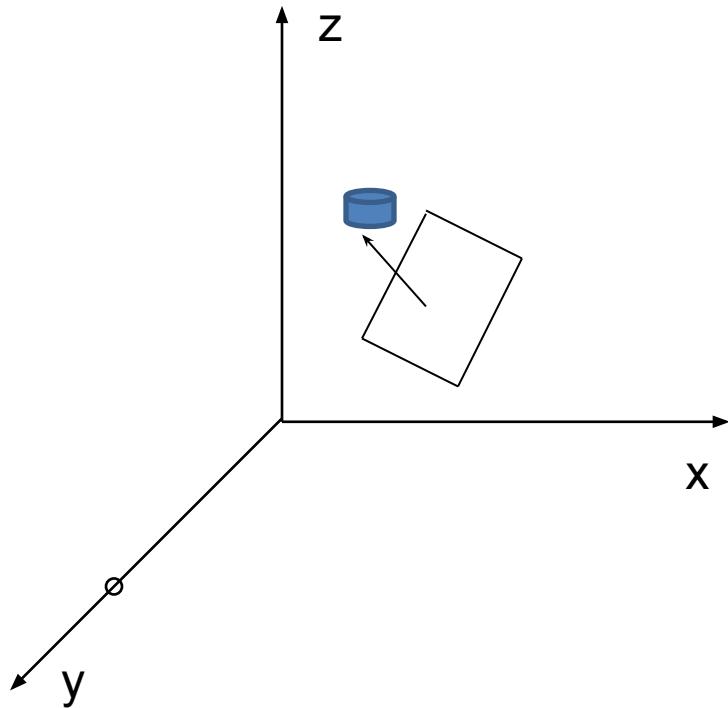
Spatial orientation of a plane in 3D space is given by its outward normal

Translate plane so that it passes thru origin

Rotate the Normal two times so that it gets aligned with z axis and the plane lands on xy plane

Take reflection of object about that plane

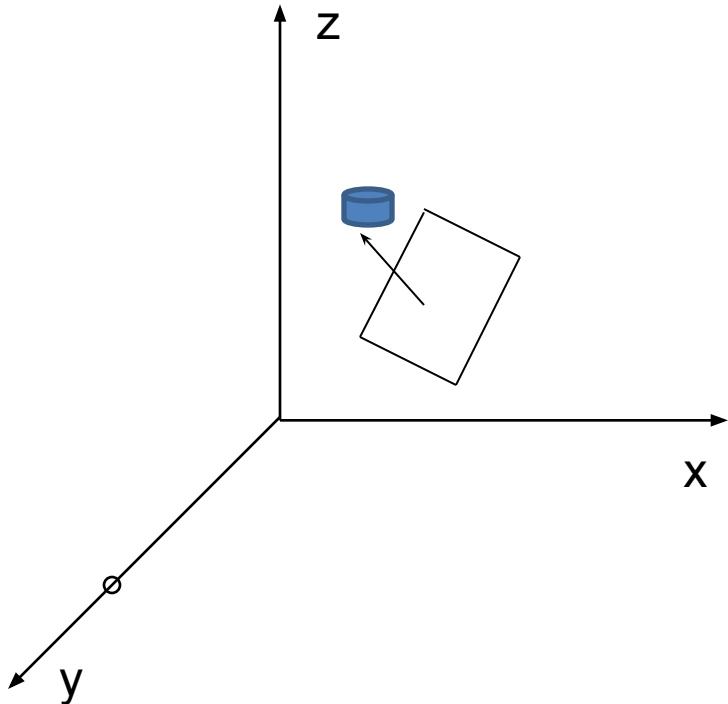
Perform inverse rotations and translation



Reflection about an Arbitrary Plane

Step 1

Translate the plane along with the object to be reflected about it so that the plane passes thru origin

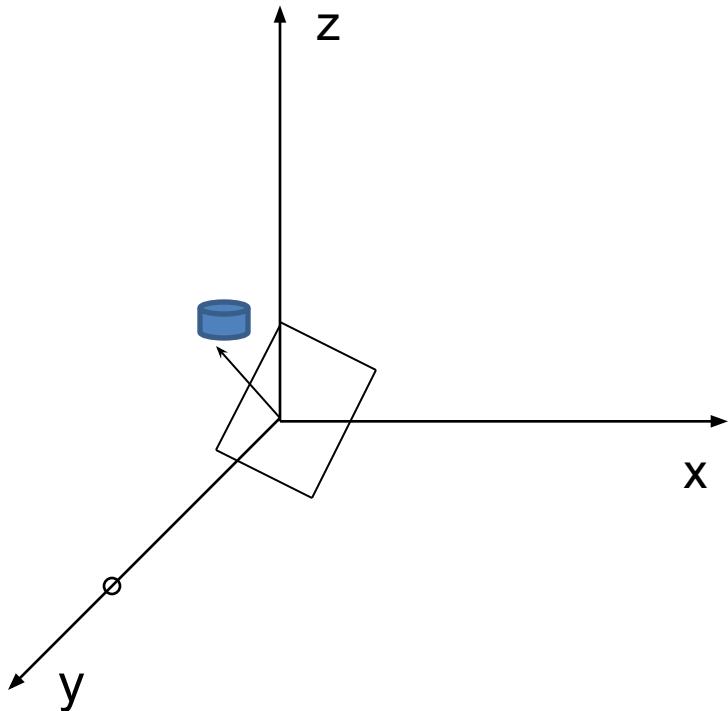


$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

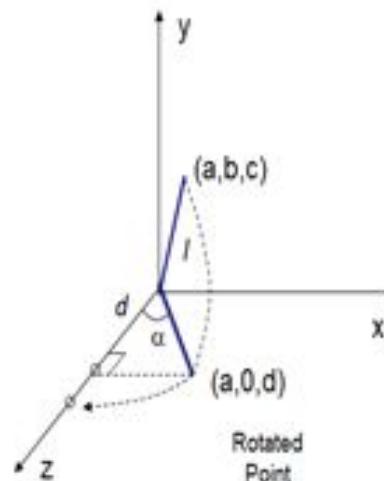
Step 2

Rotate the normal about x then y axis so that the normal is aligned with z axis and the plane is aligned with xy plane



$$\left[R_{x_\theta} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

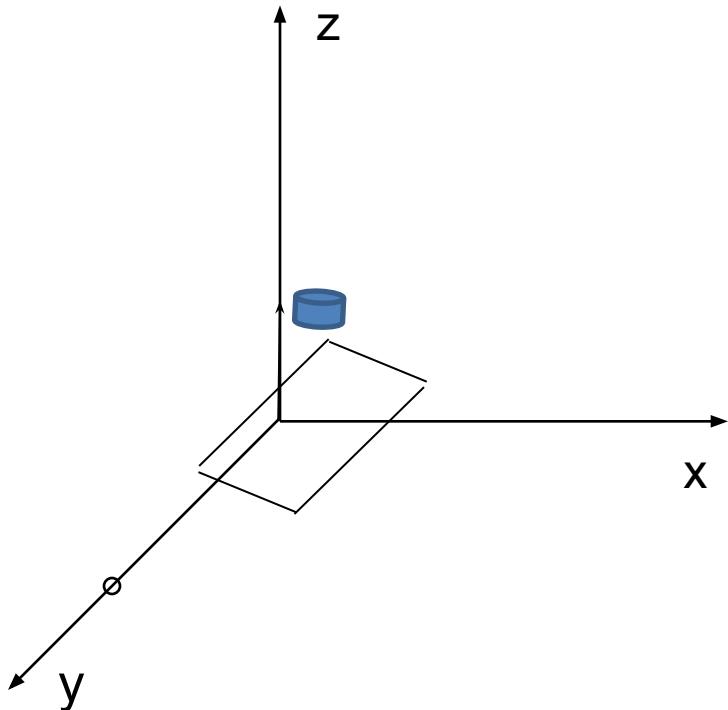
$$\left[R_{y_\alpha} \right] = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Reflection about an Arbitrary Plane

Step 3

Reflect the object about the xy plane with which the plane has been aligned

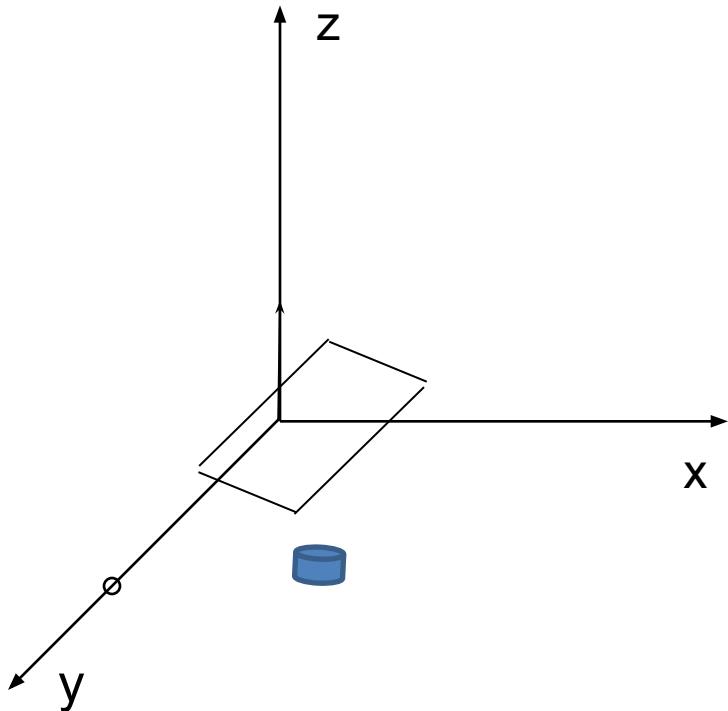


$$[Rf_{xy}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

Step 3

Reflect the object about the xy plane with which the plane has been aligned

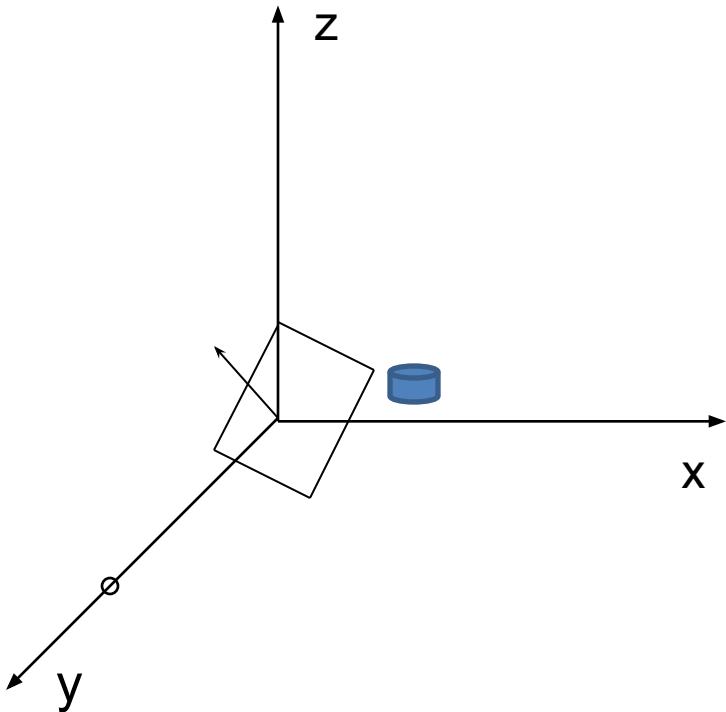


$$[Rf_{xy}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

Step 4

Rotate the normal about y then x axis so that the normal is **de-aligned** from z axis and the plane is de-aligned with xy plane



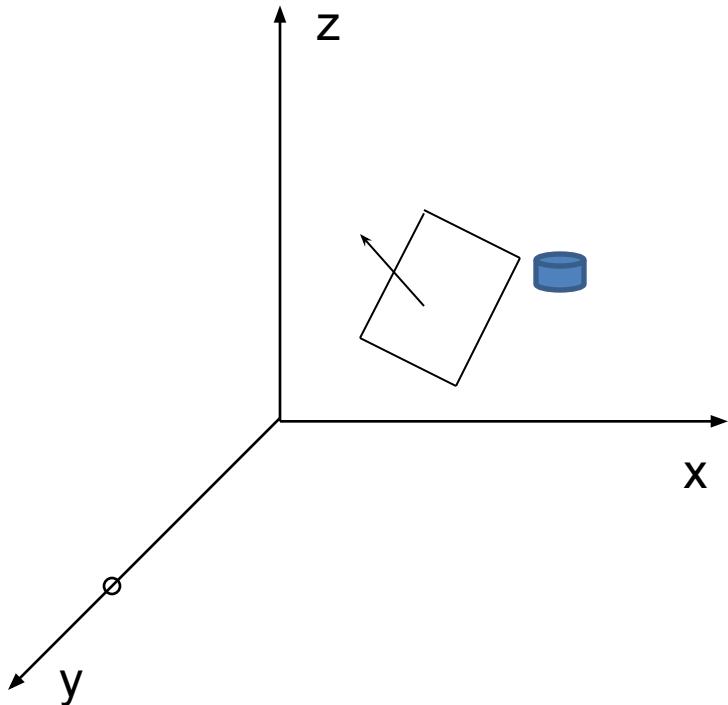
$$\left[R'_{y_\alpha} \right] = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left[R'_{x_\theta} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

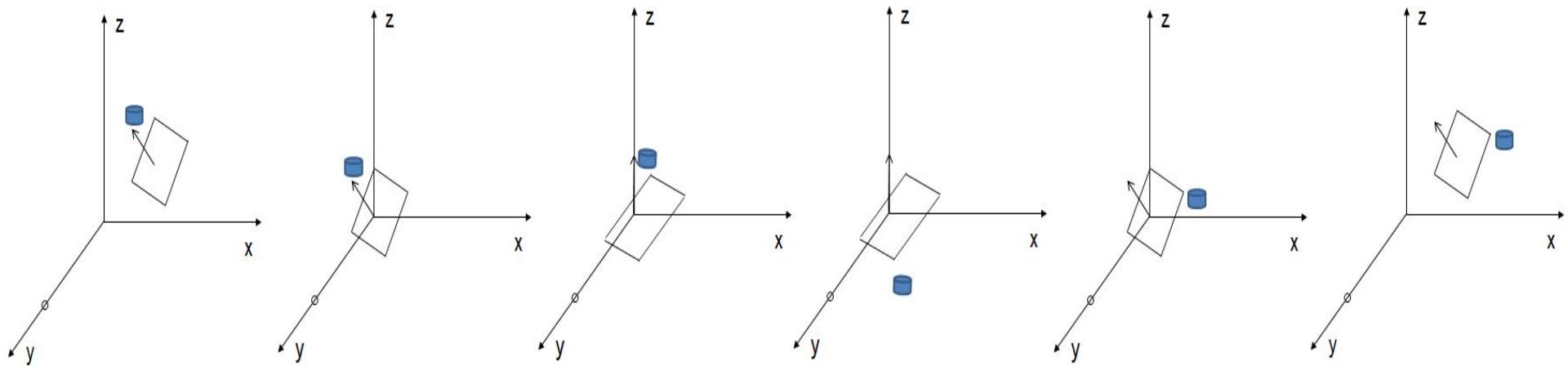
Step 5

Translate the plane along with the **reflected object** back to its original location



$$T' = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

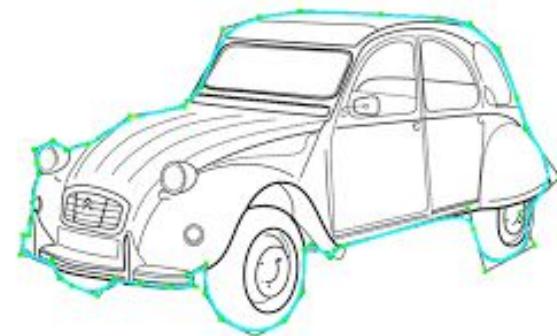
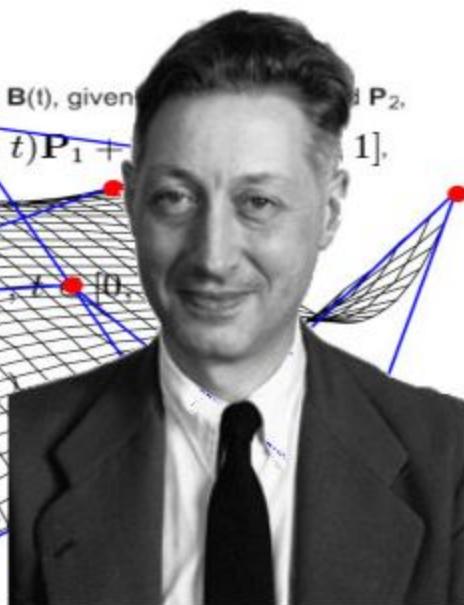
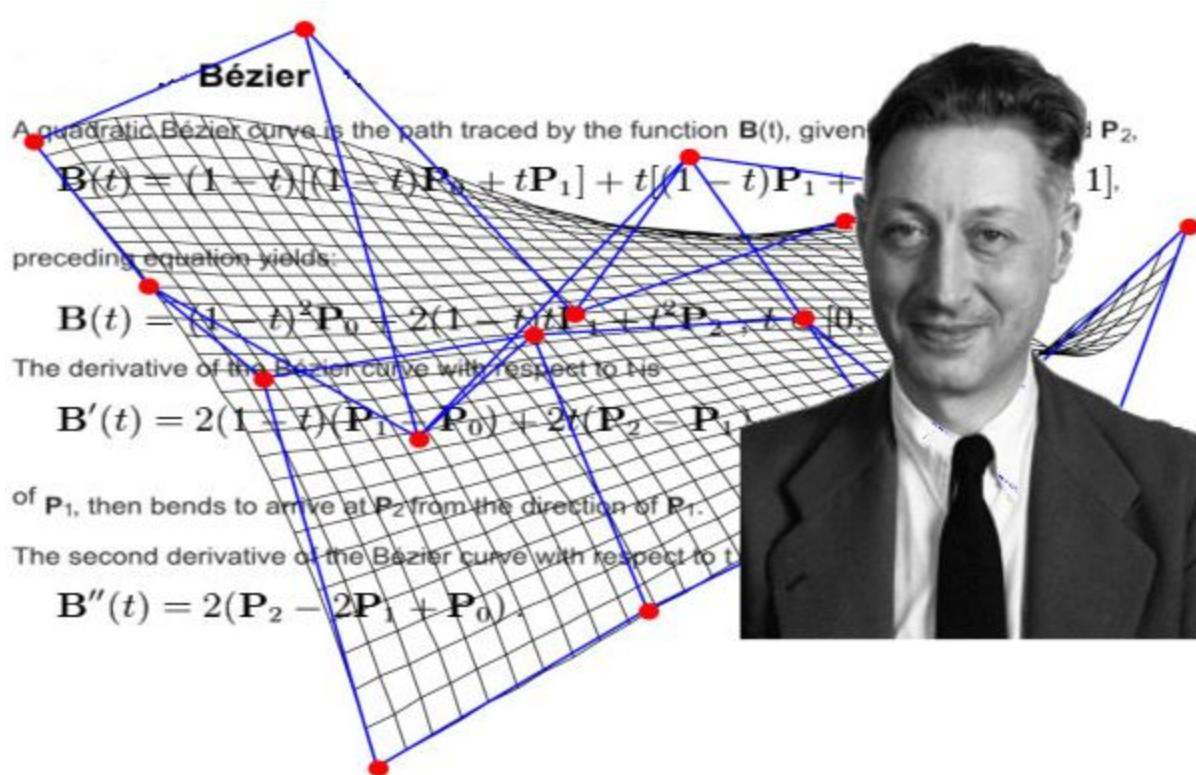


$$CM = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

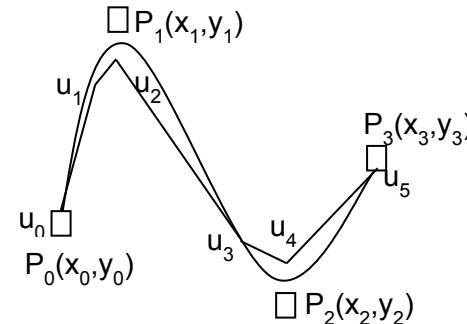
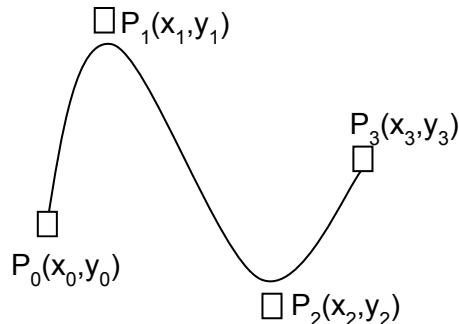
Spline : A spline is a flexible strip that passes thru a designated control points.

Bezier Curve

- Pierre Bezier
- Makes use of **Bernstein** polynomials
- Used in 1960s for designing curves for the bodywork of Renault cars.
- Bezier curves are commonly found in **painting and drawing packages**, as well as CAD system, for generating Computer **fonts** and **animation**



Spline : A spline is a flexible strip that passes thru a designated control points.



Bezier Curve

The above figure shows a **smooth curve comprising of a large number of very small line segments**. For understanding the concept to draw such a line we deal with a curve as show above which is an **approximation of the curve with five line segments** only

The approach below is used to draw a **curve for any number of control points**

n = number of control points -1 is its order ($n = 1$ for linear, 2 for quadratic, etc.)

Suppose P_0, P_1, P_2, P_3 are four control points ($n = 3$ Cubic Bezier Curve)

Number of segments in a line segment : nSeg (= 5 say)

$i = 0$ to $nSeg$

$u = i/nSeg \quad [0,1] \quad 0 \leq u \leq 1$

$u_0 = 0/5 \quad u_1 = 1/5 \quad u_2 = 2/5 \quad u_3 = 3/5 \quad u_4 = 4/5 \quad u_5 = 5/5$

$$x(u) = \sum_{j=0}^n x_j BEZ_{j,n}(u) \quad n \text{ (order)} : \text{number of control points - 1}$$

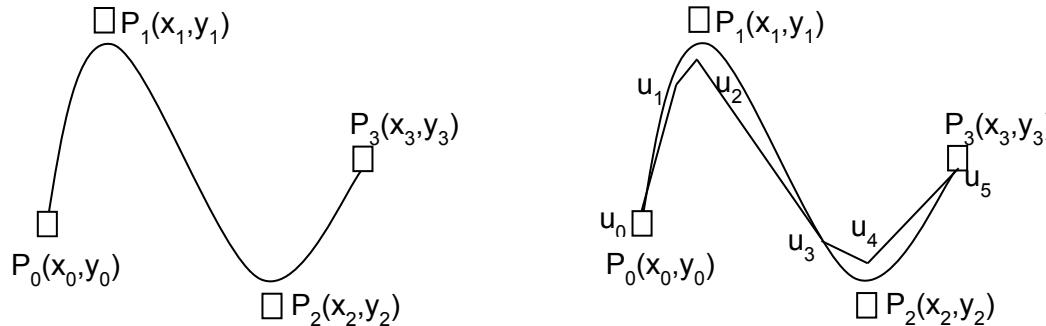
$$x(u) = x_0 BEZ_{0,3}(u) + x_1 BEZ_{1,3}(u) + x_2 BEZ_{2,3}(u) + x_3 BEZ_{3,3}(u)$$

Similarly

$$y(u) = \sum_{j=0}^n y_j BEZ_{j,n}(u) \quad n \text{ (order)} : \text{number of control points - 1}$$

$$y(u) = y_0 BEZ_{0,3}(u) + y_1 BEZ_{1,3}(u) + y_2 BEZ_{2,3}(u) + y_3 BEZ_{3,3}(u)$$

Spline : A spline is a flexible strip that passes thru a designated control points.



$$x(u) = x_0 \text{BEZ}_{0,3}(u) + x_1 \text{BEZ}_{1,3}(u) + x_2 \text{BEZ}_{2,3}(u) + x_3 \text{BEZ}_{3,3}(u)$$

$$y(u) = y_0 \text{BEZ}_{0,3}(u) + y_1 \text{BEZ}_{1,3}(u) + y_2 \text{BEZ}_{2,3}(u) + y_3 \text{BEZ}_{3,3}(u)$$

The Bezier blending function $\text{BEZ}_{j,n}(u)$ is defined as,

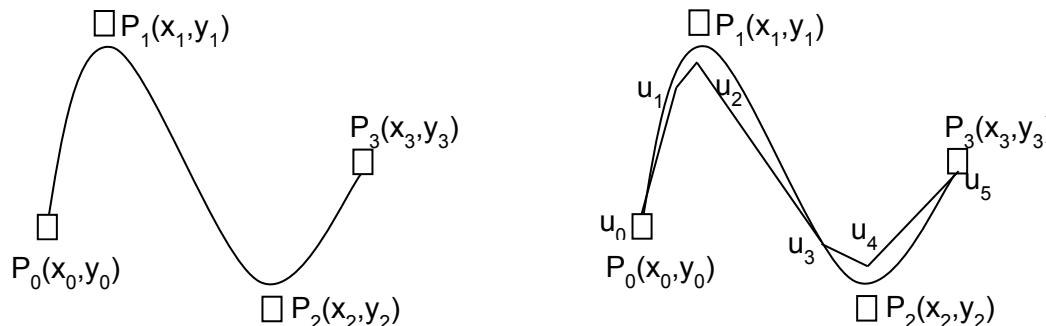
$$\text{BEZ}_{j,n}(u) = \frac{n!}{j!(n-j)!} u^j (1-u)^{n-j}$$

$$\text{BEZ}_{j,n}(u) = C_{(n,j)} u^j (1-u)^{n-j} \quad \text{Where } C_{(n,j)} \text{ is the Binomial Coefficient}$$

$$C_{(n,j)} = \frac{n!}{j!(n-j)!}$$

Blending Function
Control Points
Convex Hull

Spline : A spline is a flexible strip that passes thru a designated control points.



For each 'u' the coordinates x and y are computed and desired curve is produced when the adjacent coordinates (x,y) are connected with a straight line segment

$$\begin{aligned} x(u) &= x_0 \text{BEZ}_{0,3}(u) + x_1 \text{BEZ}_{1,3}(u) + x_2 \text{BEZ}_{2,3}(u) + x_3 \text{BEZ}_{3,3}(u) \\ y(u) &= y_0 \text{BEZ}_{0,3}(u) + y_1 \text{BEZ}_{1,3}(u) + y_2 \text{BEZ}_{2,3}(u) + y_3 \text{BEZ}_{3,3}(u) \end{aligned}$$

Now

$$Q(u) = P_0 \text{BEZ}_{0,3}(u) + P_1 \text{BEZ}_{1,3}(u) + P_2 \text{BEZ}_{2,3}(u) + P_3 \text{BEZ}_{3,3}(u)$$

Four blending functions must be found based on Bernstein Polynomials

$$\text{BEZ}_{0,3}(u) = \frac{3!}{0! 3!} u^0 (1-u)^3 = (1-u)^3 \quad \text{BEZ}_{1,3}(u) = \frac{3!}{1! 2!} u^1 (1-u)^2 = 3u(1-u)^2$$

$$\text{BEZ}_{2,3}(u) = \frac{3!}{2! 1!} u^2 (1-u) = 3u^2 (1-u) \quad \text{BEZ}_{3,3}(u) = \frac{3!}{3! 0!} u^3 (1-u)^0 = u^3$$

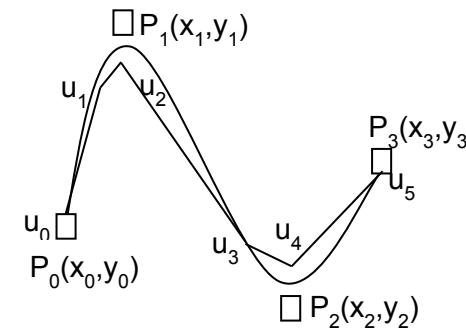
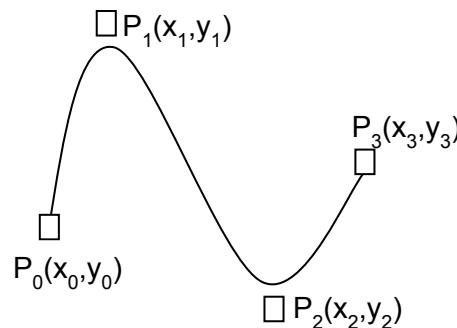
Normalizing properties apply to blending functions, that means they all add up to one

Substituting these functions in above equation

$$Q(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$$

When $u = 0$ then $Q(u) = P_0$ and when $u = 1$ then $Q(u) = P_3$

Spline : A spline is a flexible strip that passes thru a designated control points.



$$Q(u) = (1-u)^3 P_0 + 3u(1-u)^2 P_1 + 3u^2(1-u) P_2 + u^3 P_3$$

in Matrix Form

$$Q(u) = \begin{bmatrix} (1-u)^3 & 3u(1-u)^2 & P_1 & 3u^2(1-u) & u^3 \\ & P_3 & & & \end{bmatrix}$$

$$\left[\begin{array}{c} P_2 \\ P_0 \end{array} \right]$$

or

$$Q(u) = \begin{bmatrix} (1-3u+3u^2-u^3) & (3u-6u^2+3u^3) & (3u^2-3u^3) & u^3 \\ & P_3 & & \end{bmatrix}$$

$$\left[\begin{array}{c} P_2 \\ P_0 \end{array} \right]$$

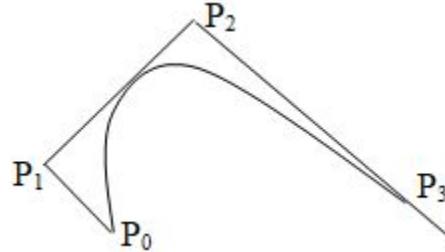
or

$$Q(u) = \begin{bmatrix} -1 & 3 & -3 & \\ 3 & -6 & 3 & \\ u^3 & u^2 & u^1 & 1 \\ 1 & 0 & 0 & \end{bmatrix} \begin{bmatrix} 1 & P_0 \\ 0 & P_1 \\ -3 & 3 & 0 & 0 & P_2 \\ 0 & P_3 & & & \end{bmatrix}$$

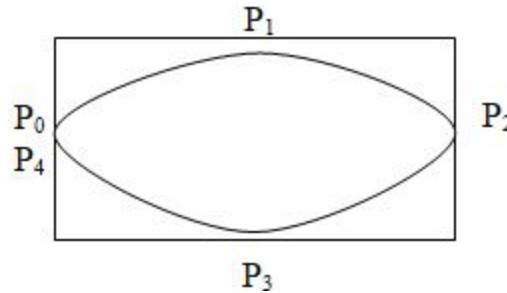
$$\left[\begin{array}{c} P_2 \\ P_0 \end{array} \right]$$

Properties of a Bezier Curve

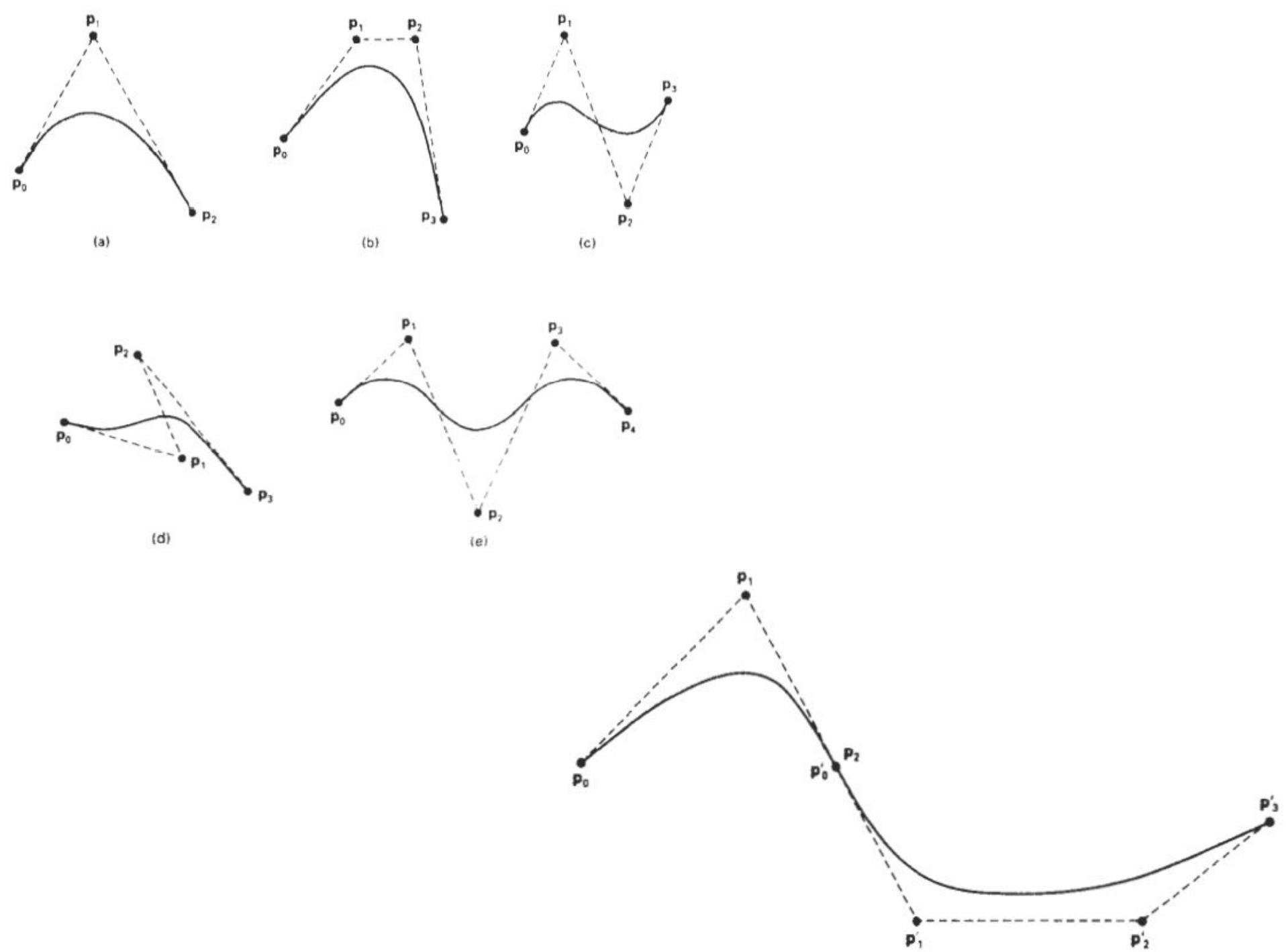
1. Bezier curve **lies in the convex hull of the control points** which ensure that the curve smoothly follows the control Points



2. Four Bezier polynomials are used in the construction of curve to fit four control points
3. It always passes thru the **end points (first and last control points)**
4. Closed curves can be generated by specifying **the first and last control points at the same position**



5. Specifying multiple control points at a single position **gives more weight** to that position
6. Complicated curves are formed by **piecing several sections** of lower degrees together
7. The tangent to the curve at an end point is along the line joining the end point to the adjacent control point



Bezier surfaces (Non Planar Surfaces)

To create a Bézier surface, We blend a mesh of Bézier curves using the blending function

$$P(u, v) = \sum_{j=0}^m \sum_{k=0}^n P_{jk} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

where j and k are points in parametric space and P_{jk} represents the location of the knots in real space. The Bézier functions specify the weighting of a particular knot. They are the Bernstein coefficients. The definition of the Bézier functions is

$$BEZ_{k,n}(u) = C(n, k) u^k (1-u)^{n-k}$$

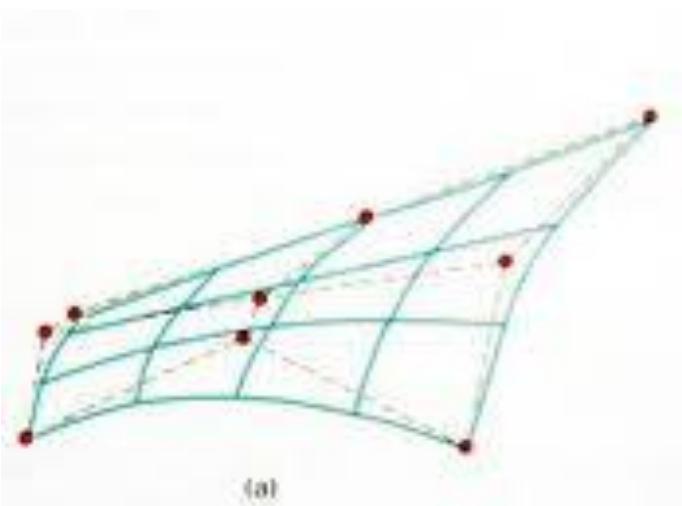
where $C(n, k)$ represents the binary coefficients. When $u=0$, the function is one for $k=0$ and zero for all other points.

When we combine **two orthogonal parameters**, we find a Bézier curve along each edge of the surface, as defined by the points along that edge.

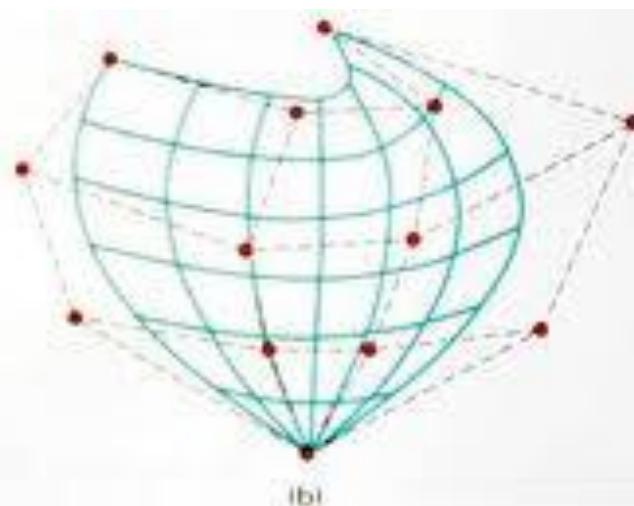
Bézier surfaces are useful for interactive design and were first applied to car body design.

The properties of Bezier surfaces are controlled by the blending functions

- The surface takes the general **shape of the control points**
- The surface is **contained within the convex hull** of the control points
- The **corners of the surface and the corner control vertices are coincident**

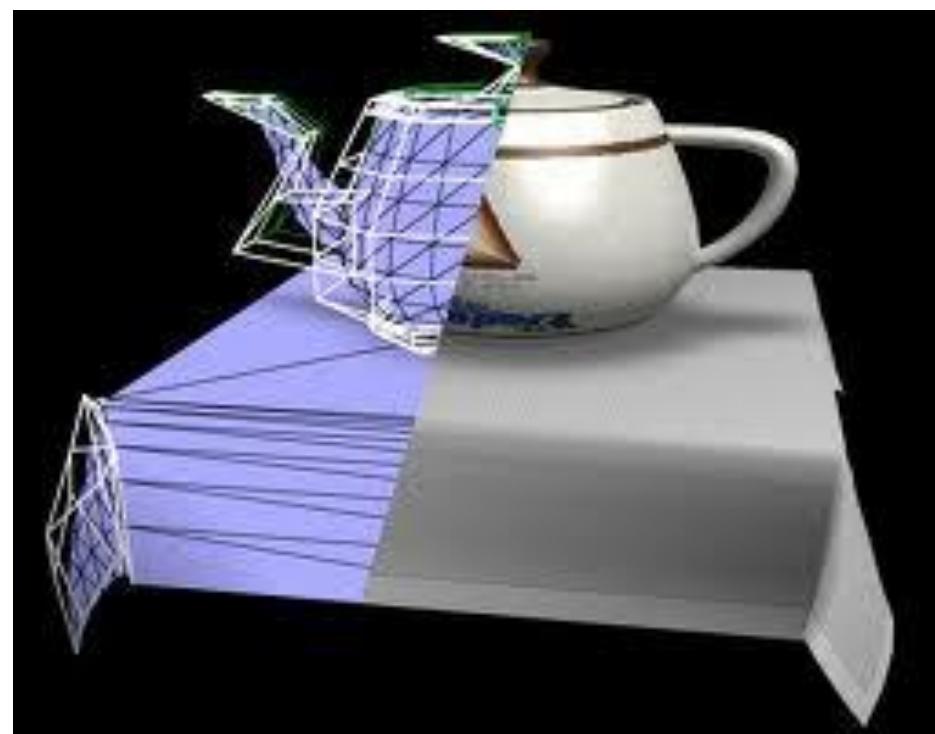


(a)



(b)

HEARN & BAKER, 1997



Parametric Cubic Curve

3

$$\text{A parametric cubic curve is defined as } P(t) = \sum_{i=0}^3 a_i t^i \quad 0 \leq t \leq 1 \quad \text{--- (i)}$$

where, $P(t)$ is a point on the curve

Expanding equation (i) yields

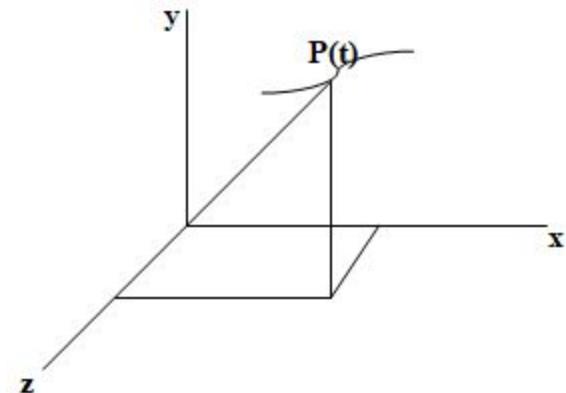
$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad \text{--- (ii)}$$

This equation is separated into three components of $P(t)$

$$x(t) = a_{3x} t^3 + a_{2x} t^2 + a_{1x} t + a_{0x}$$

$$y(t) = a_{3y} t^3 + a_{2y} t^2 + a_{1y} t + a_{0y}$$

$$z(t) = a_{3z} t^3 + a_{2z} t^2 + a_{1z} t + a_{0z} \quad \text{--- (iii)}$$



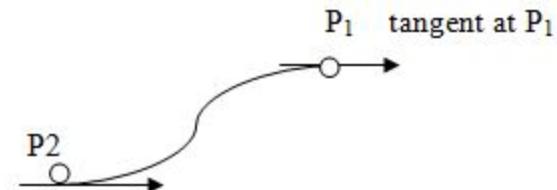
To solve (iii), **twelve unknown coefficients a_{ij}** (algebraic coefficients) must be specified

From **known end point coordinates** of each segment, six of twelve needed equations are obtained.

The other six are found by using **tangent vectors** at the two ends of each segment

Direction of tangent vectors establishes slopes(direction cosines) of curve at end points

This procedure for defining a cubic curve using end points and tangent vector is one form of **Hermite Interpolation**



$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad \text{-----(ii)}$$

Each cubic curve segment is parameterized from 0 to 1 so that known end points correspond to the limit values of the parametric variable t , that is $P(0)$ and $P(1)$

Substituting $t = 0$ and $t = 1$ the relationship between two end point vectors and the algebraic coefficients are found

$$\begin{aligned} P(0) &= a_0 & {}_3P(1) &= 3a_3 + 3a_2 + 3a_1 + 3a_0 \\ P'(1) &= 3a_3 + 2a_2 + a_1 \\ 3P(1) - P'(1) &= a_2 + 2a_1 + 3a_0 \end{aligned}$$

To find the tangent vectors equation ii must be differentiated with respect to t

$$P'(t) = 3a_3 t^2 + 2a_2 t + a_1$$

The tangent vectors at the two end points are found by substituting $t = 0$ and $t = 1$ in this equation

$$P'(0) = a_1 \quad P'(1) = 3a_3 + 2a_2 + a_1$$

The algebraic coefficients ' a_i ' in equation (ii) can now be written explicitly in terms of boundary conditions – endpoints and tangent vectors

$$a_0 = ? \quad a_1 = ?$$

$$a_2 = ? \quad a_3 = ?$$

The algebraic coefficients 'a_i' in equation

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \quad \text{---(ii)}$$

can now be written explicitly in terms of boundary conditions – endpoints and tangent vectors

$$a_0 = P(0)$$

$$a_1 = P'(0)$$

$$a_2 = -3 P(0) + 3 P(1) - 2 P'(0) - P'(1)$$

$$a_3 = 2 P(0) - 2 P(1) + P'(0) + P'(1)$$

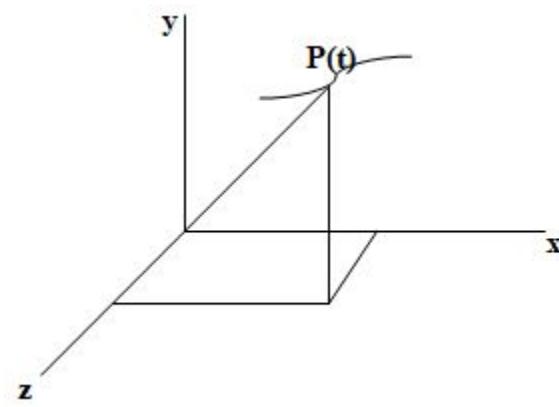
Substituting these values of 'a_i' in equation (ii) and rearranging the terms yields

P(0), P(1), P'(0), P'(1): *geometric coefficients*, represent known vector quantities

$$P(t) = (2t^3 - 3t^2 + 1) P(0) + (-2t^3 + 3t^2) P(1) + (t^3 - 2t^2 + t) P'(0) + (t^3 - t^2) P'(1)$$

Polynomial coefficients of vector quantities: *blending functions*

By varying parameter t in these blending function from 0 to 1 several points on curve segments can be found



Calculate (x,y) coordinates of Bézier curve described by the following **4 control points**: $P_0(0,0)$, $P_1(1,2)$, $P_2(3,3)$, $P_3(4,0)$ with **5 line segments**

For four control points, $n = 3$. = no of control points -1

1. Calculate all the blending functions, $BEZ_{j,n}$ for $j = 0, \dots, n$ using the formula:

$$BEZ_{j,n}(u) = \frac{n!}{j!(n-j)!} u^j (1-u)^{n-j}$$

$$BEZ_{0,3}(u) = (1-u)^3 =$$

$$BEZ_{1,3}(u) = 3u(1-u)^2 =$$

$$BEZ_{2,3}(u) = 3u^2(1-u) = BEZ_{3,3}(u) = u^3$$

2. Calculate the points on the curve.

$$\begin{aligned} x(u) &= x_0 BEZ_{0,3}(u) + x_1 BEZ_{1,3}(u) + x_2 BEZ_{2,3}(u) + x_3 BEZ_{3,3}(u) \\ y(u) &= y_0 BEZ_{0,3}(u) + y_1 BEZ_{1,3}(u) + y_2 BEZ_{2,3}(u) + y_3 BEZ_{3,3}(u) \end{aligned}$$

$$u_0 = 0/5 = 0 \quad x(0.0) = 0 \quad y(0.0) = 0$$

$$u_3 = 3/5 = 0.6 \quad x(0.6) = 1 \quad y(0.6) = 1.7$$

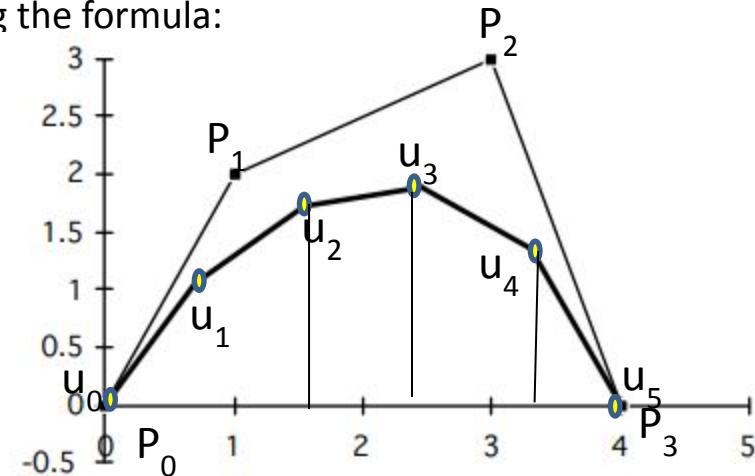
$$\begin{aligned} u_1 &= 1/5 = 0.2 \quad x(0.2) = 0. (1-0.2)^3 + 1. 3.0.2 (1-0.2)^2 + 3. 3.0.2^2 (1-0.2) + 4.0.2^3 = 0.6 \\ y(0.2) &= 0. (1-0.2)^3 + 2. 3.0.2 (1-0.2)^2 + 3. 3.0.2^2 (1-0.2) + 0.0.2^3 = 1.34 \end{aligned}$$

$$u_2 = 2/5 = 0.4 \quad x(0.4) = 1.5 \quad y(0.4) = 1.7$$

$$u_3 = 1/5 = 0.6 \quad x(0.6) = 2.45 \quad y(0.6) = 1.9$$

$$u_4 = 4/5 = 0.8 \quad x(0.8) = 3.3 \quad y(0.8) = 1.3$$

$$u_5 = 5/5 = 1 \quad x(1) = 4 \quad y(0.4) = \quad y(1) = 0$$



Calculate (x,y) coordinates of Bézier curve described by the following **4 control points**: (0,0), (1,2), (3,3), (4,0) with **11 line segments**

For four control points, $n = 3$.

1. Calculate all the blending functions, $BEZ_{j,n}$ for $j = 0, \dots, n$ using the formula:

$$BEZ_{j,n}(u) = \frac{n!}{j!(n-j)!} u^j (1-u)^{n-j}$$

$$BEZ_{0,3}(u) = (1-u)^3$$

$$BEZ_{1,3}(u) = 3u(1-u)^2$$

$$BEZ_{2,3}(u) = 3u^2(1-u)$$

$$BEZ_{3,3}(u) = u^3$$

2. Calculate the points on the curve.

$$u = 0 \quad x(0) = \quad u = \quad x(\) =$$

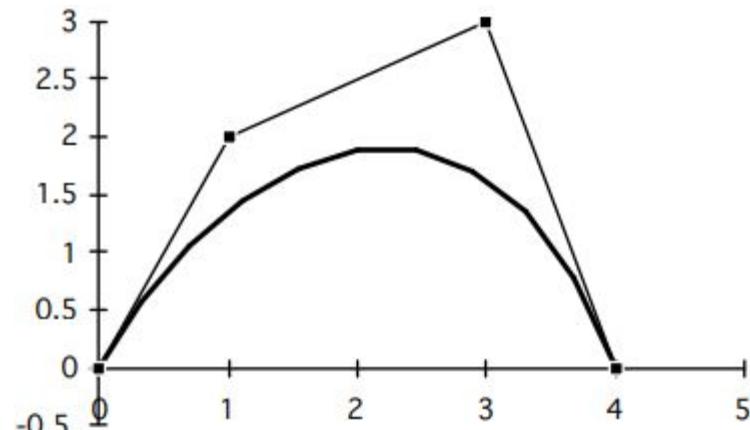
$$y(0) = \quad y(\) =$$

$$u = \quad x(\) = \quad u = \quad x(\) =$$

$$y(\) = \quad y(\) =$$

$$u = \quad x(\) = \quad u = 1 \quad x(1) =$$

$$y(\) = \quad y(1) =$$



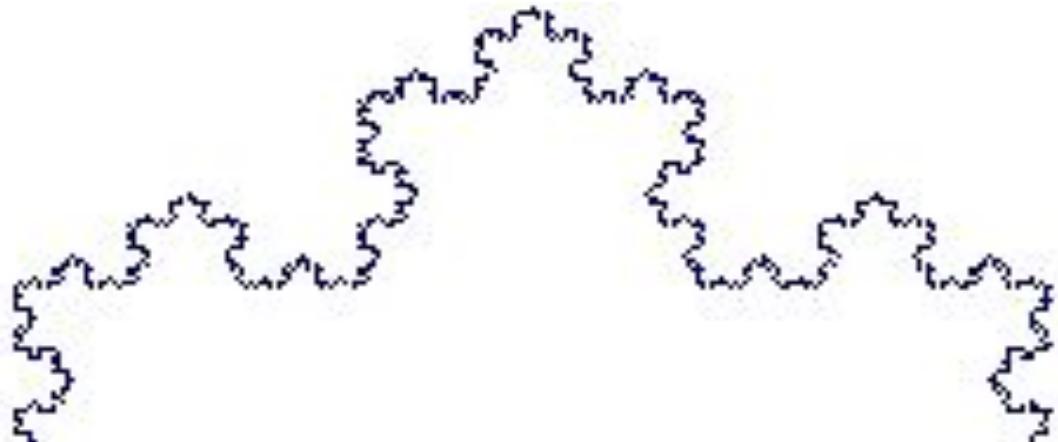
Fractal Geometry Methods & Procedural Modelling

- Natural objects can be realistically described using **fractal geometry methods**
- Fractal methods use procedures rather than equations to model objects - **procedural modelling**
- The major characteristic of any procedural model is that the model is not based on data, but rather on the implementation of a procedure following a particular set of rules

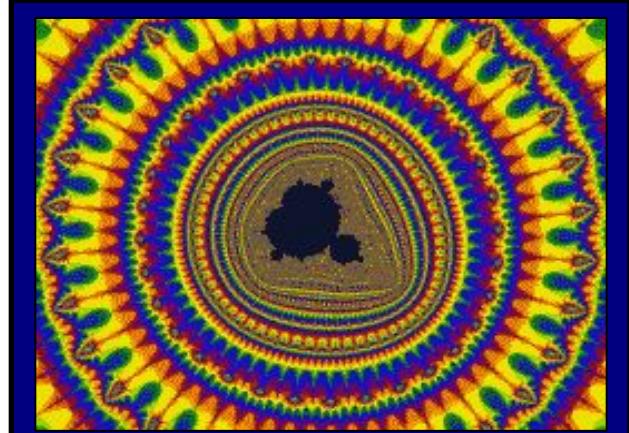
Modelling On The Fly!

Fractals

- A fractal object has two basic characteristics:
 - Infinite detail at every point
 - A certain self similarity between object parts and the overall features of the object



The Koch Curve



Mandelbrot Set Video From:

<http://www.fractal-animation.net/ufvp.htm>

Generating Fractals

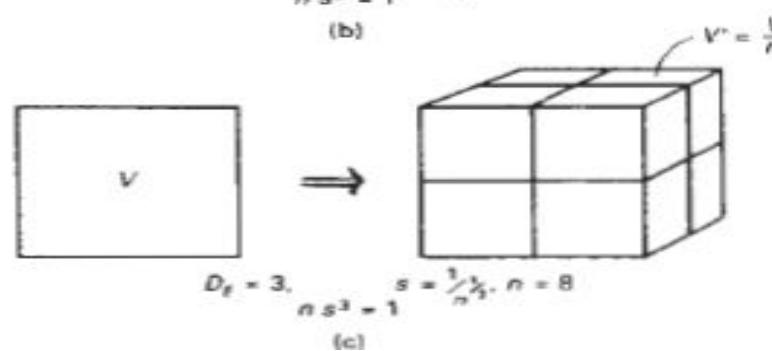
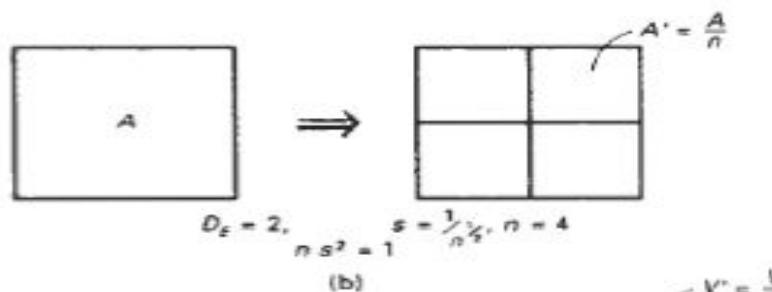
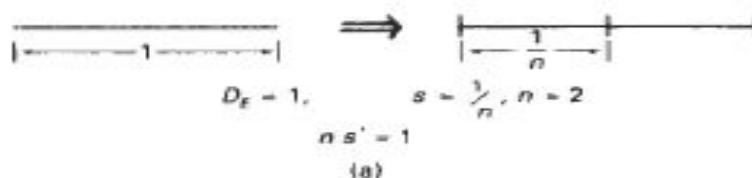
- A fractal object is generated by repeatedly applying a specified transform function to points in a region of space
- If $P_0 = (x_0, y_0, z_0)$ is a selected initial position, each iteration of a transformation function F generates successive levels of detail with the calculations:

$$P_1 = F(P_0), \quad P_2 = F(P_1), \quad P_3 = F(P_2), \quad \dots$$

- In general the transformation is applied to a specified point set, or to a set of primitives (e.g. lines, curves, surfaces)

Generating Fractals

- Relationships between the scaling factor s ; and the number of subparts n for subdivision of a unit straight-line segment, A square, and a cube



Fractal Dimension

- If take a line segment having length L and divide it into n pieces each piece having length ' l ', The scaling factor $s_1 = 1/n$

we get original segment back by scaling a segment by S

$$n = s^1 \quad \text{where } s = 1/s_1$$

- Similarly, a square is divided into four equal-area subparts by $s_1 = 0.5$

$$n = 4 \text{ and } s = 1/s_1 = 2 \text{ so } n = s^2 \text{ or } 4 = 2^2$$

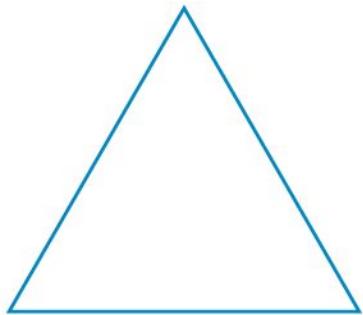
- A cube is divided into eight equal-volume subparts $n = s^3$
- Here, 1,2,3 are dimension of object , the relationship between the number of subparts and the scaling factor is $n = s^D$
- Solving this expression for D , the fractal similarity dimension, we have

$$D = (\log n) / (\log s)$$

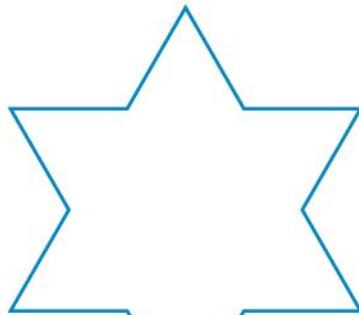
Generating Fractals (cont...)

- Although fractal objects, by definition have **infinite detail**, we only apply the transformation a **finite number of times**
- Obviously objects we display have finite dimension – they fit on a page or a screen
- A procedural representation **approaches a true representation** as we increase the number of iterations
- The **amount of detail is limited** by the resolution of the display device, but we can always *zoom in* for further detail

Example: The Koch Snowflake

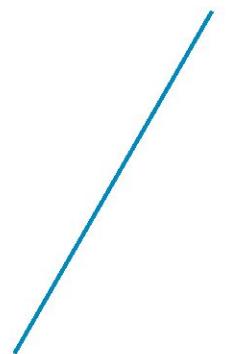


0



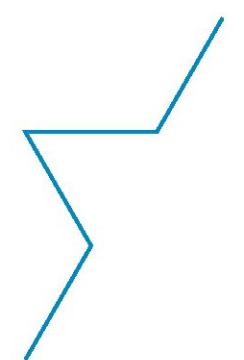
1

Segment Length = 1

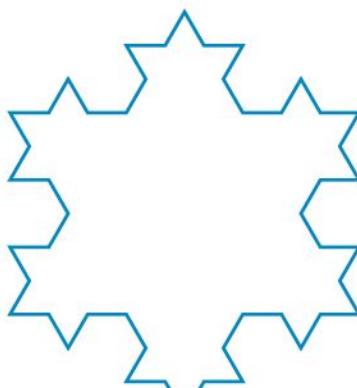


Length = 1

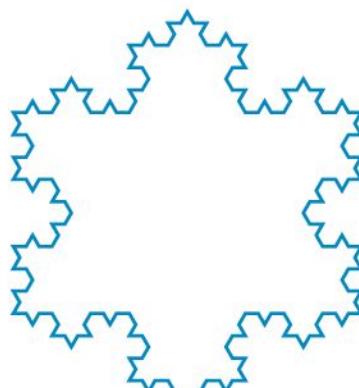
Segment Length = $\frac{1}{3}$



Length = $\frac{4}{3}$

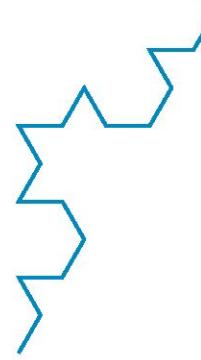


2



3

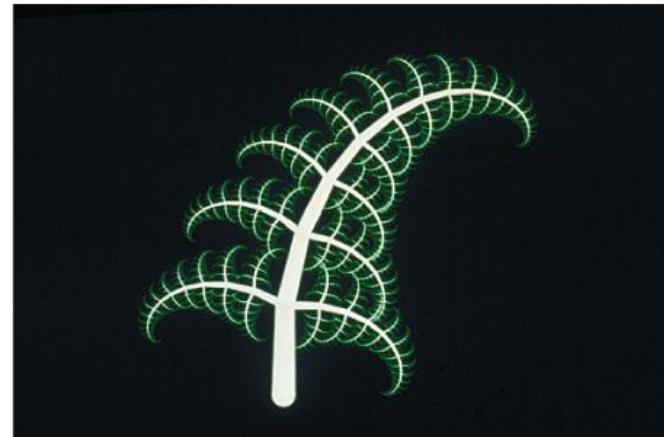
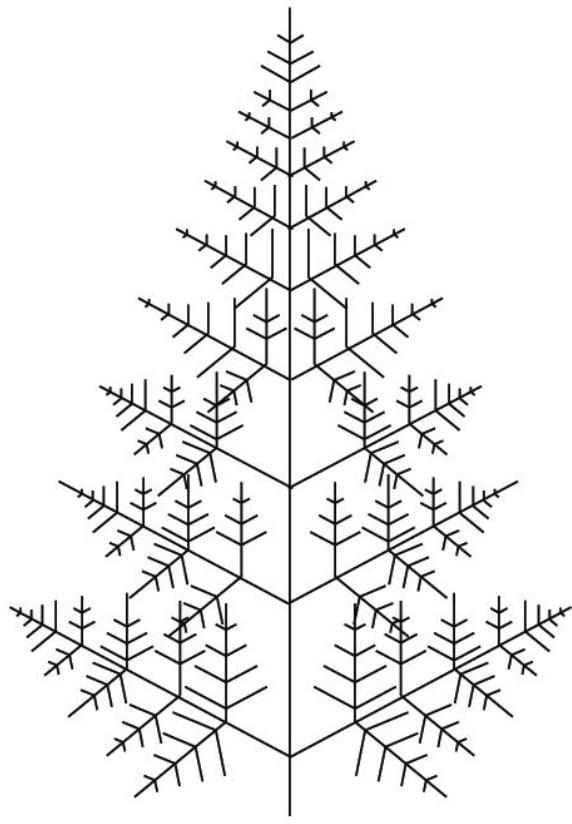
Segment Length = $\frac{1}{9}$



Length = $\frac{16}{9}$

Example: Ferns

- Very similar techniques can be used to generate vegetation



Fractal Dimension

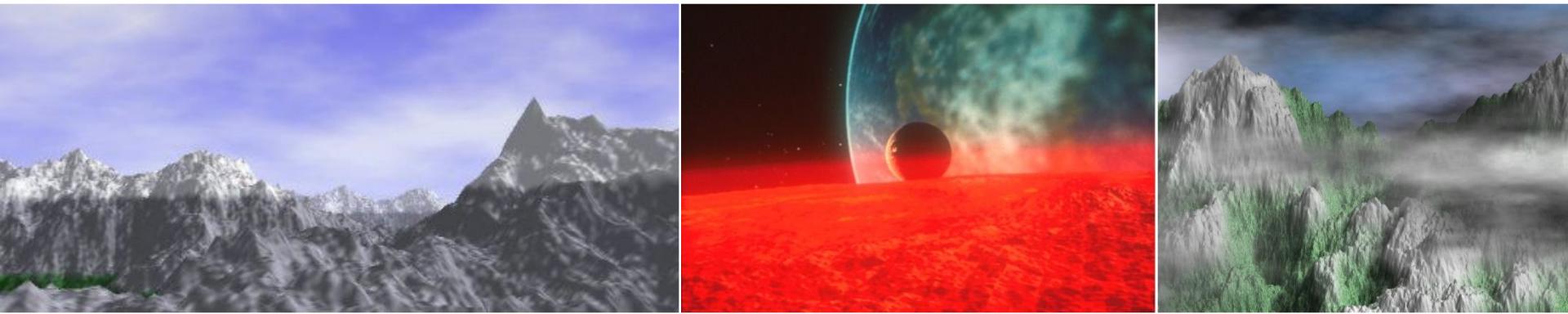
- The **amount of variation in the structure** of a fractal object is described as the **fractal dimension**, D
 - **More jagged looking objects** have larger fractal dimensions
- Calculating the fractal dimension can be difficult, especially for particularly complex fractals

Types Of Fractals

- Fractals can be classified into three groups
 - Self similar fractals
 - These have parts that are **scaled down versions of the entire object**
 - Commonly used to **model trees, shrubs** etc
 - Self affine fractals
 - Have parts that are formed with **different scaling parameters in each dimension**
 - Typically used for **terrain, water and clouds**
 - Invariant fractal sets
 - Fractals formed with **non-linear transformations**
 - Mandelbrot set, Julia set – generally not so useful

Random Midpoint Displacement Methods For Topography

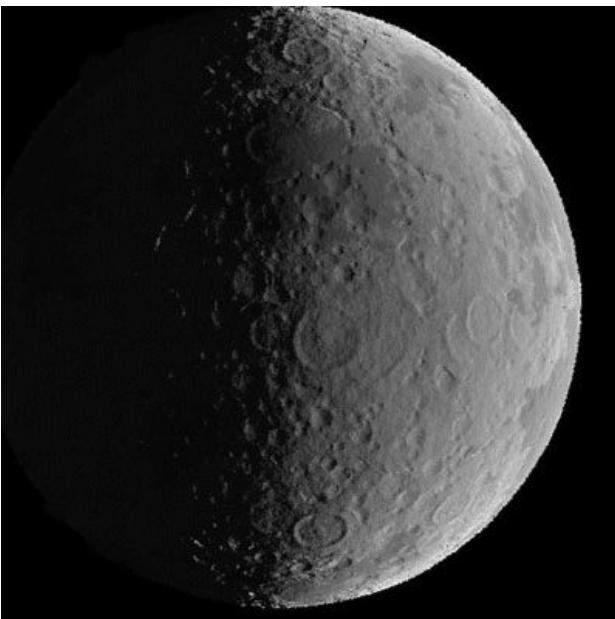
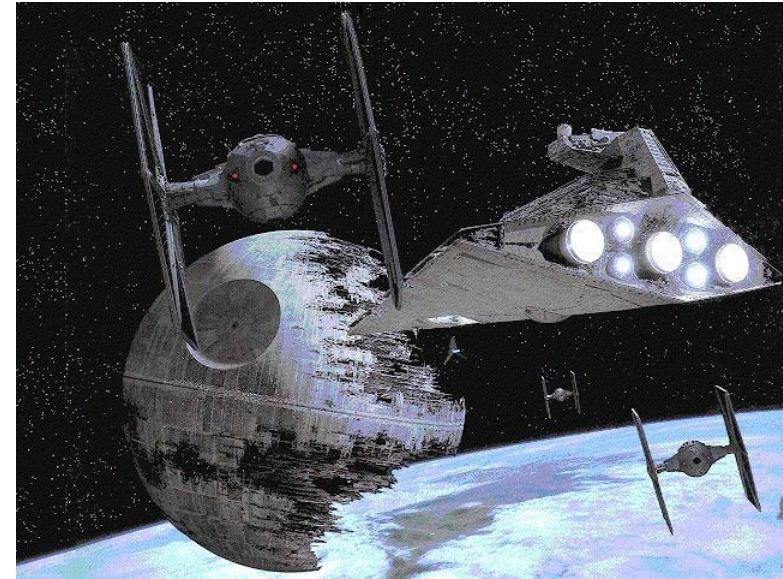
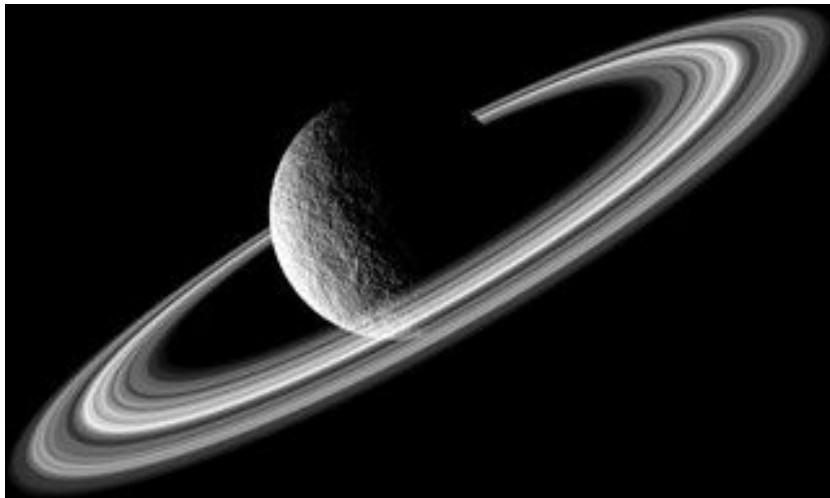
- One of the most successful uses of fractal techniques in graphics is the generation of landscapes
- One efficient method for doing this is **random midpoint displacement**



Random Midpoint Displacement Methods For Topography (cont...)

- Easy to do in two dimensions
- Easily expanded to three dimensions to generate terrain
- Can introduce a roughness factor H to control terrain appearance
- Control surfaces can be used to start with a general terrain shape

Fractals In Film Special Effects



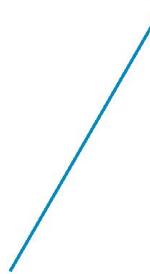
Geometric Construction of Deterministic Self-Similar Fractals

To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the *initiator*. Subparts of the initiator are then replaced with a pattern, called the generator.

Koch Curve

1. Begin with a line segment
2. Divide it into thirds i.e. scaling factor = $1/3$ and replace the center third by the two adjacent sides of an equilateral triangle
3. There are now 4 equal length segments each $1/3$ the original length , so the new curve has $4/3$ length of the original length

Segment Length = 1



Length = 1

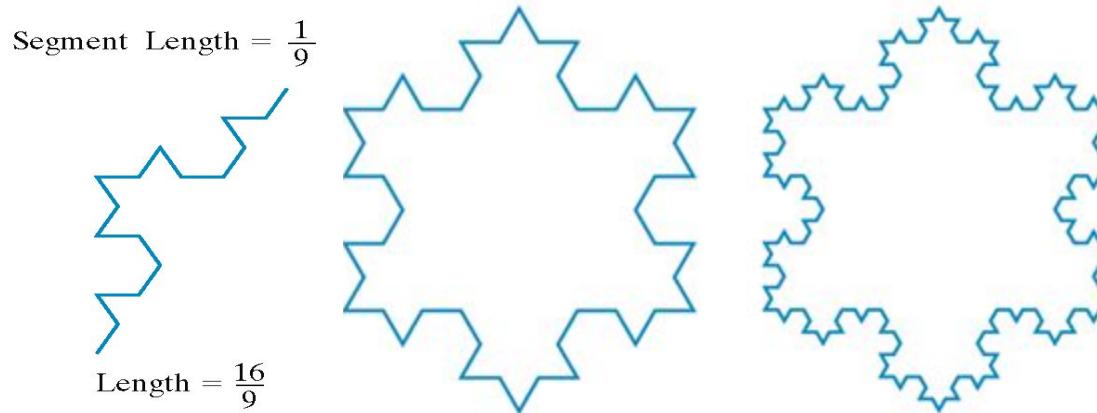
Segment Length = $\frac{1}{3}$



Length = $\frac{4}{3}$

Geometric Construction of Deterministic Self-Similar Fractals

5. Repeat the process for each of the four segments
6. The curve has gained more wiggles and its length now is $16/9$ times the original



7. Repeat this indefinitely and the length every time increases by $4/3$ factor. The curve will be infinite but is folded in lots of tiny wiggles
8. Its topological dimension is 1 and it's Fractal dimension can be calculated as follows

We have to assemble 4 such curves to make the original curve so $N = 4$ and Scaling factor $S = 3$ as each segment has $1/3$ the original segment length

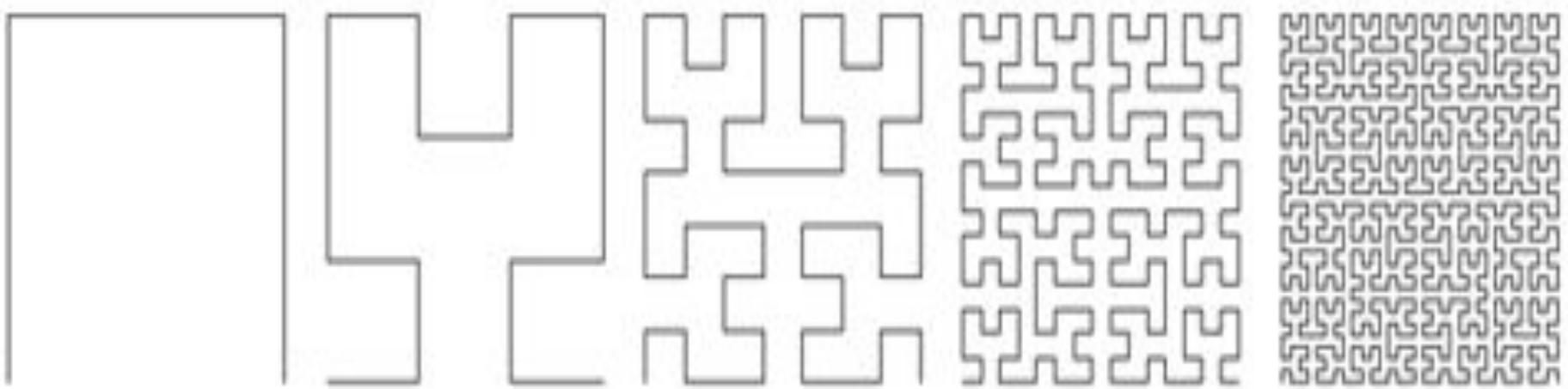
So the fractal dimension is $D = (\log 4) / (\log (3)) = 1.2618$

Geometric Construction of Deterministic Self-Similar Fractals

Peano Curve

It is also called space filling curve and is used for filling two dimensional object e.g. a square

Steps to generate a Peano curve



The Fractal dimension of the curve:

At each subdivision the scale changes by 2 but length changes by 4

For the square it takes 4 curves of half scale to build the full sized object so the Dimension is given by

$$D = (\log 4) / (\log 2)$$

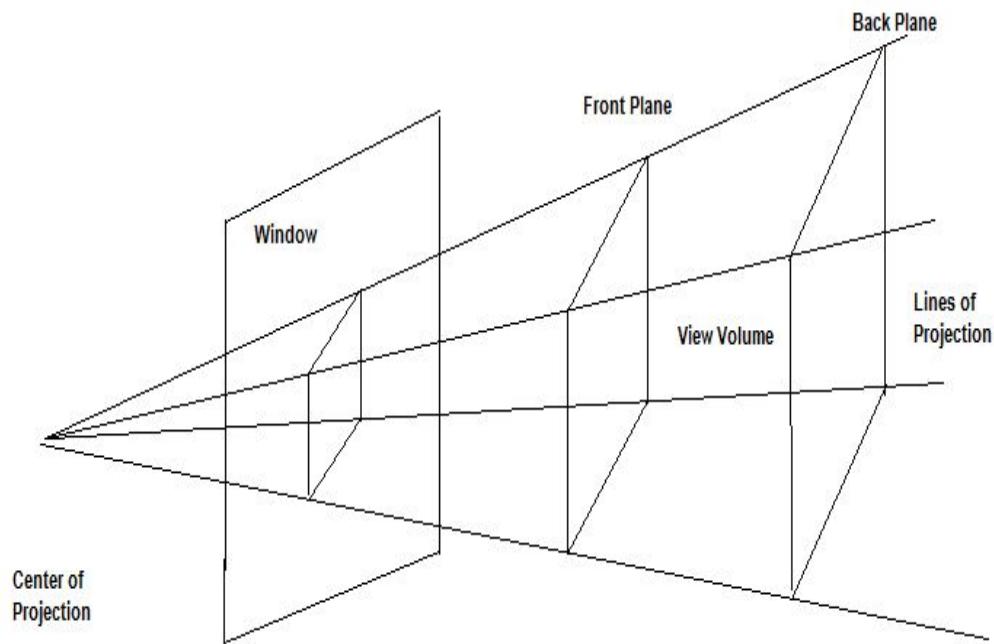
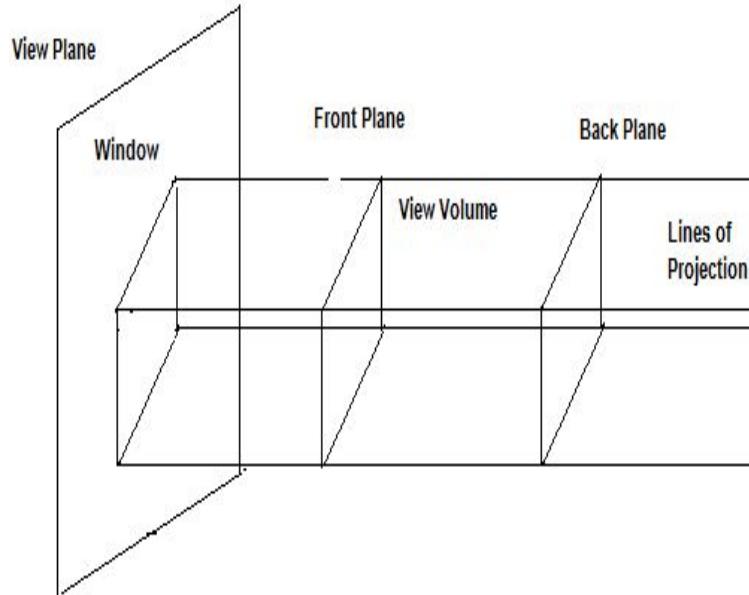
So, the Fractal Dimension is 2 and Topological Dimension is 1

3D Clipping

- **2D Clipping : window is considered** as a clipping boundary
- **3D : a view volume is considered**, which is a box between the two planes, the front and the back plane
- The part of the object which **lies inside the view volume will be displayed** and the part that **lies outside will be clipped**
- Cohen Sutherland 's region code approach **can be extended** for 3D clipping as well
- In 2D, a point is checked if it is inside the visible window/region or not but in 3D clipping a point is compared against a plane

3D Clipping

- For a parallel projection a box or a region is a **rectangular area** and in case of perspective projection it is a **truncated pyramidal volume** called a frustum of vision
- The view volume has **6 sides**: Left, Right , Bottom, Top, Near and Far



3D Clipping

- Here six bits are used to denote the region code
- The bits are set to 1 as per the following rule:
 - Bit 1 is set to 1 if $x < x_{v\min}$ Bit 2 is set to 1 if $x > x_{v\max}$
 - Bit 3 is set to 1 if $y < y_{v\min}$ Bit 4 is set to 1 if $y > y_{v\max}$
 - Bit 5 is set to 1 if $z < z_{v\min}$ Bit 6 is set to 1 if $z > z_{v\max}$
- If both the end points have region codes 000000 then the line is **completely visible**
- If the logical AND of the two end points region codes are not 000000 i.e. the same bit position of both the end points have the value 1, then the line is **completely rejected or invisible** else it is the case of **partial visibility** so the intersections with the planes must be computed

3D Clipping

- For a line with end points $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$, the parametric equation can be expressed as:

$$x = x_1 + (x_2 - x_1) u \quad y = y_1 + (y_2 - y_1) u \quad z = z_1 + (z_2 - z_1) u$$

- If we are testing a line against the front plane of the viewport then
 $z = z_{vmin}$ and $u = (z_{vmin} - z_1)/(z_2 - z_1)$
therefore

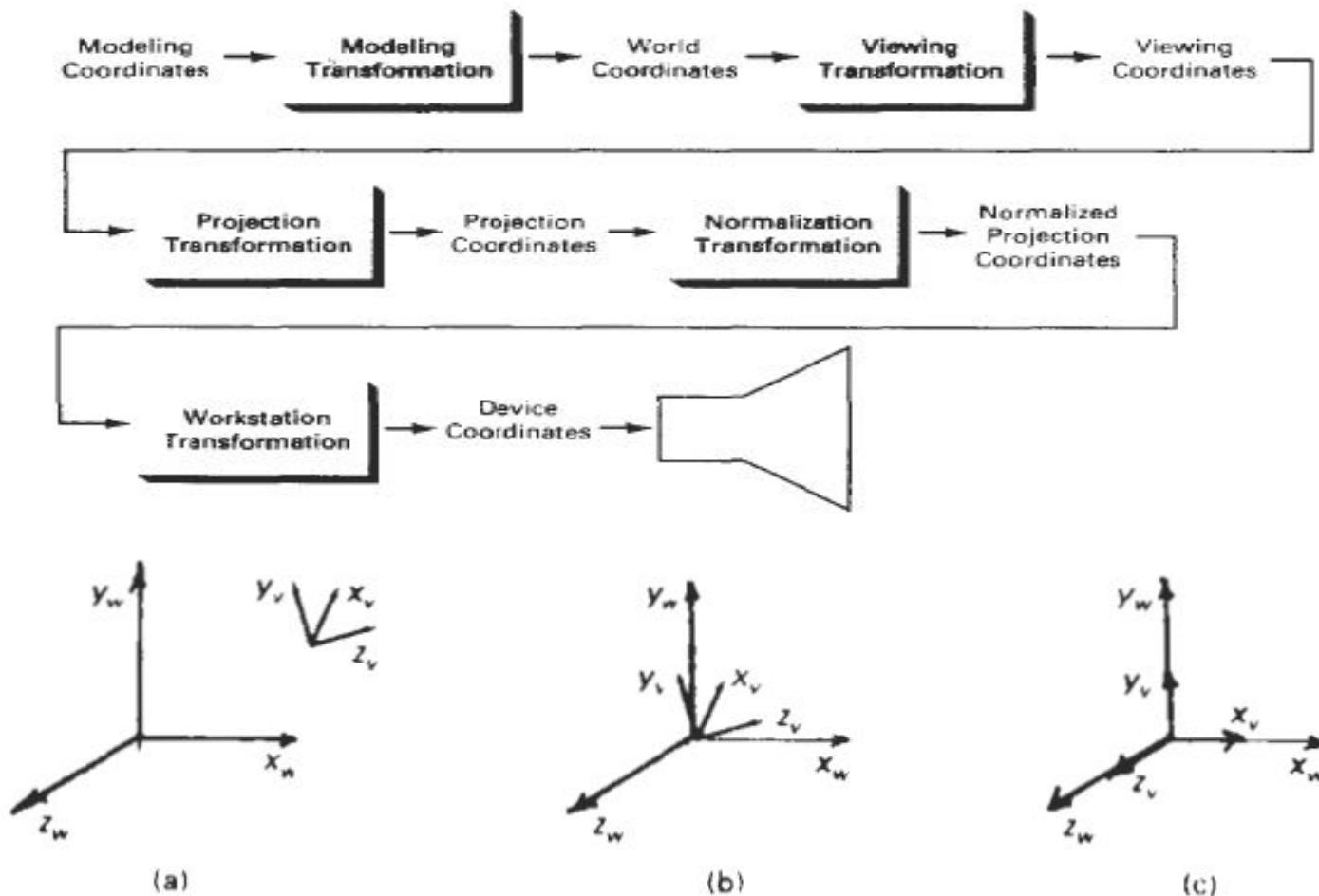
$$x_i = x_1 + (x_2 - x_1) \{(z_{vmin} - z_1)/(z_2 - z_1)\}$$

$$y_i = y_1 + (y_2 - y_1) \{(z_{vmin} - z_1)/(z_2 - z_1)\}$$

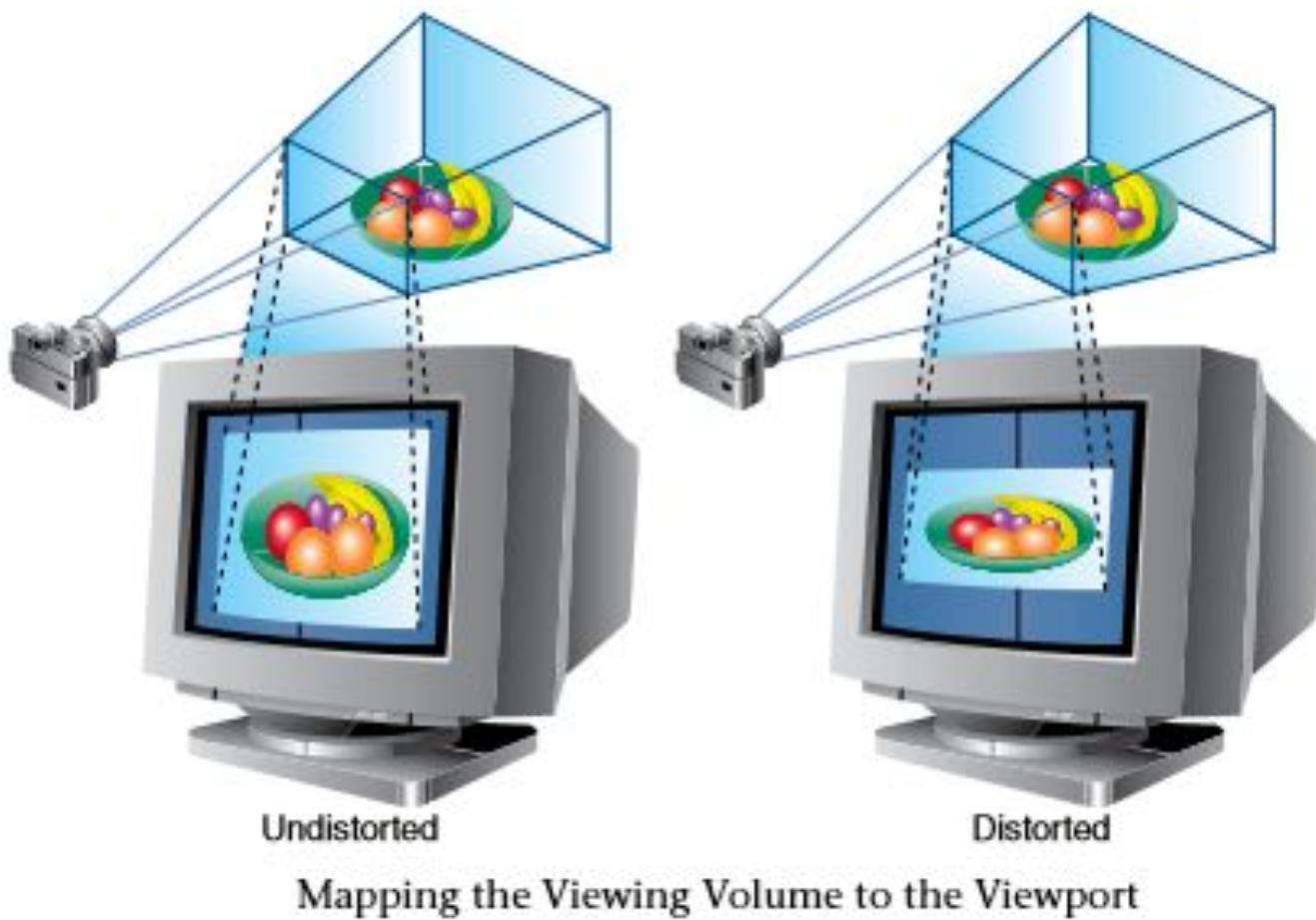
where x_i and y_i are the intersection points with the plane

3D Viewing Transformation

Mapping positions within a rectangular view volume to a three-dimensional rectangular viewport is accomplished with a combination of scaling and translation, similar to the operations needed for a two-dimensional window-to viewport mapping.

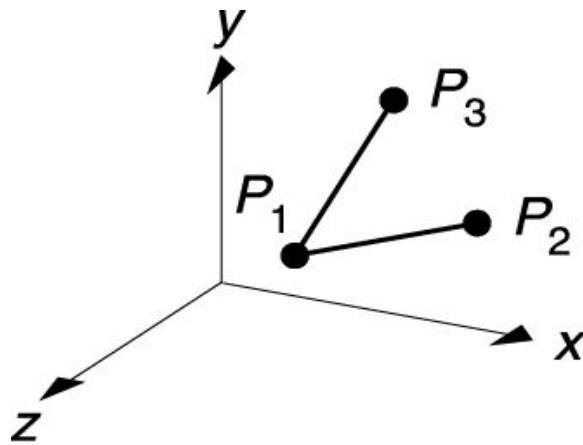


3D Viewing Pipeline

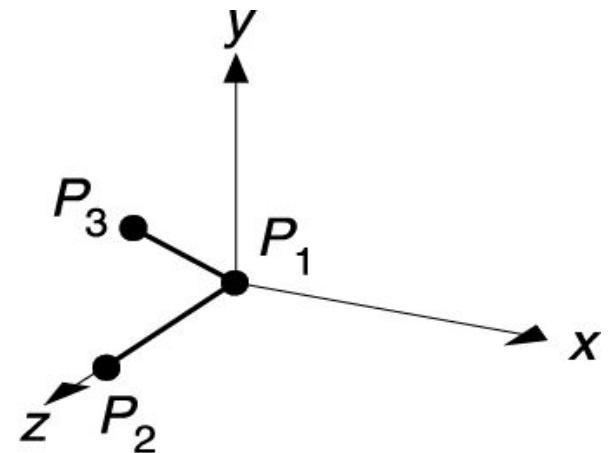


Example: Composition of 3D Transformations

- Goal: Transform P_1P_2 and P_1P_3



(a) Initial position

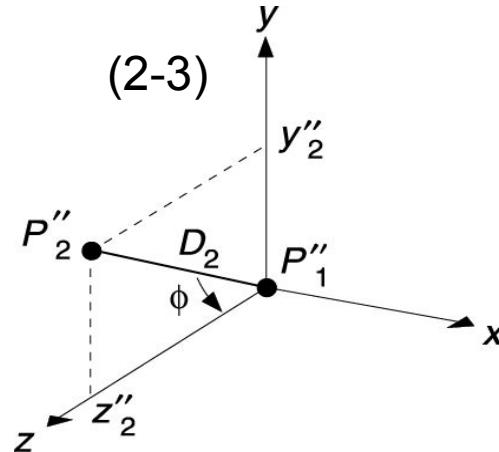
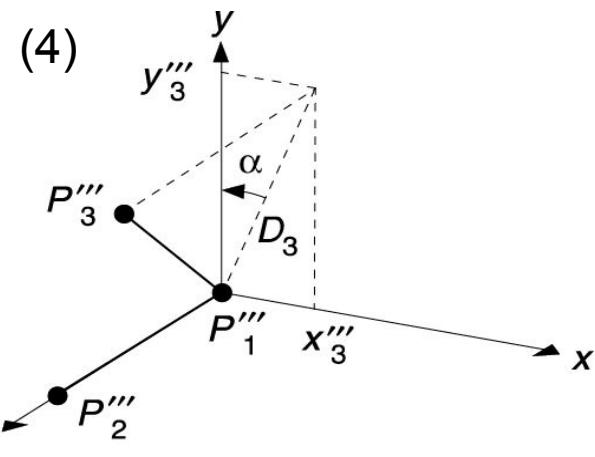
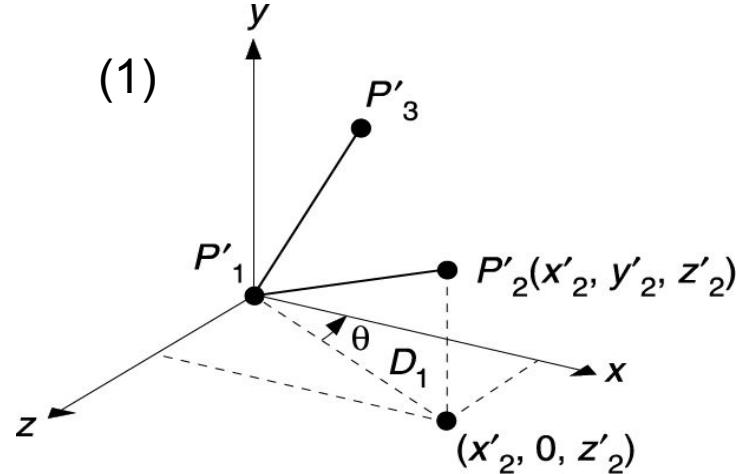


(b) Final position

Example (Cont.)

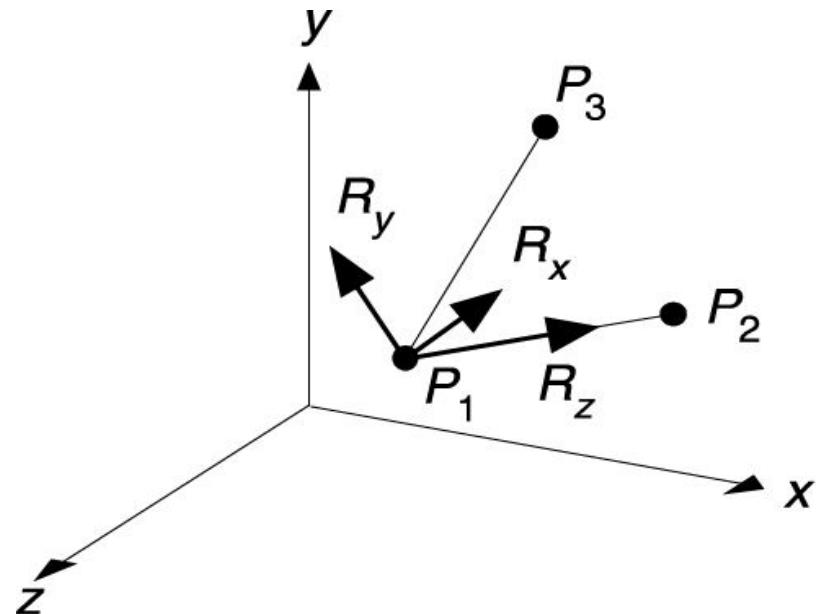
- Process

1. Translate P_1 to $(0,0,0)$
2. Rotate about y
3. Rotate about x
4. Rotate about z



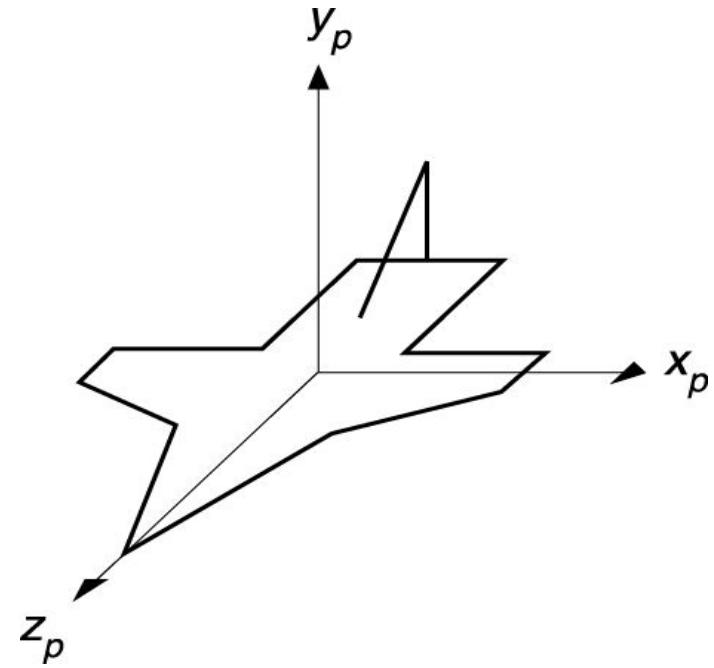
Final Result

- What we've really done is transform the local coordinate system R_x, R_y, R_z to align with the origin x, y, z



Example 2: Composition of 3D Transformations

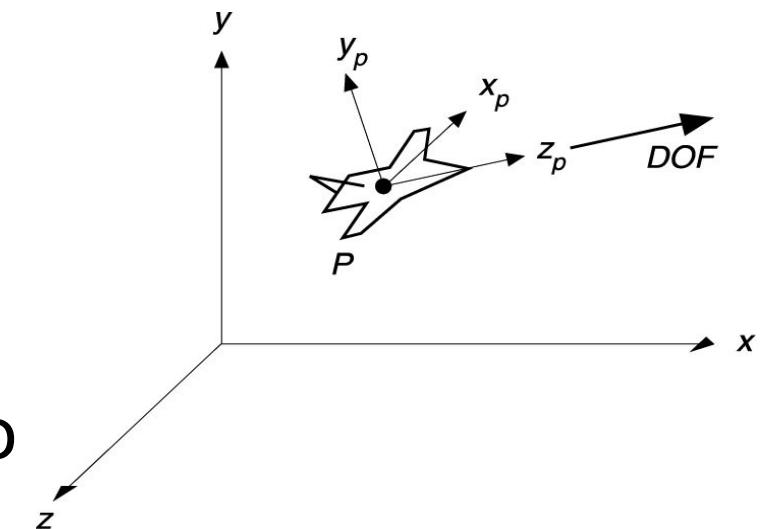
- Airplane defined in x,y,z
- Problem: want to point it in Dir of Flight (DOF) centered at point P
- Note: DOF is a vector
- Process:
 - Rotate plane
 - Move to P



Example 2 (cont.)

- Z_p axis to be DOF
- X_p axis to be a horizontal vector perpendicular to DOF
 - $y \times DOF$
- Y_p , vector perpendicular to both Z_p and X_p (i.e. $Z_p \times X_p$)

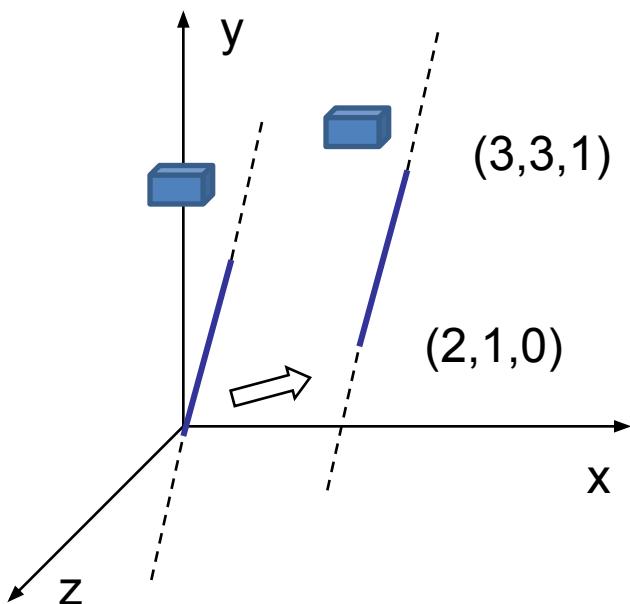
$$R = \begin{bmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



How will you rotate a unit cube 90° about an axis defined by its endpoints A(2,1,0) and B(3,3,1).

Step 7

Translate end point back to (2, 1, 0)



$$T = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about an Arbitrary Plane

What do you mean by affine transformation? How would you reflect an object with respect to a plane passing thru point P(1,2,3) and having a normal vector N=I+J+K?

Here,

- 1.Translate the point by T(-1,-2,-3) to make the plane pass thru origin
- 2.Then align the normal with the z axis for that rotate the normal two times first about x axis in ccw , then about y axis in cw direction then
- 3.Take reflection of object about xy plane
- 4.Now perform inverse rotations about y then x axis
- 5.Then retranslate the plane along with the reflected object

Reflection about an Arbitrary Plane

Hints:

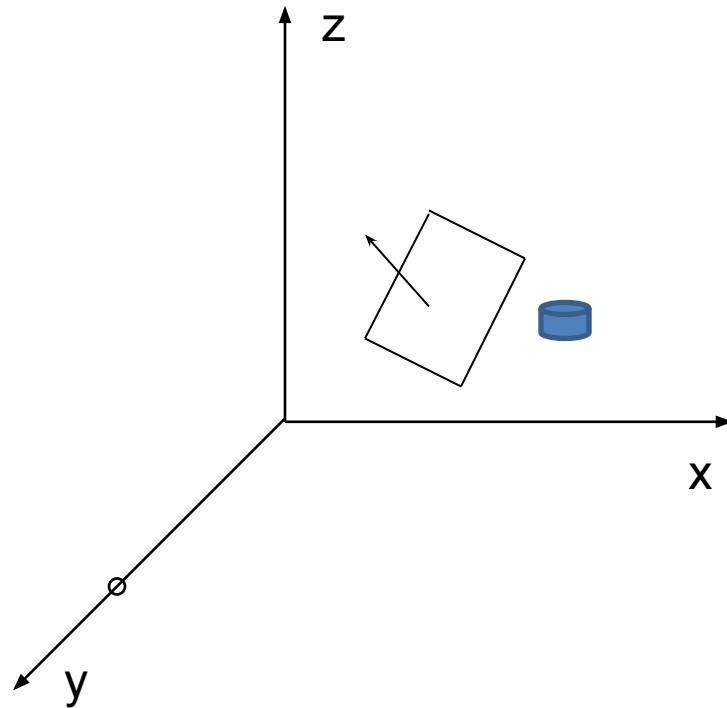
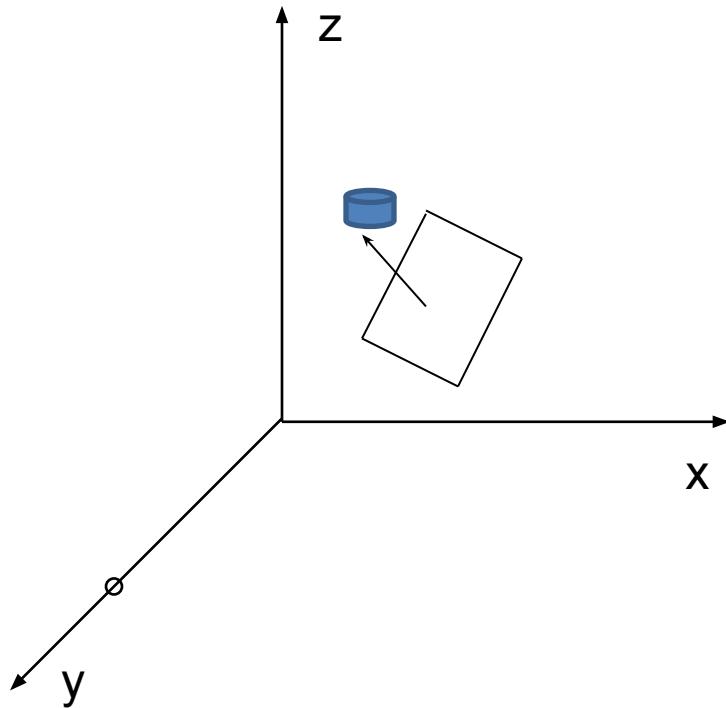
Spatial orientation of a plane in 3D space is given by its outward normal

Translate plane so that it passes thru origin

Rotate the Normal two times so that it gets aligned with z axis and the plane lands on xy plane

Take reflection of object about that plane

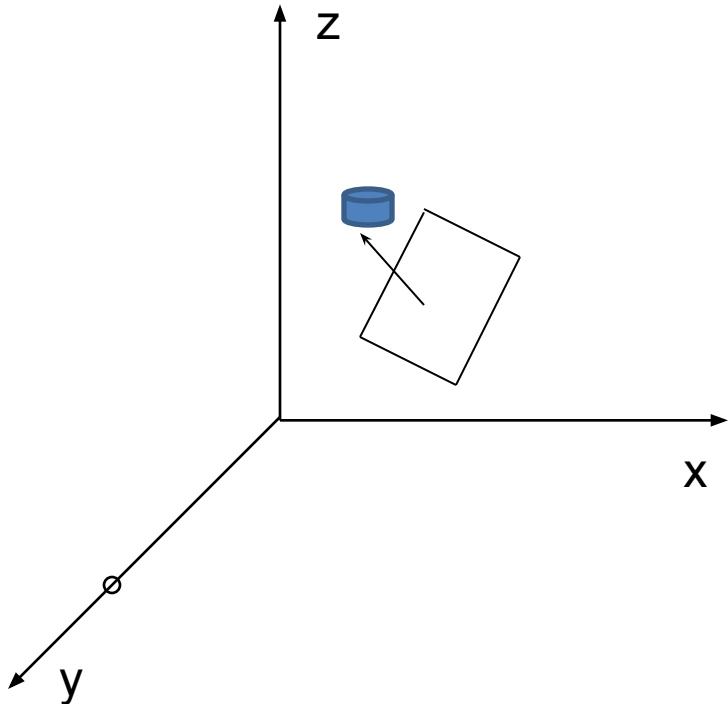
Perform inverse rotations and translation



Reflection about an Arbitrary Plane

Step 1

Translate the plane along with the object to be reflected about it so that the plane passes thru origin

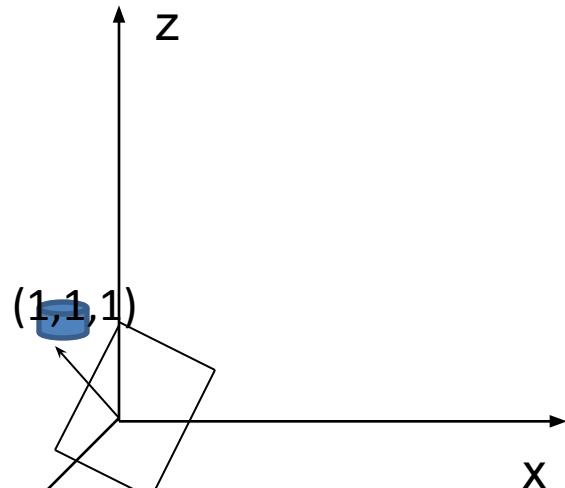


$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

Step 2

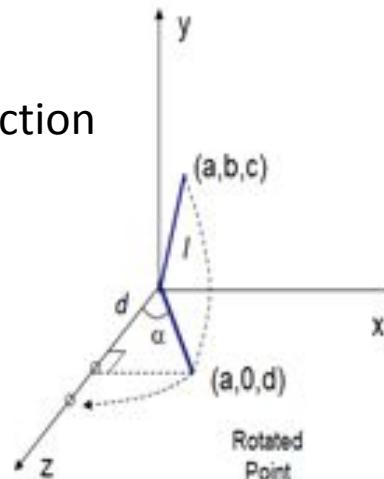
Rotate the normal about x then y axis so that the normal is aligned with z axis and the plane is aligned with xy plane



$$\left[R_{x_\theta} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left[R_{y_\alpha} \right] = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

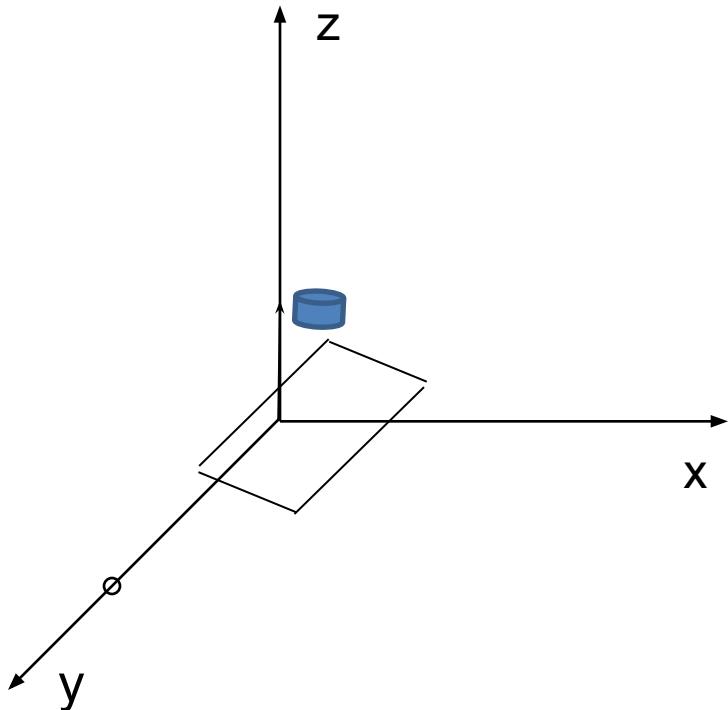
Here, $n = i + j + k$ so $(a,b,c) = (1,1,1)$ its projection is $(0,b,c) = (0,1,1)$ in this case
So length of axis is $\sqrt{3}$ and length of shadow is $\sqrt{2}$



Reflection about an Arbitrary Plane

Step 3

Reflect the object about the xy plane with which the plane has been aligned

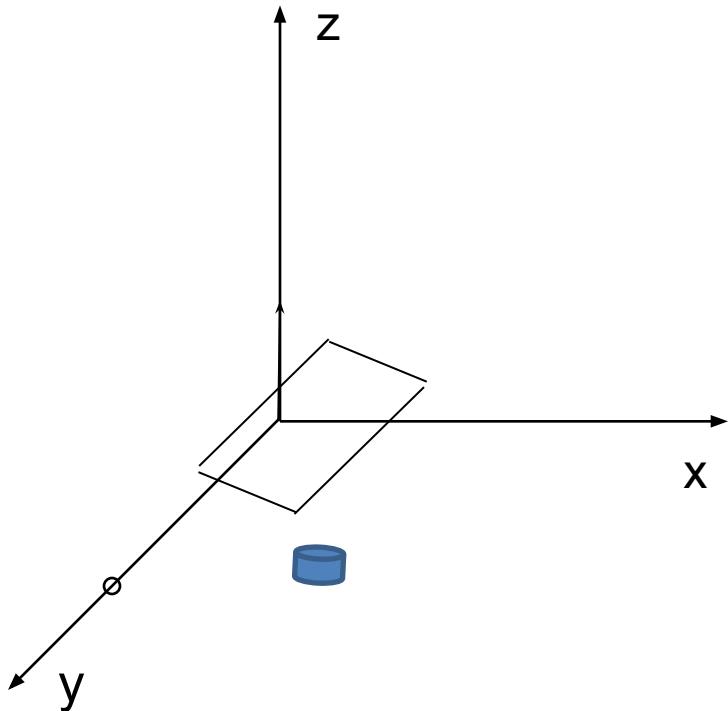


$$[Rf_{xy}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

Step 3

Reflect the object about the xy plane with which the plane has been aligned

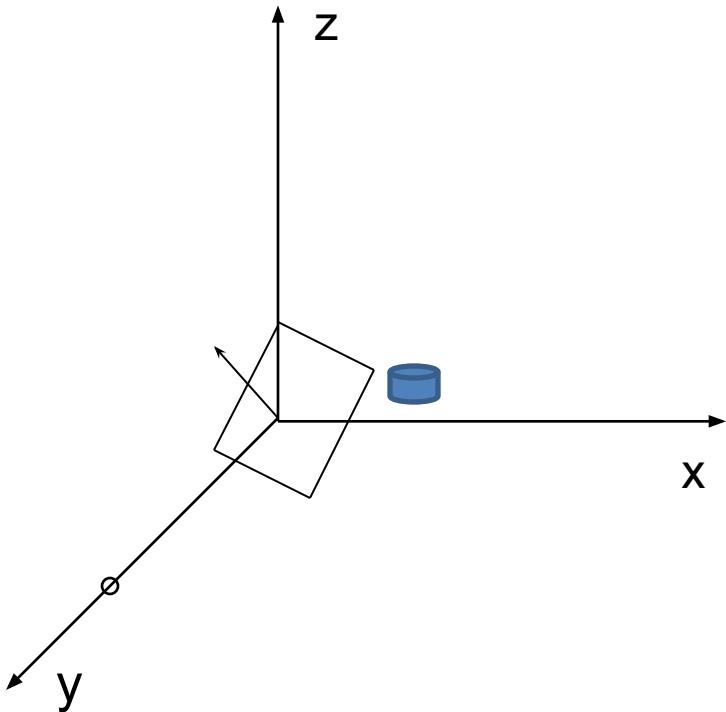


$$[Rf_{xy}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

Step 4

Rotate the normal about y then x axis so that the normal is **de-aligned** from z axis and the plane is de-aligned with xy plane



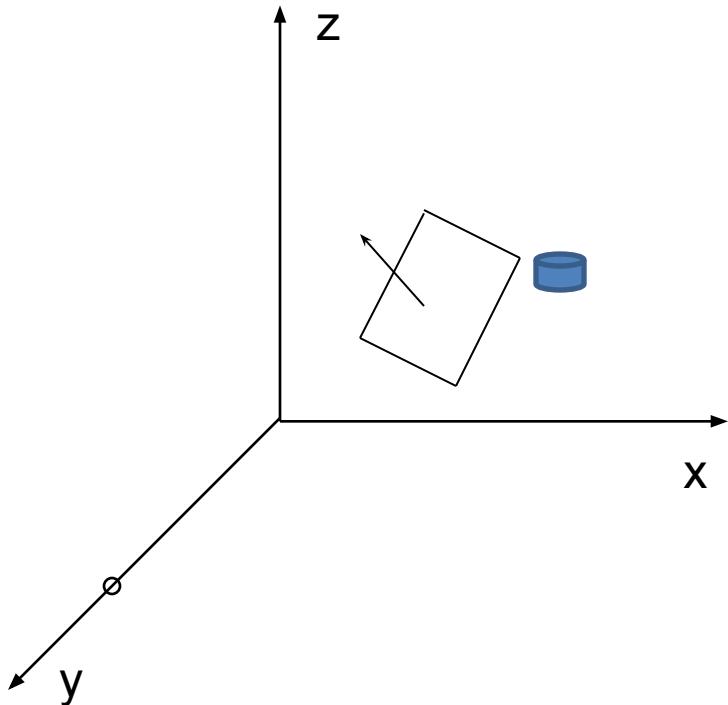
$$\left[R'_{y_\alpha} \right] = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\left[R'_{x_\theta} \right] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & \sin \theta & 0 \\ 0 & -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

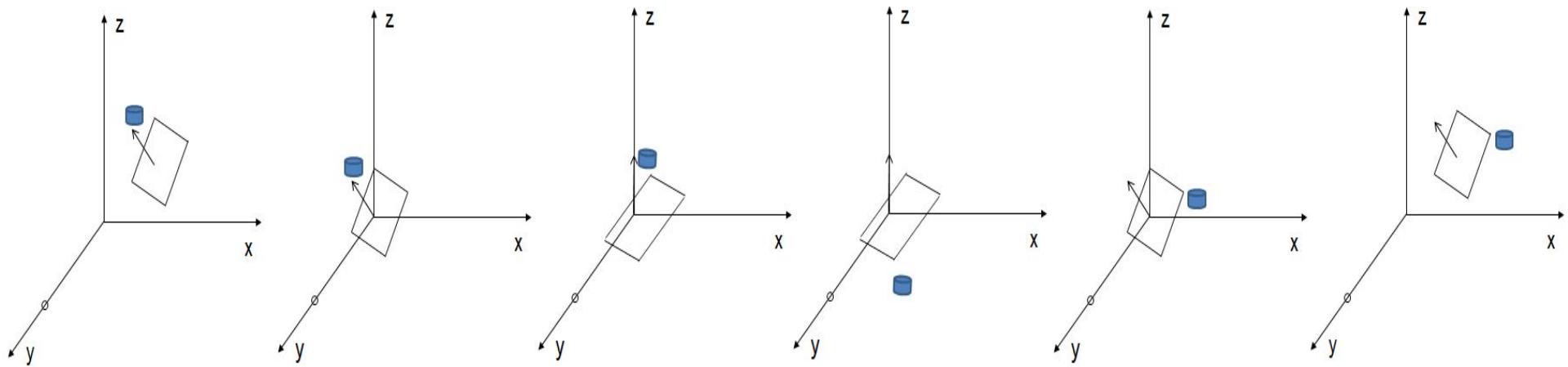
Step 5

Translate the plane along with the **reflected object** back to its original location



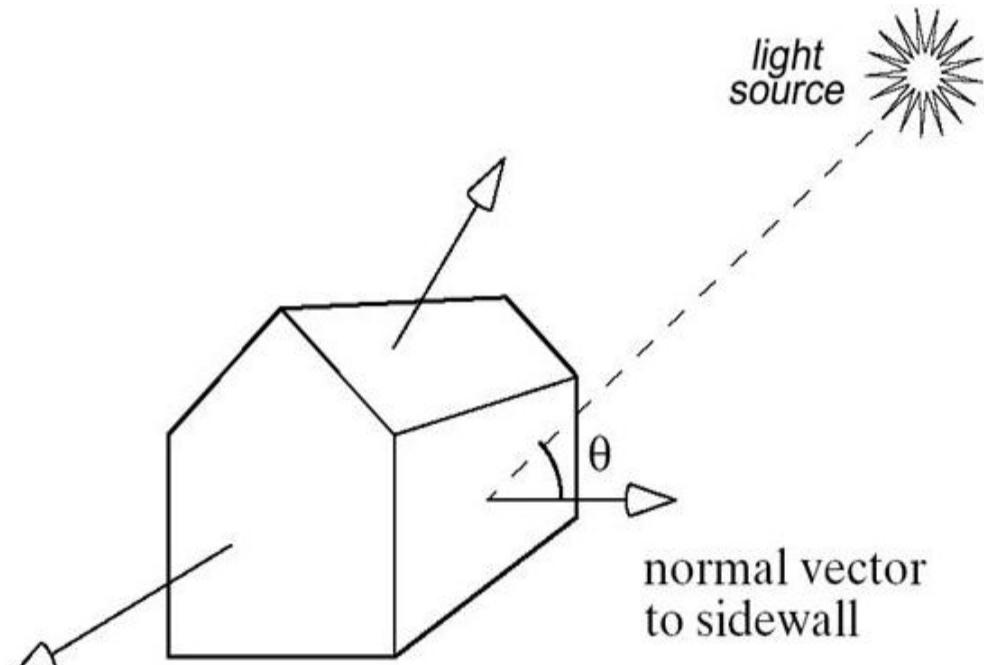
$$T' = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Reflection about an Arbitrary Plane

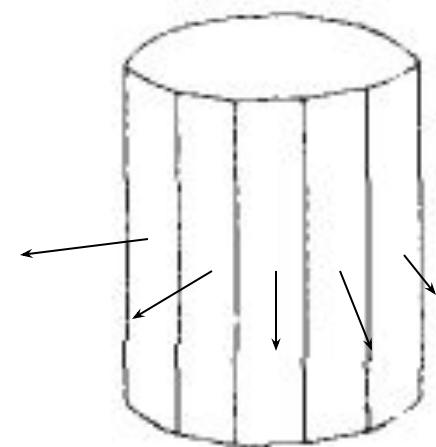
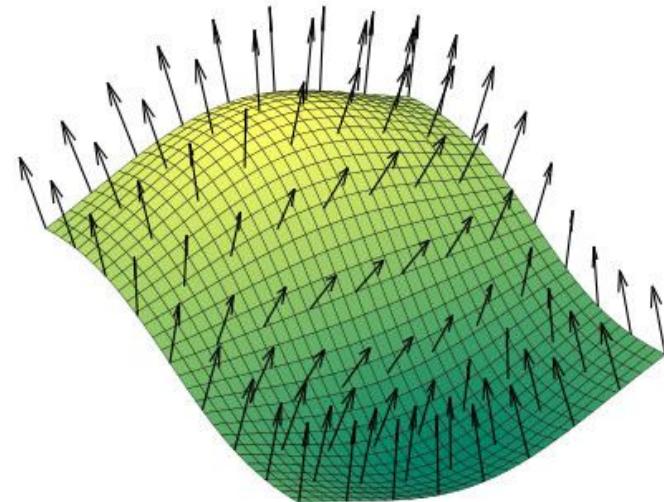


$$CM = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Object Representation



normal vector
to front wall



3D Object Representation

Graphics scenes can contain trees, flowers , clouds rocks water, rubber, paper , bricks etc.

Polygon and quadratic surfaces provide precise descriptions for simple Euclidean objects such as polyhedrons ellipsoid.

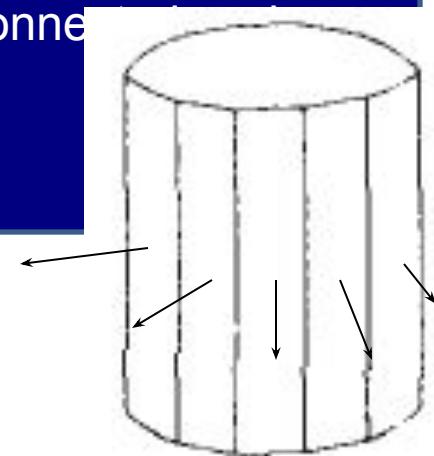
Polygon Surfaces

3-D graphics object is a set of surface polygons that enclose the object interior

A polygon mesh is a set of connected polygonally bounded planar surfaces

Equation of plane is $Ax + By + Cz + D = 0$

A polygon mesh is a collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons.



Polygon Tables

Polygon data tables can be organized into two groups:

- i.geometrical table
- ii.attribute table

Geometric Data Tables

- contain **vertex coordinates** and parameters to identify the spatial orientation of polygon surfaces

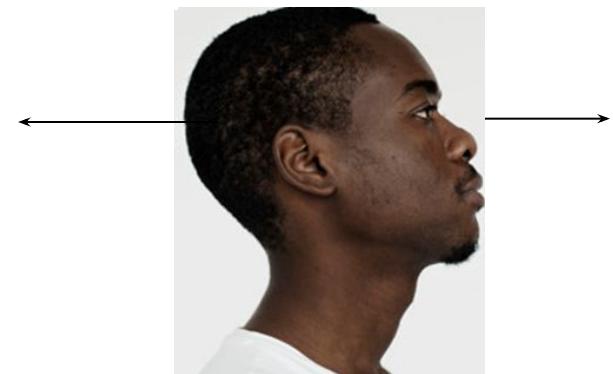
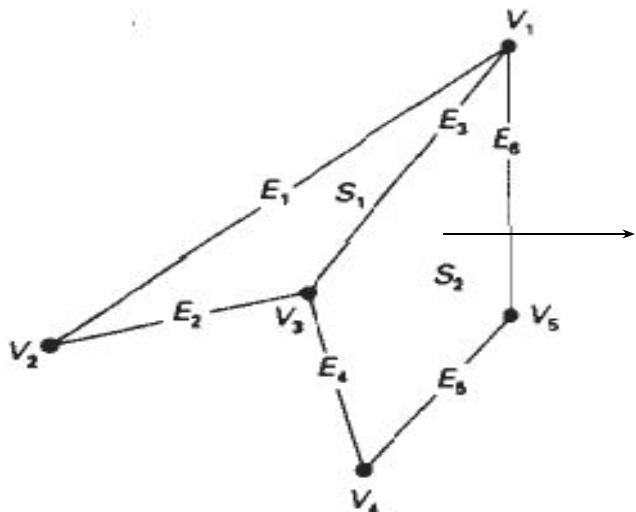
Attribute Table

- includes parameters specifying the degree of **transparency** of object and its **surface reflectivity** and **texture** characteristics.

3D Object Representation

A convenient organization for storing geometric data is to create three lists:

- i. a vertex table
- ii. an edge table
- iii. a polygon table



VERTEX TABLE
V ₁ : x ₁ , y ₁ , z ₁
V ₂ : x ₂ , y ₂ , z ₂
V ₃ : x ₃ , y ₃ , z ₃
V ₄ : x ₄ , y ₄ , z ₄
V ₅ : x ₅ , y ₅ , z ₅

EDGE TABLE
E ₁ : V ₁ , V ₂
E ₂ : V ₂ , V ₃
E ₃ : V ₃ , V ₁
E ₄ : V ₃ , V ₄
E ₅ : V ₄ , V ₅
E ₆ : V ₅ , V ₁

POLYGON-SURFACE TABLE
S ₁ : E ₁ , E ₂ , E ₃
S ₂ : E ₃ , E ₄ , E ₅ , E ₆

Plane Equations

The equation for a plane surface can be expressed as

$$Ax + By + Cz + D = 0 \quad \dots\dots(i)$$

where, (x,y,z) is any point on the plane

Coefficients ABCD are constants describing spatial properties of plane

For solving ABCD consider three successive polygon vertices (x_1, y_1, z_1) ,
 (x_2, y_2, z_2) , (x_3, y_3, z_3)

So equation (i) is modified to,

$$\frac{A}{D}x_k + \frac{B}{D}y_k + \frac{C}{D}z_k = -1 \quad \dots\dots(ii) \qquad k = 1, 2, 3$$

For solving ABCD consider three successive polygon vertices

$(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$

So equation (i) is modified to,

$$\frac{A}{D} x_k + \frac{B}{D} y_k + \frac{C}{D} z_k = -1 \quad \dots\dots \text{(ii)} \qquad k = 1, 2, 3$$

The solution for this set of equations can be obtained in determinant form, using Cramer's rule, as

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = - \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding the determinants, we can write the calculations for the plane coefficients in the form

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

$$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$$

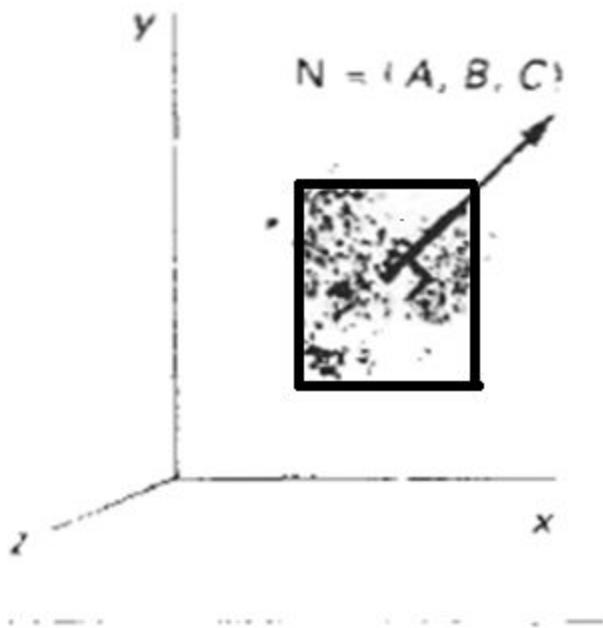
$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$$

As vertex values and other information are entered into the polygon data structure, values for A, B, C, and D are computed for each polygon and stored with the other polygon data.

Orientation of a plane surface in space can be described with the normal vector to the plane

This surface normal vector has Cartesian components (A, B, C), where **parameters A, B, and C are the plane coefficients** (describing spatial properties of plane) calculated



The vector \mathbf{N} , normal to the surface of a plane described by the equation $Ax + By + Cz + D = 0$, has Cartesian components (A, B, C)

3D Object Representation

Plane equations are used to identify the position of spatial points relative to the plane surfaces of an object. For any point (x,y,z) not on plane with parameters ABCD we have

$$Ax + By + Cz + D = 0$$

We can identify the point as either inside or outside the plane surface according to the sign(+ or -) of

$$Ax + By + Cz + D$$

If $Ax + By + Cz + D < 0$ then the point (x,y,z) is inside the surface

If $Ax + By + Cz + D > 0$ then the point (x,y,z) is outside the surface

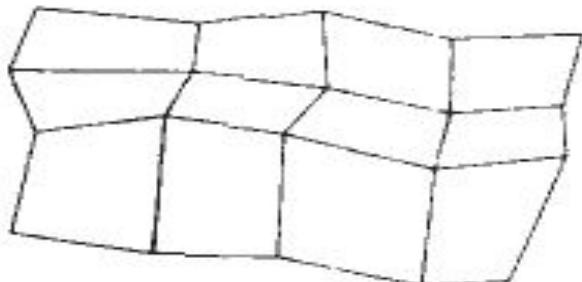
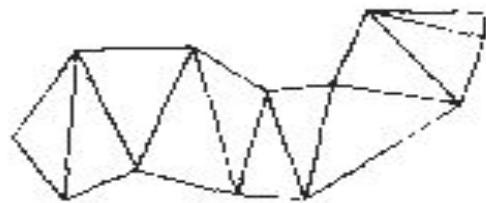
Polygon Meshes

Some graphics packages (PHIGS programmer's Hierarchical Interactive Graphics Standard) provide several polygon functions for modeling objects.

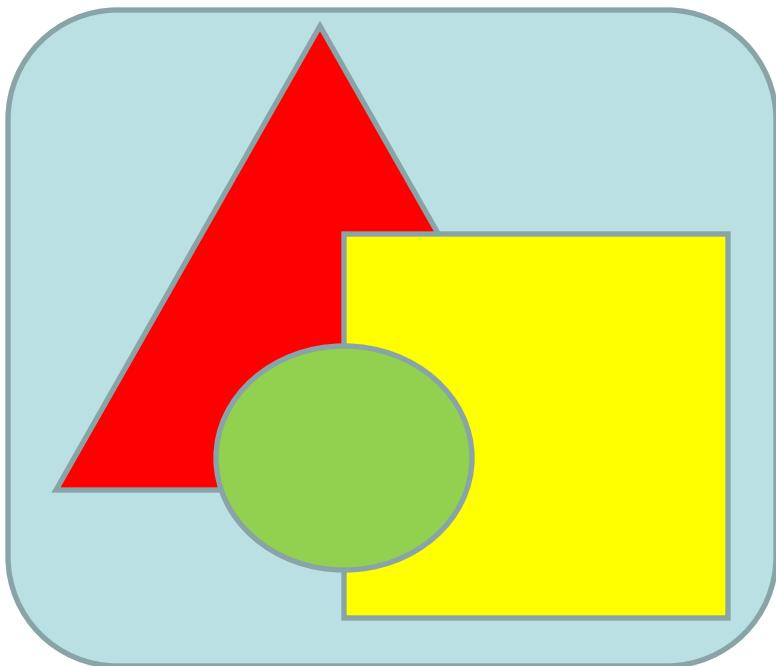
One type of polygon mesh is the triangle strip

It produces $n - 2$ connected triangles, given the coordinates for n vertices.

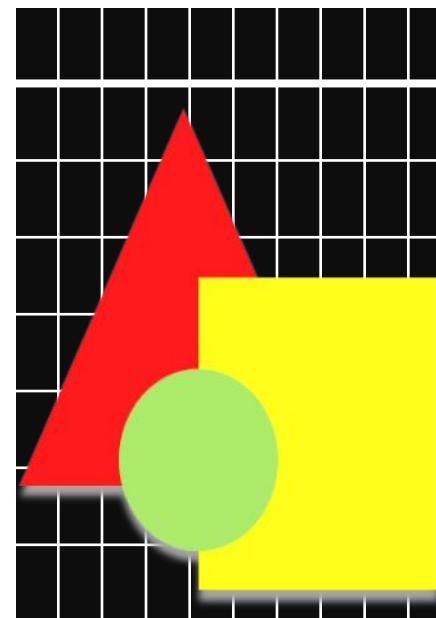
Another similar function generates a quadrilateral mesh that generates a mesh of $(n-1)(m-1)$ quadrilaterals, given the coordinates for an n by m array of vertices



Hidden Surface Removal (Visible Surface Detection)

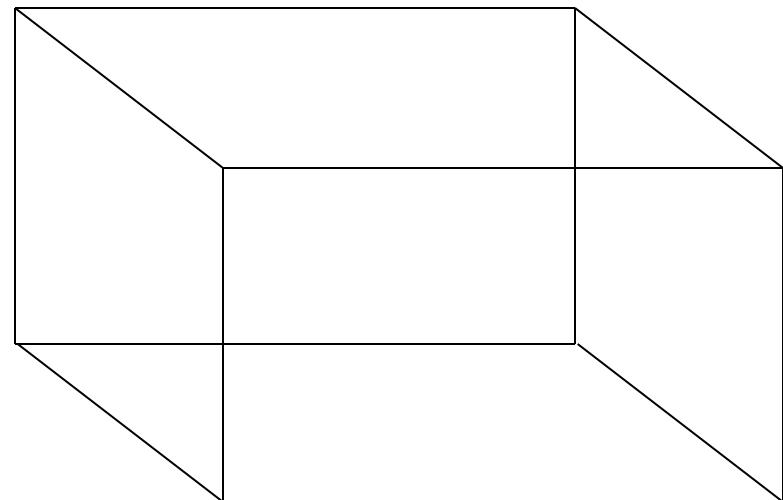
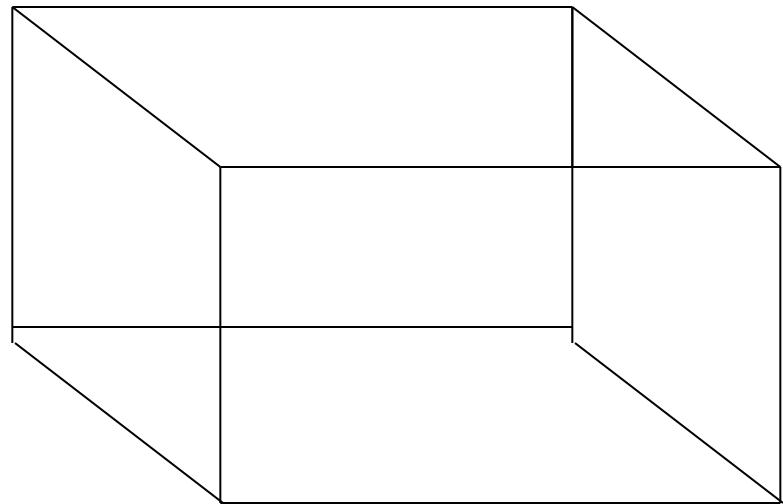
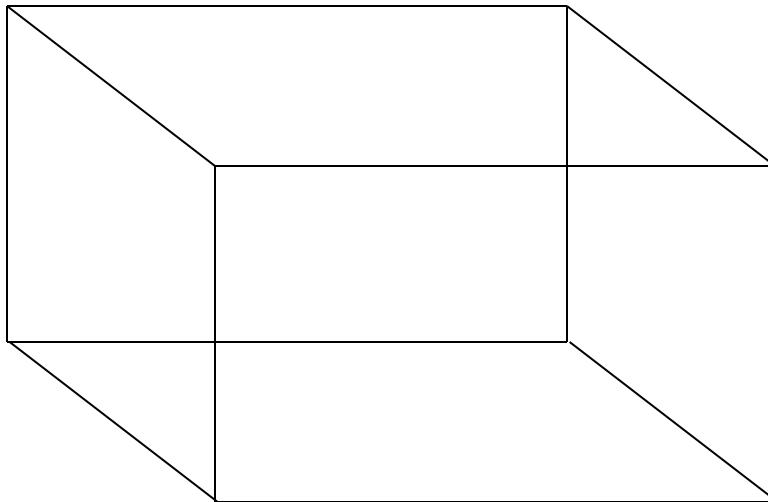


Screen



FrameBuffer

Hidden Surface Removal



Hidden Surface Removal (Visible Surface Detection)

Major concern for realistic graphics is identifying those parts of a scene that are visible from a chosen viewing position

Object Space Methods (OSM)

- OSM compares objects and parts of object to each other within the scene definition to determine which surfaces as a whole we should label as visible e.g. *Back Face Detection method*

Image Space Method (ISM)

- ISM decides point by point at each pixel position and is most widely used.

Hidden Surface Removal (Visible Surface Detection)

Object Space Methods (OSM)

- *Back Face Detection method*

Image Space Method (ISM)

- *Z Buffer*

- *A Buffer*

- *Scanline*

Hybrid (Image/Object)

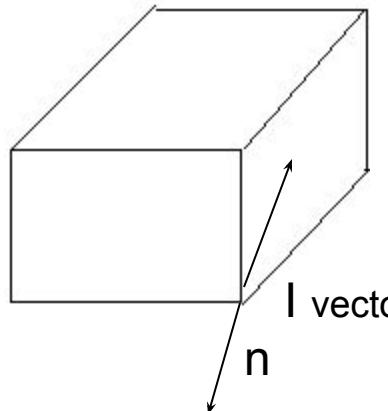
- Depth Sorting

Area Subdivision Method

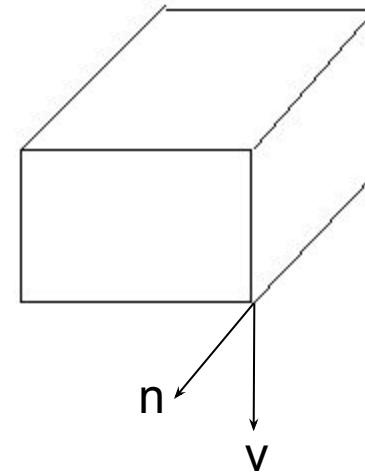
Backface Detection

V , a view vector is constructed from any point on the surface to the viewpoint, the dot product of this vector and the normal n , indicates visible faces as follows:

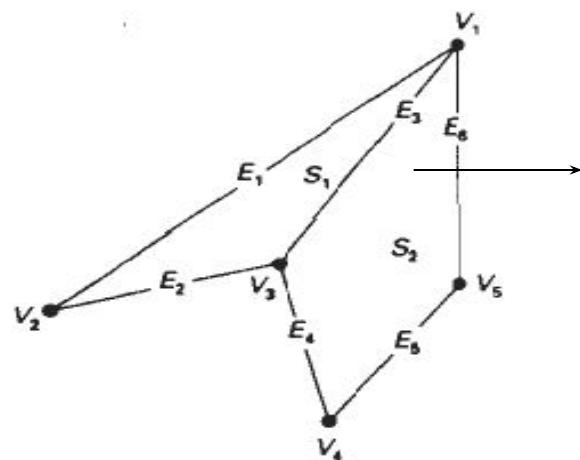
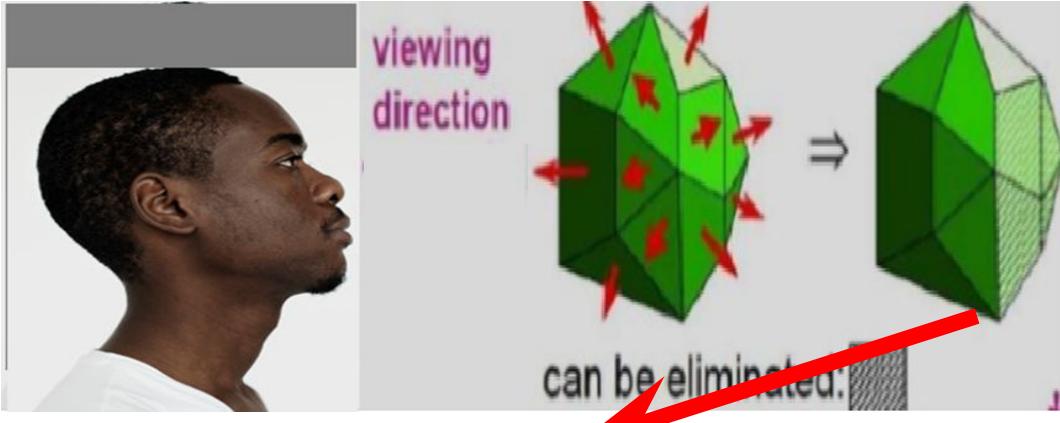
If $v \cdot n > 0$ the face is visible else face is hidden



v vector directed to a point inside object



Backface Detection



VERTEX TABLE
$V_1: x_1, y_1, z_1$
$V_2: x_2, y_2, z_2$
$V_3: x_3, y_3, z_3$
$V_4: x_4, y_4, z_4$
$V_5: x_5, y_5, z_5$

EDGE TABLE
$E_1: V_1, V_2$
$E_2: V_2, V_3$
$E_3: V_3, V_1$
$E_4: V_3, V_4$
$E_5: V_4, V_5$
$E_6: V_5, V_1$

POLYGON-SURFACE TABLE
$S_1: E_1, E_2, E_3$
$S_2: E_3, E_4, E_5, E_6$

Z-buffer or Depth buffer Method:

Commonly used *ISM*

Since object depth is usually measured from the view plane along the z axis of a viewing surface, each surface of a scene is processed separately , one point at a time across the surface

Usually for polygon surface, because depth values can be computed very quickly and the method is easy to implement and can apply to non planar surfaces.

Implementing the algorithm in normalized coordinates , z value ranges from 0 at the back clipping plane to z_{\max} at the front clipping plane.

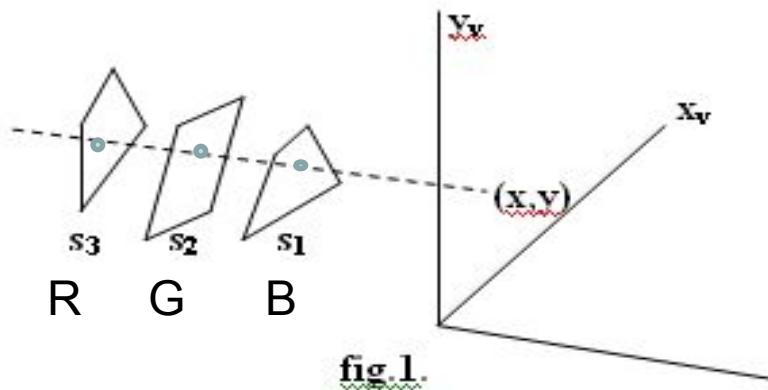
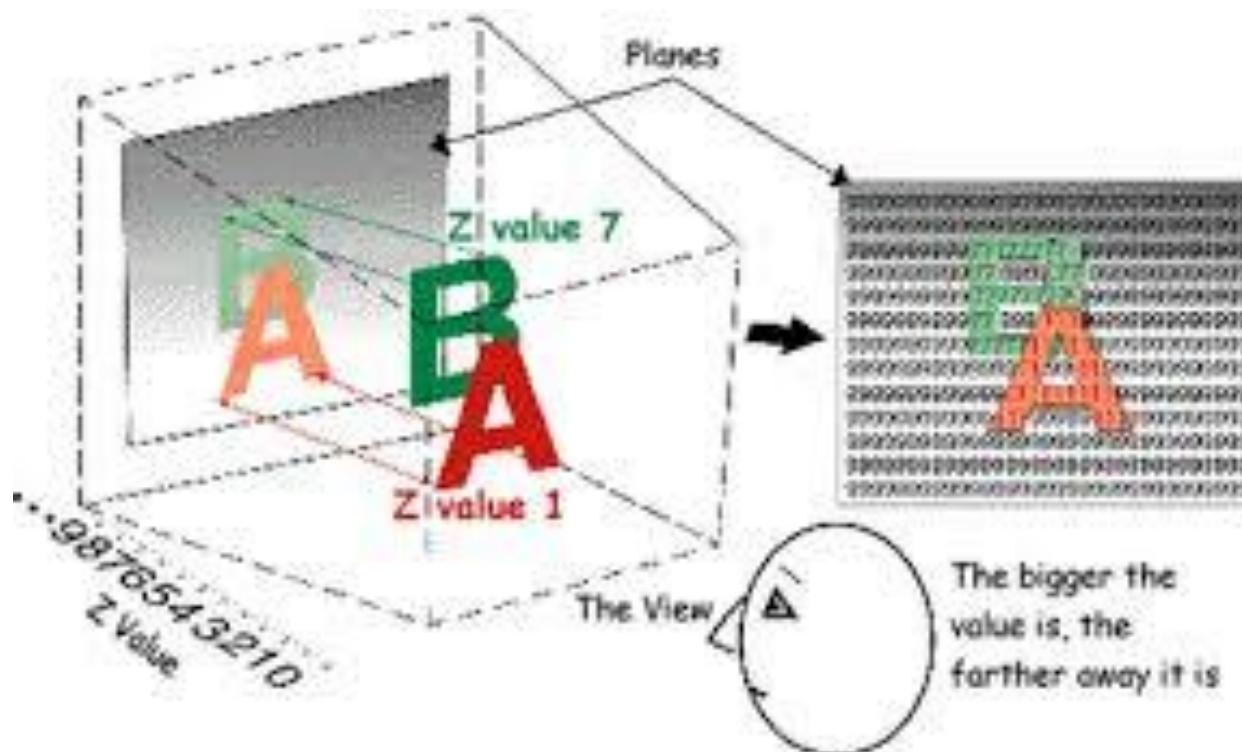


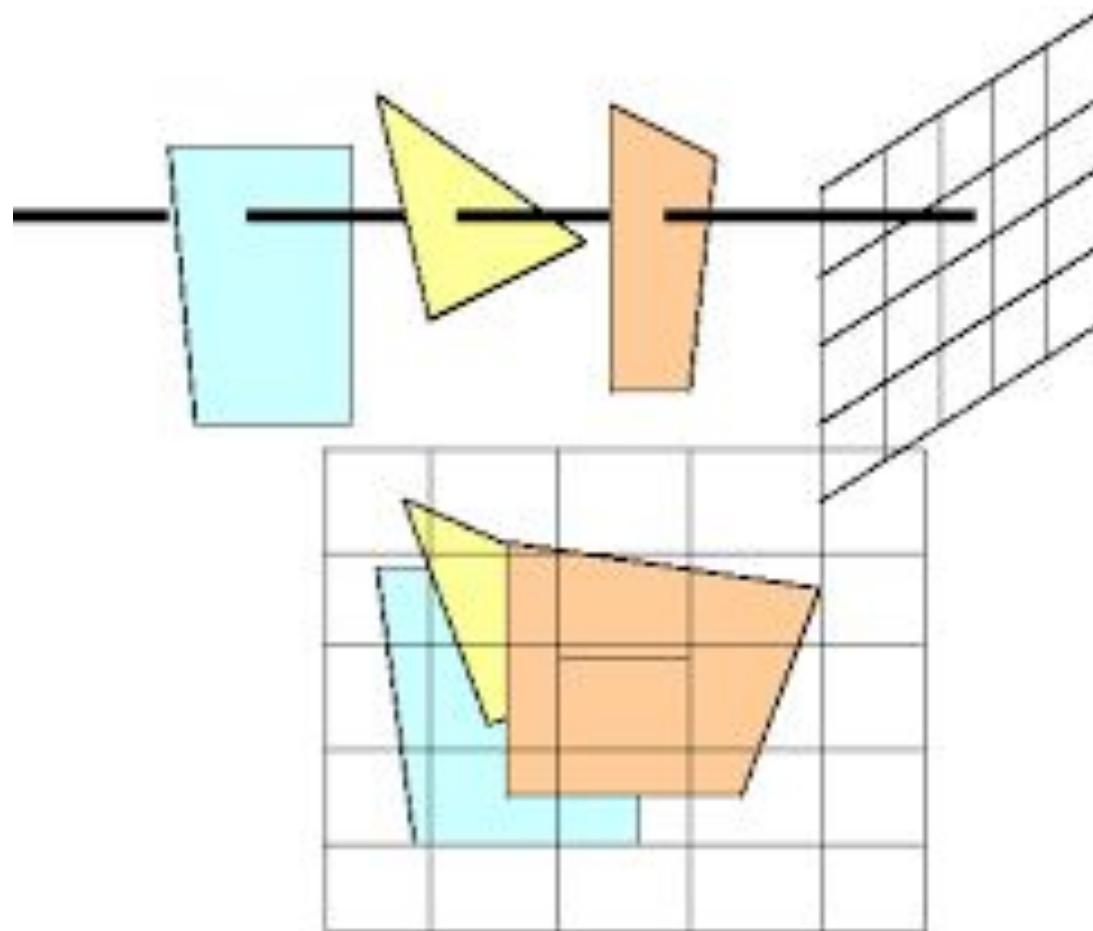
fig.1.

z_v

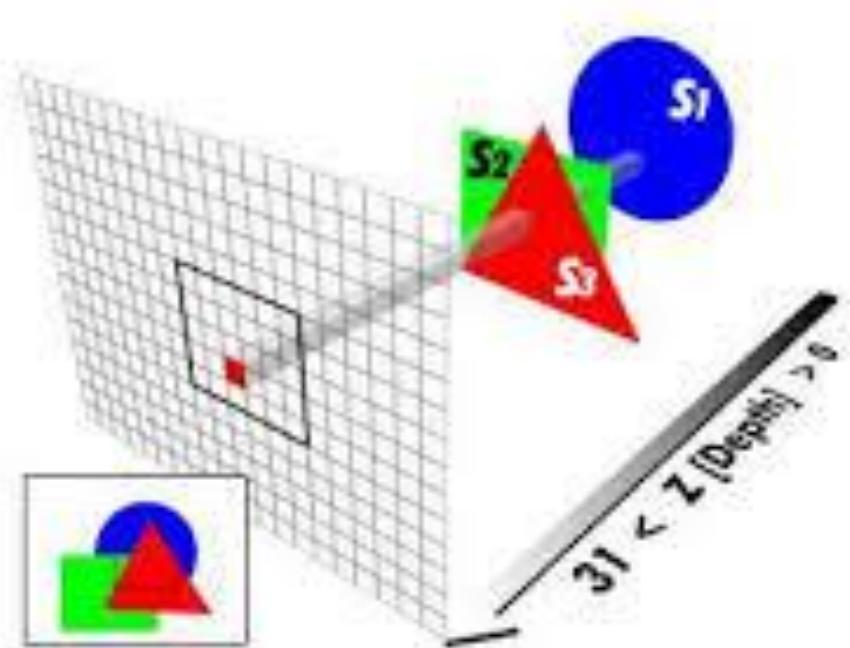
Z-buffer or Depth buffer Method:



Z-buffer or Depth buffer Method:



Z-buffer or Depth buffer Method:



1	0 0 0 0 0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0 0 0 0 0
3	5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4	5 5 15 15 5 5 5 5 15 15 15 5 0 15 15 15 15 15 0 15 15 15 15 15 15 15 15 15 15 15

Z-buffer or Depth buffer Method:

Algorithm:

- i. Initialize the *depth buffer* and *refresh buffer* so that for all buffer position (x,y)
 $\text{depth}(x,y) = 0 \text{ or } (\text{zmin})$ and $\text{refresh}(x,y) = I_{\text{background}}$
- ii. For each position on each polygon surface compare depth values to previously stored values in the depth buffer to determine visibility
Calculate the depth 'z' for each (x,y) position on the polygon
- iii. If $z > \text{depth}(x,y)$ then set
 $\text{depth} = z$ and $\text{refresh}(x,y) = I_{\text{surface}}(x,y)$
where $I_{\text{background}}(x,y)$ is background intensity , $I_{\text{surface}}(x,y)$ is projected intensity value for surface at pixel position (x,y)

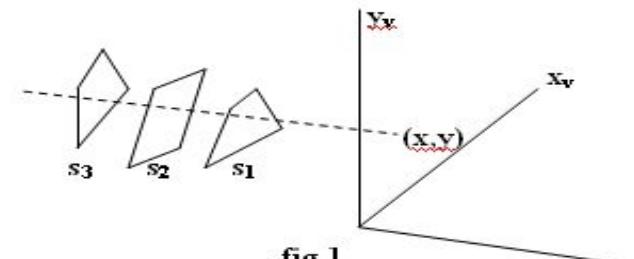


fig 1.

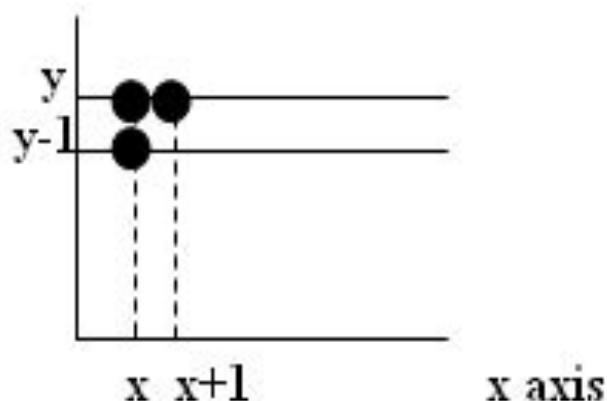
Hidden Surface Removal (Visible Surface Detection)

After all surfaces have been processed the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces

Depth value for a surface position (x, y) is

Let depth z' at $(x + 1, y)$

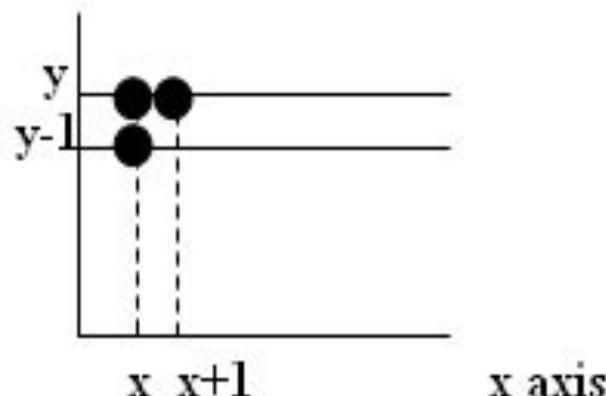
$$z' = -A(x+1) - By - D/c \quad \text{or} \quad z' = z - A/c \dots \dots \dots \text{(ii)}$$



Hidden Surface Removal (Visible Surface Detection)

Now since $(-A/c)$ is constant for each surface, so succeeding depth values across a scan-line are obtained from preceding values with a simple mathematics.

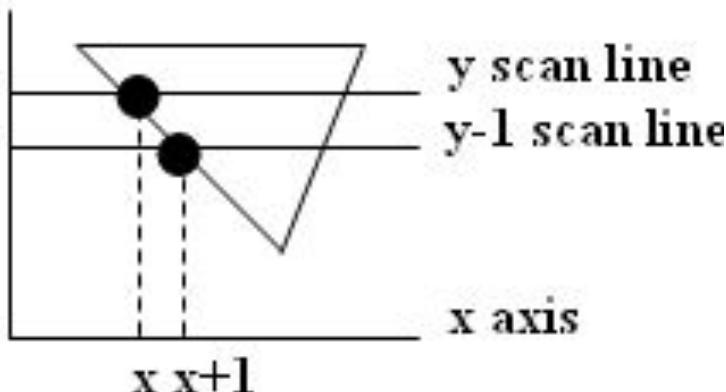
On each scan-line , we start by calculating depth on a left edge of the polygon that intersect that scan-line (fig ii) ‘z’ at each successive position across the scan line are the calculated by equation $z' = -A(x+1) - By - D/c$ or $z' = z - A/c$(ii)



Hidden Surface Removal (Visible Surface Detection)

first determine y coordinate extents of each polygon
y scan line and process from top scan to bottom scan starting y-1 scan
line at a top vertices we can recursively calculate 'x' position
down a left edge of the polygon as
 $x' = x - 1/m$ where m is the slope of edge fig (iii)

'z' values down the edge are then obtained recursively as figure.
 $z' = z + (a/m + b)/c$ or $z' = z + b/c$ as $m \neq 0$ for vertical edge



A Buffer Method

An **extension** of the ideas in the depth-buffer method

It represents an antialiased, **area-averaged, accumulation-buffer** method developed by **Lucasfilm** for implementation in the surface-rendering system called REYES ("Renders Everything You Ever Saw").

A drawback of the depth-buffer method

- deals only with **opaque surfaces** and cannot accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed.

The A-buffer method **expands** the depth buffer

- each position in the buffer can reference a **linked list** of surfaces.
- **more than one surface intensity** can be taken into consideration at each pixel position, and object edges can be antialiased.

A Buffer Method

Each position in the A-buffer has two fields:

- i) depth field - stores a positive or negative real number
- ii) intensity field - stores surface-intensity information or a pointer value.

Depth field is **positive**:

Number stored at that position is the depth of a **single surface** overlapping the corresponding pixel area.

Intensity field then stores the RCB components of the surface color at that point and the percent of pixel coverage.

Depth field is **negative**:

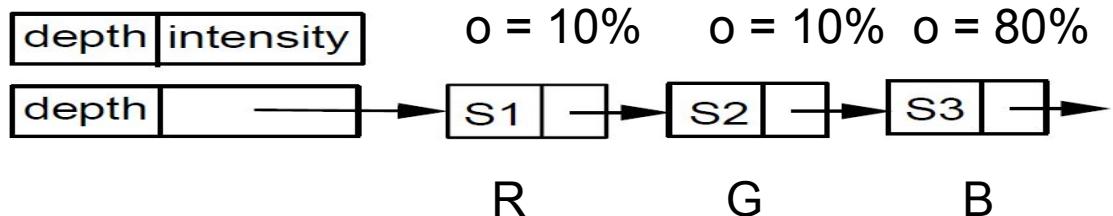
Indicates **multiple-surface** contributions to the pixel intensity.

Intensity field then stores a **pointer to a linked list of surface data**

A Buffer Method

Data for each surface in the linked list includes:

- RGB intensity components
- opacity parameter (percent of transparency)
- depth
- percent of area coverage
- surface identifier
- other surface-rendering parameters
- pointer to next surface



A Buffer Method

The A-buffer can be constructed using methods similar to those in the depth-buffer algorithm.

Scan lines are processed to determine surface overlaps of pixels across the individual scanlines.

Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries.

Using the opacity factors and percent of surface overlaps, we can calculate the intensity of each pixel as an average of the contributions from the overlapping surfaces.

Scan-line Method

ISM for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors.

Instead of filling just one surface we **consider multiple surfaces**.

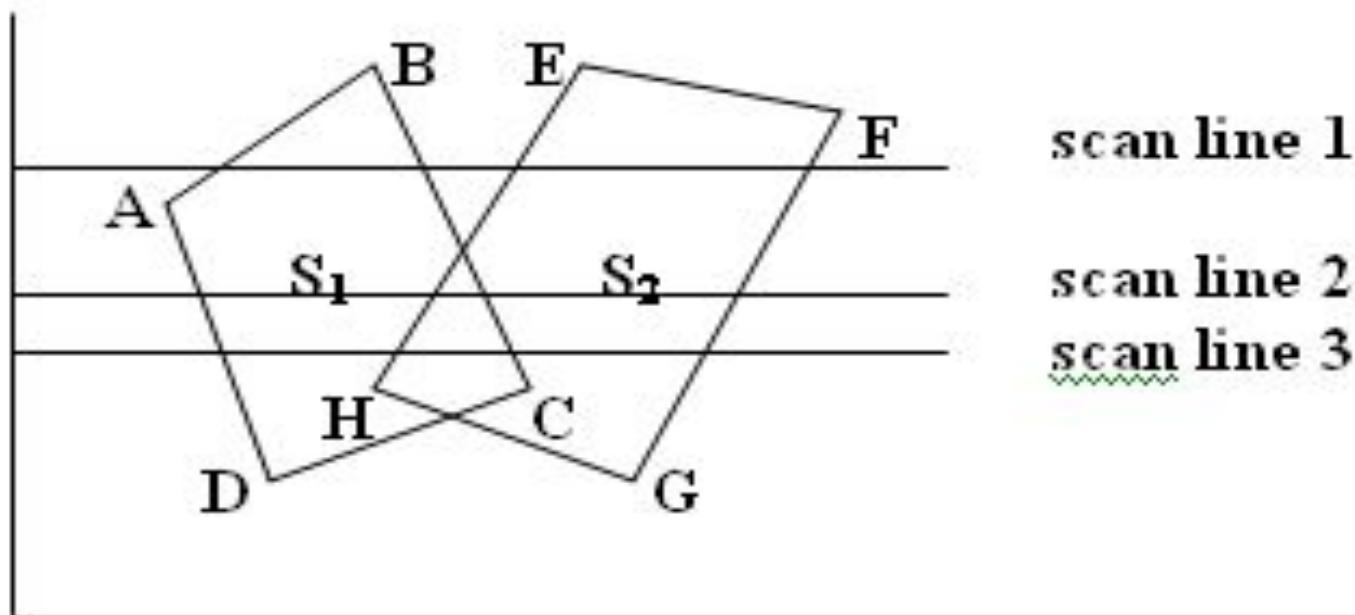
As each scan-line is processed all polygon surfaces intersecting that line are examined to determine which are visible

We assume that **tables** (*polygon*) are set up for the various surfaces (*edge, polygon etc*)

Scan-line Method

Polygon table contains *coefficients of the plane equation* for each surface intensity information for the surfaces and *possible pointers into the edge table*

Active edge list for scan-line 1 contains scan line 1 information from the edge table for edges AB, BC, EH, FG



Scan-line Method

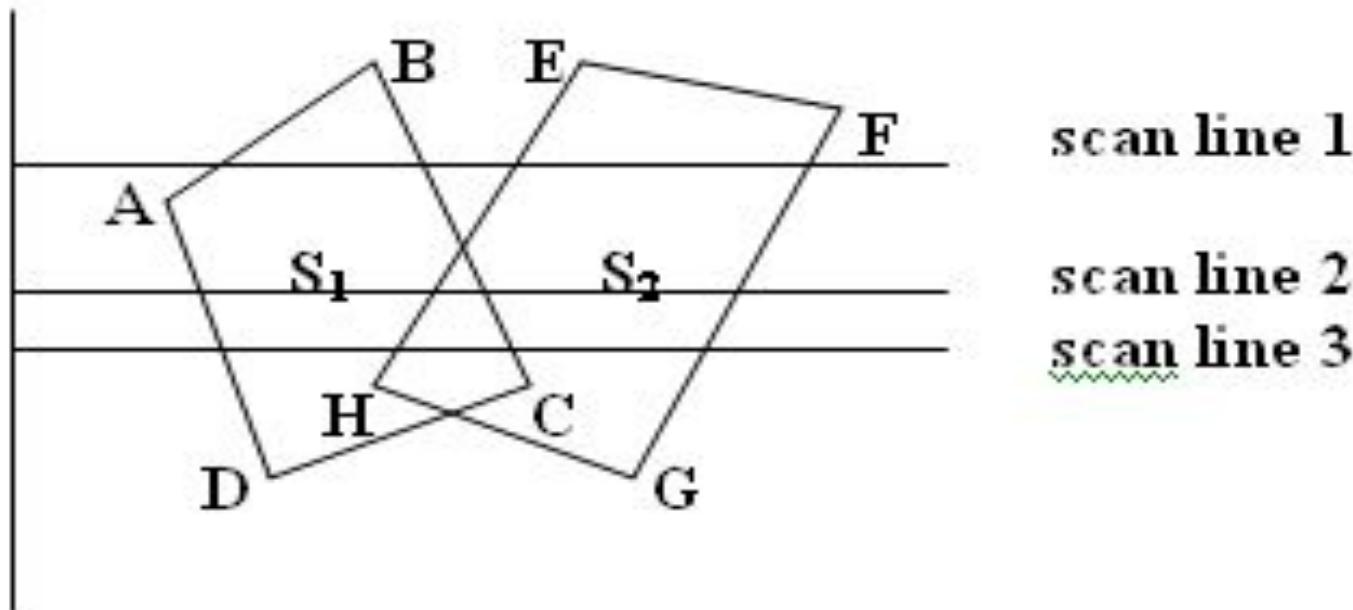
Scan-line 1 edges AB and BC

Only the flags for surfaces S_1 is on

No depth calculation is necessary

Intensity information for surface S_1 is entered from the polygon table into the refresh buffer

Between edges EH and FG only the flag for surface S_2 is on.



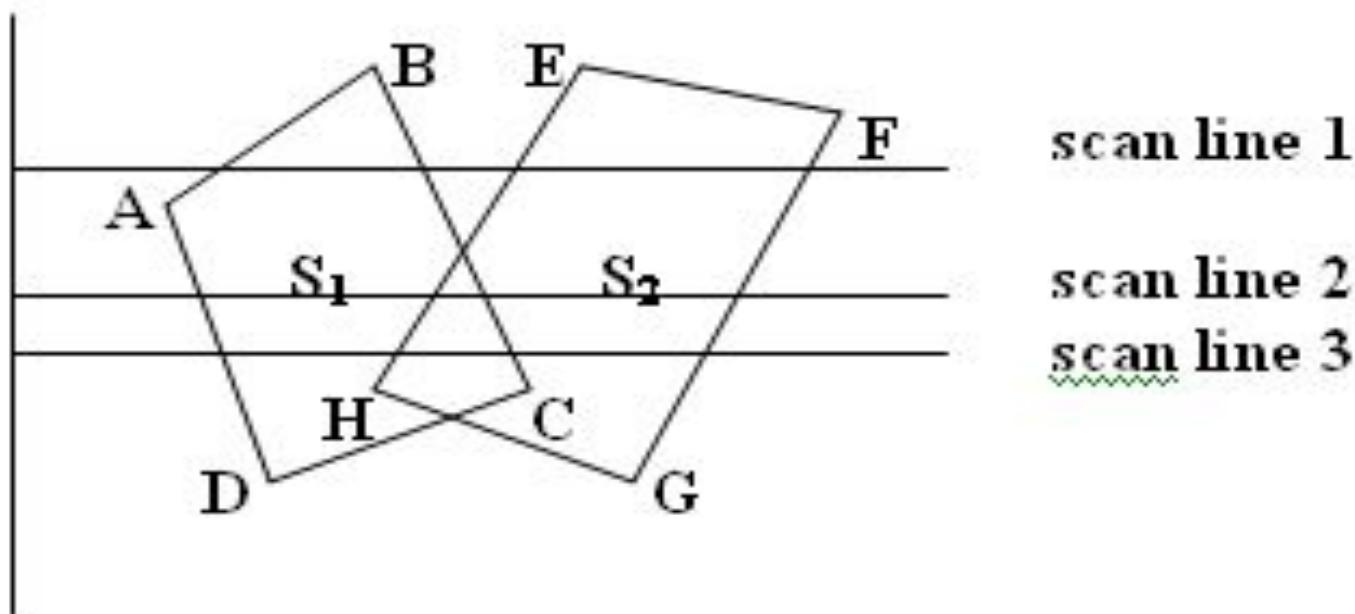
Scan-line Method

Scan line 1 (edges EH and FG)

Only the flag for surface S_2 is on

No other position along scan-line1 intersect surfaces so intensity values in the other areas are set to background intensity

For scan-line 2 and 3 the active edge list contains edges AD, EH, BC,FG



Scan-line Method

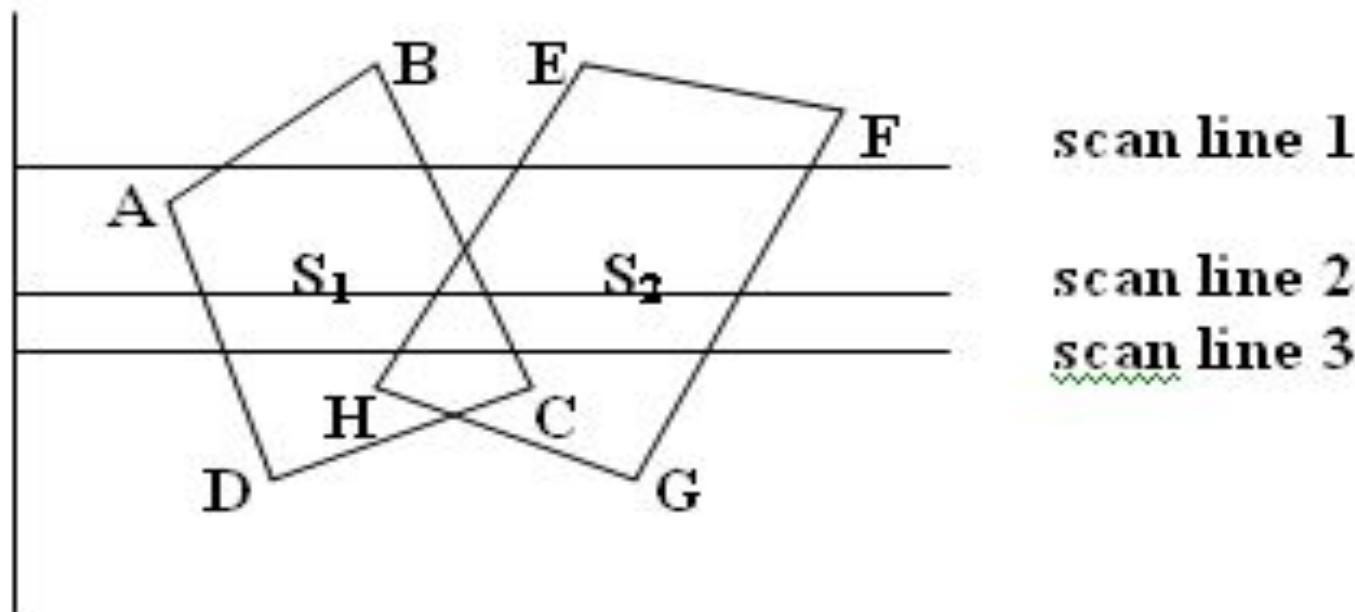
Scan-line 2 (active edge list contains edges AD, EH, BC and FG)

From edge AD to edge EH

Only the flag for surface S_1 is on

From edge EH and BC

Flags for both surfaces (S_1, S_2) are on and depth calculations must be made using plane coefficients for the two surfaces,

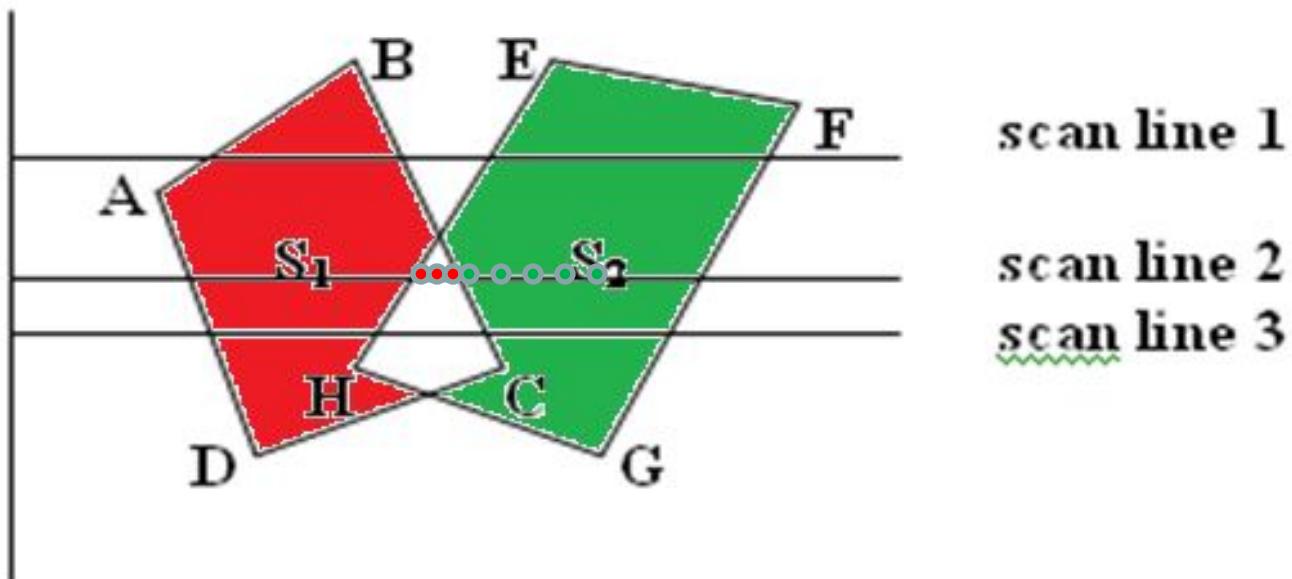


Scan-line Method

if 'z' of surface S_1 is less than that of S_2
Intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered.

Then the flag for surface S_1 goes off and intensities for surface S_2 are stored until edge FG is passed

Any number of overlapping polygon surfaces can be processed with this scan-line method.



List Priority Algorithms

Determines a visibility ordering for objects ensures that a correct picture results if objects are rendered in that order.

e.g. if no object overlaps in ‘z’ then we need only to sort objects by increasing ‘z’ and render them in that order.

Farther objects are obscured by closer ones as pixels from the closer polygons overwrite those of more distant ones.

If objects overlap in ‘z’, we may still be able to determine a correct order.

If objects cyclically overlap, or penetrate each other, then there is no correct order.

List Priority Algorithms

Hybrids that combine both **object** and **image precision operations**.

Depth comparison and object splitting are done with **object precision**.

Scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with **image precision**.

Depth sorting Method

Makes use of both **image and object space** operations to:

- i. Sort surfaces in decreasing depth order (using both image and object space)
- ii. Scan converted in order starting with surface with greatest depth (using image space)

Also referred to as **painters algorithm** because an artist first paints background color then most distant object then nearer object and so on.

In the end **the foreground objects are painted** on canvas over the background. Each layer of paint covers up previous layer

Depth sorting Method

Approach:

First **sort surfaces** according to their distance from view plane.

Intensity values for farthest surface are entered into refresh buffer.

Taking each surface in turn (in decreasing depth order) **paint surface intensities onto frame buffer over intensity of previously processed surface**

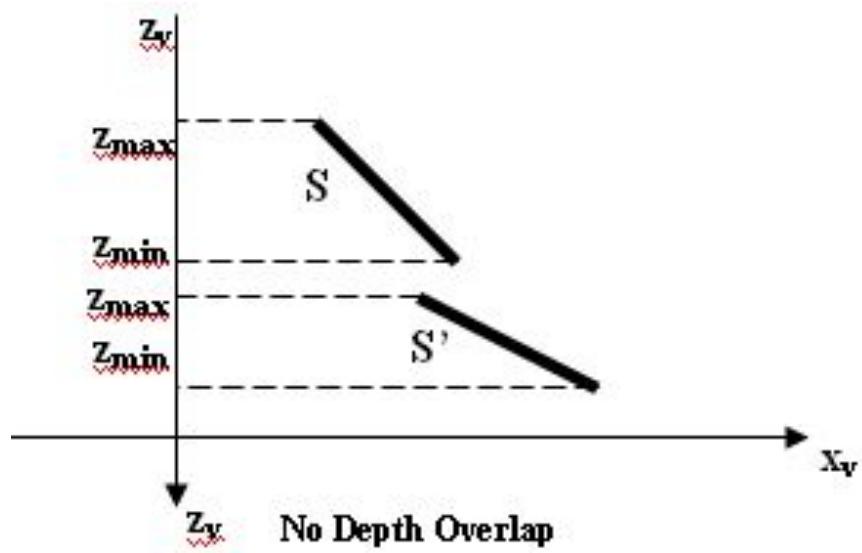
Depth sorting Method

Assume we're viewing along 'z' direction.

Surfaces z_{\max} are ordered according to largest 'z' value on each surface z_{\min}

Surface with greatest depth is compared with other z_{\max} surface in list to see if there are any overlaps in depth z_{\min}

If no depth overlap then surface S is scan converted



B. Depth sorting Method

Additional test are required for each surface in case of depth overlap with surface S:

- i. **Bounding rectangle** in xy plane for two surfaces don't overlap
- ii. **Surface S is completely behind** overlapping surface relative to viewing position
- iii. **Overlapping surface S' is completely in-front** of S relative to viewing position
- iv. **Projection of 2 surfaces** onto view plane don't overlap.

If any of these conditions is true then no reordering of surfaces is necessary.
i.e. if all surfaces pass at least one of the tests then none of those surfaces are behind S.

Depth sorting Method

Similarly, we first sort surfaces according to their distance from view plane.

Intensity values for farthest surface are entered into refresh buffer.

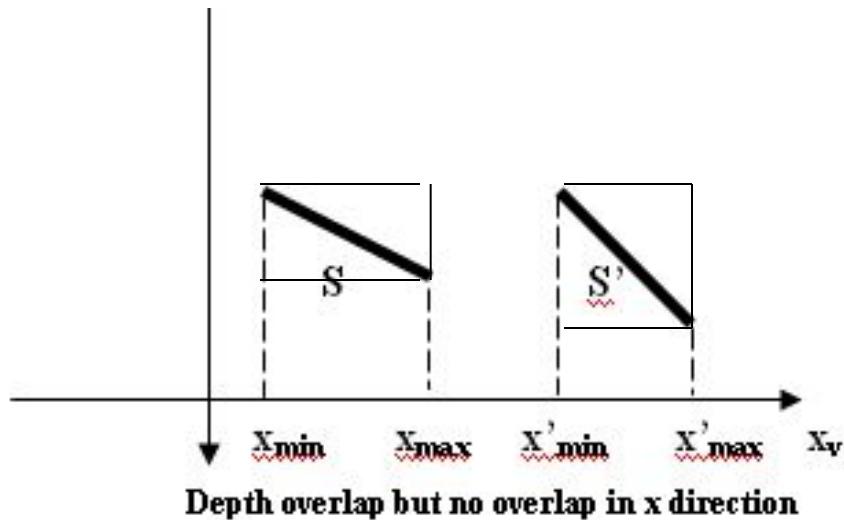
Taking each surface in turn (in decreasing depth order) we paint surface intensities onto frame buffer over intensity of previously processed surface

Depth sorting Method

For test (i)

Perform two steps to check overlap first in 'x' then in 'y' direction.

If either of these directions show no overlap S is completely behind then the two planes cannot obscure one another.



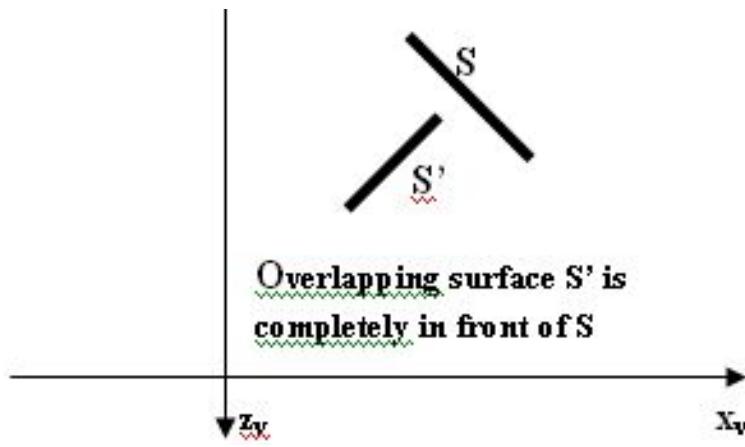
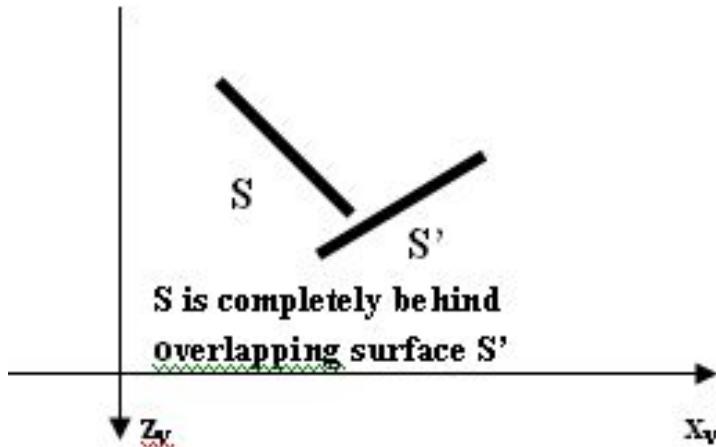
Depth sorting Method

For tests (ii) and (iii),
Perform inside outside polygon test.

Substitute coordinates for all vertices into plane equation for overlapping surfaces and check the sign of the result obtained.

Based on plane equation if all vertices of S are inside S' then S is completely behind S' .

Similarly, S' is completely in-front of S if all vertices of S are outside of S' .



AREA-SUBDIVISION METHODS

This technique for hidden-surface removal is essentially an image-space method, but object-space operations can be used to accomplish depth ordering of surfaces.

The area-subdivision method takes advantage of area coherence (consistency) in a scene by locating those view areas that represent part of a single surface.

This method is applied successively by **dividing the total viewing area into smaller and smaller rectangles** until each small area is the projection of part of n single visible surface or no surface at all.

To implement this method, tests need to establish that can quickly identify the area as part of a single surface or show that the area is too complex to analyze easily.

AREA-SUBDIVISION METHODS

Starting with the total view, apply the tests to determine whether subdivision of the total area must be done into smaller rectangles.

If the tests indicate that the view is sufficiently complex, subdivide it.

Next, apply the tests to each of the smaller areas, subdividing these if the tests indicate that visibility of a single surface is still uncertain.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel.

An easy way to do this is to successively divide the area into four equal parts at each step.

AREA-SUBDIVISION METHODS

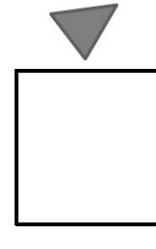
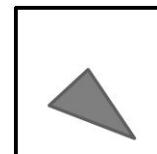
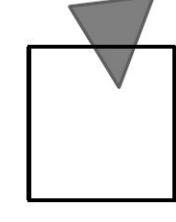
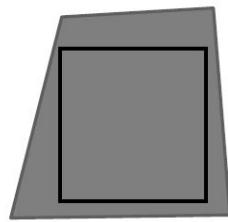
Four possible relationships of a surface with a specified area boundary.

Relative surface characteristics:

- **Surrounding** surface-One that completely encloses the area.
- **Overlapping** surface-One that is partly inside and partly outside the area.
- **Inside** surface-One that is completely inside the area.
- **Outside** surface-One that is completely outside the area.

No further subdivisions of a specified area are needed if one of the following conditions is true:

1. All surfaces are **outside** surfaces with respect to the area.
2. Only **one inside, overlapping, or surrounding surface** is in the area.
3. A **surrounding surface obscures all other surfaces** within the area boundaries.

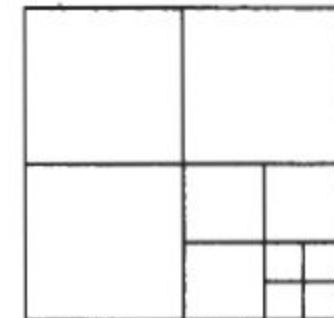


surrounding

intersecting

contained

disjoint



AREA-SUBDIVISION METHODS

Test 1 : Carried out by checking the bounding rectangles of all surfaces against the area boundaries.

Test 2 : **Uses** the bounding rectangles in the **xy** plane to identify an inside surface. Once a single inside, overlapping, or surrounding surface has been identified, its pixel intensities are transferred to the appropriate area within the frame buffer.

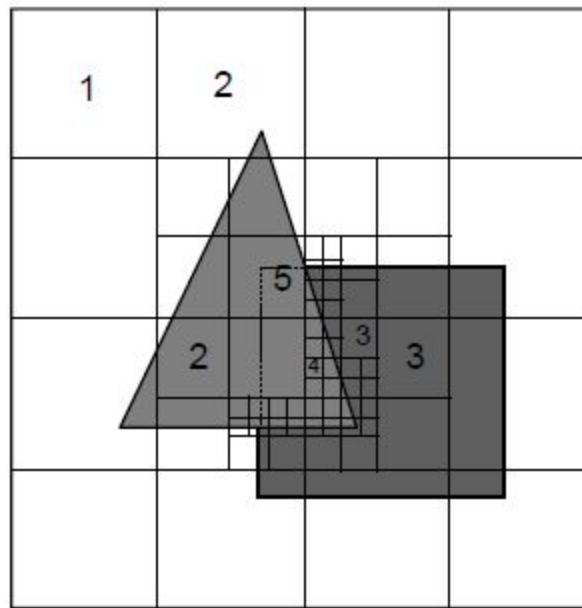
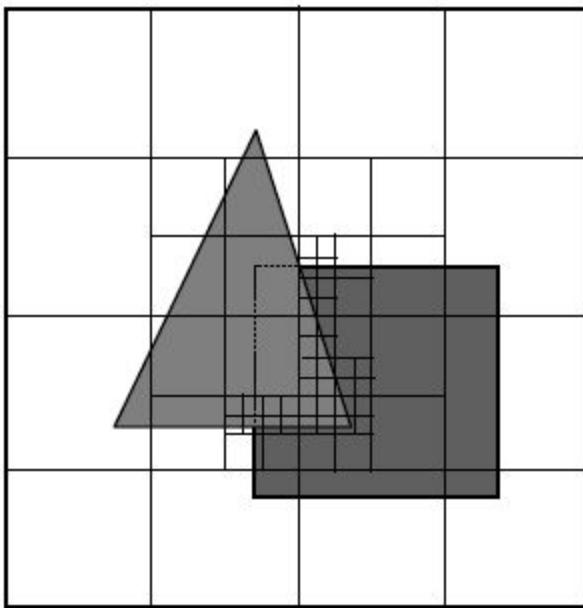
Test 3 : Order surfaces according to their minimum depth from the view plane.

For each surrounding surface, then compute the maximum depth within the area under consideration. If the maximum depth of one of these surrounding surfaces is closer to the view plane than the minimum depth of all other surfaces within the area, test 3 is satisfied.

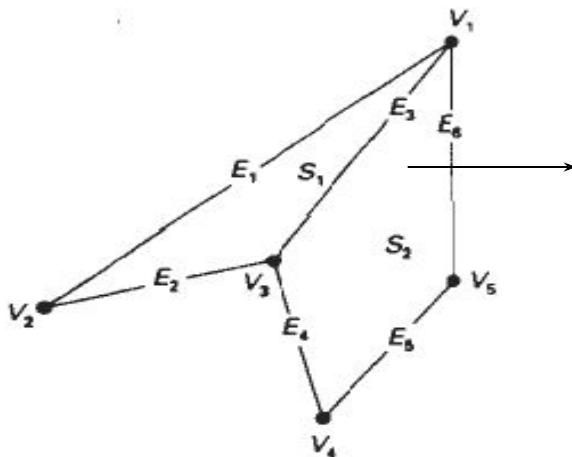
Or use plane equations to calculate depth values at the four vertices of the area for all surrounding, overlapping, and inside surfaces,

If the calculated depths for one of the surrounding surfaces is less than the calculated depths for all other surfaces, test 3 is true. Then the area can be filled with the intensity values of the surrounding surface.

AREA-SUBDIVISION METHODS



Back face culling



VERTEX TABLE
$V_1: x_1, y_1, z_1$
$V_2: x_2, y_2, z_2$
$V_3: x_3, y_3, z_3$
$V_4: x_4, y_4, z_4$
$V_5: x_5, y_5, z_5$

EDGE TABLE
$E_1: V_1, V_2$
$E_2: V_2, V_3$
$E_3: V_3, V_1$
$E_4: V_3, V_4$
$E_5: V_4, V_5$
$E_6: V_5, V_1$

POLYGON-SURFACE TABLE
$S_1: E_1, E_2, E_3$
$S_2: E_3, E_4, E_5, E_6$

Back face culling

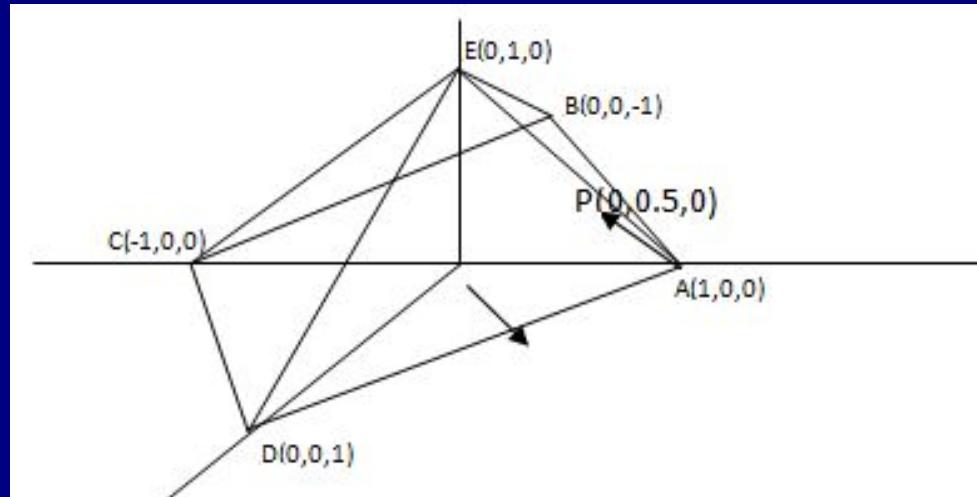
Removes faces in the back of an object away from the viewer.

Compute the outward normal to face AED in the rectangular pyramid

The normal is found by the cross product of vectors

$$AE = -i + j \quad AD = -i + k$$

$$\begin{aligned} n &= AE \times AD = \begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \\ &= i + j + k \end{aligned}$$



Choosing a point inside the pyramid $P(0,0.5,0)$

The AP vector is $AP = -i + 0.5 j$

The dot product of this vector and the normal will indicate the normal orientation:

$$\begin{aligned} n \cdot AP &= (i + j + k) \cdot (-i + 0.5 j) \\ &= 1 * -1 + 1 * 0.5 = -1 + 0.5 = -0.5 \end{aligned}$$

Since this value is negative the normal is pointing outward hence is a invisible or backface

AREA-SUBDIVISION METHODS

Back face culling

Removes faces in the back of an object away from the viewer.

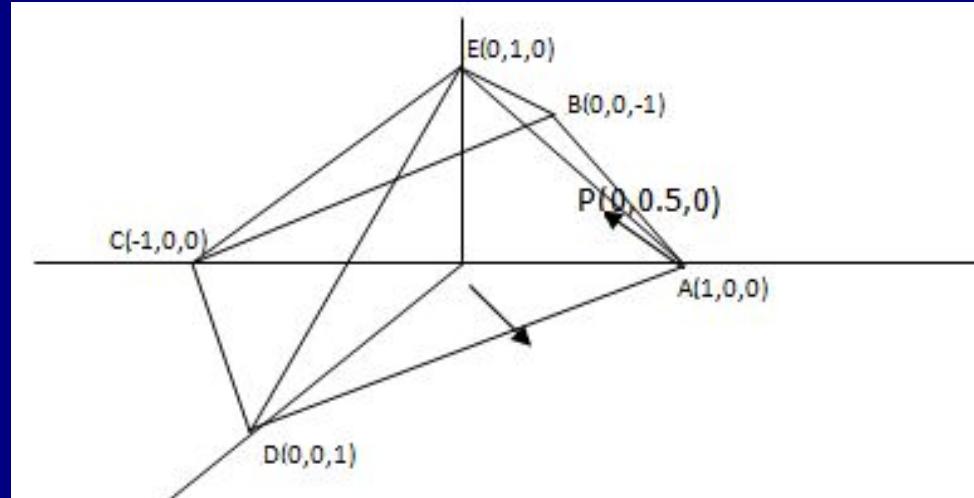
Compute the outward normal to face AED in the rectangular pyramid

The normal is found by the cross product of vectors

$$AE = -i + j \quad AD = -i + k$$

$$n = AE \times AD = \begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$$

$$= i + j + k$$



If the vector V is established from the view point P(5,5,5) to a point in AED such as point A (1,0,0)

$$\text{Then } V = 4i + 5j + 5k$$

The dot product of n and V is

$$n \cdot V = (1)(4) + (1)(5) + (1)(5) = 14$$

since $n \cdot V$ is greater than 0 the face AED is visible.

AREA-SUBDIVISION METHODS

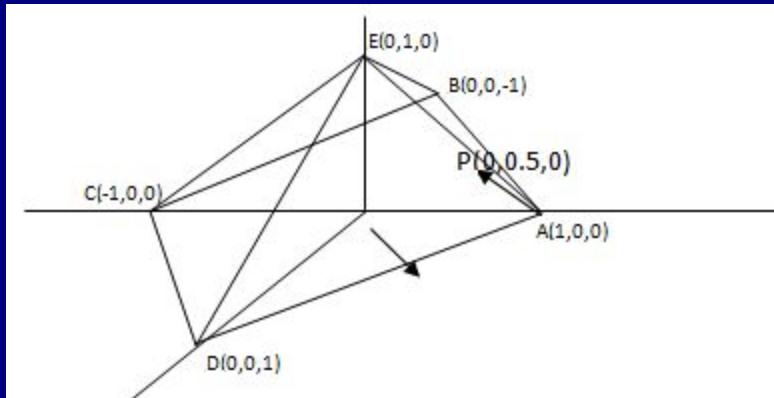
If the vector V is established from the view point P(5,5,5) to a point in AED such as point A (1,0,0)

$$\text{Then } \mathbf{V} = 4\mathbf{i} + 5\mathbf{j} + 5\mathbf{k}$$

The dot product of n and V is

$$\mathbf{n} \cdot \mathbf{V} = (1)(4) + (1)(5) + (1)(5) = 14$$

since $\mathbf{n} \cdot \mathbf{V}$ is greater than 0 the face AED is visible.



If a person situated at the location $P(-1,4,5)$ is looking at a surface that has vertices $A(1,0,0)$, $B(0,0,-1)$, $C(-1,0,0)$ and $D(0,0,1)$ will the surface be a back face

Illumination Models and Surface Rendering Methods

~~Realistic displays of a scene are obtained by~~

_perspective projection

_applying natural light effects to the visible surfaces

Illumination model (lighting model) or shading model, is used to **calculate the intensity of light** that we should see at a given point on the surface of an object.

A **surface rendering algorithm** uses the intensity calculations from an illumination model to determine the light intensity calculation

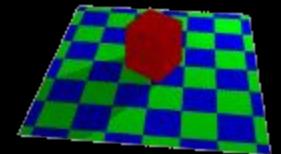
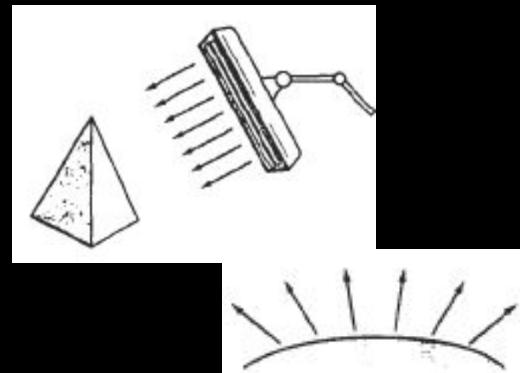
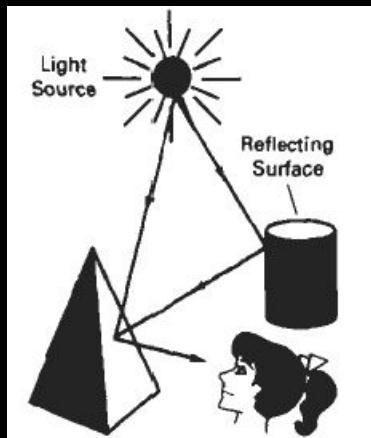
Light Sources

Point source: tungsten filament bulb image can be seen

Distributed light source: fluorescent light

When light is incident on an opaque surface part reflected part absorbed. Surface that are **rough** or grainy tend to **scatter reflected light** in all direction is called diffuse reflection

Light sources create **highlights** or bright spots called specular-reflection.



Ambient light

Ambient light surface directly not exposed directly but visible if nearby objects are illuminated

Combination of light reflections from various surfaces to produce a uniform illumination called the ambient light or **background** light(no shadow's produced)

It has no spatial or direction characteristics and amount on each object is a constant for all surfaces and over all directions

$$I_{am} = k_a \cdot I_a$$

Ambient light is an approximation of global diffuse, light effects .

Diffuse Reflections

Diffuse reflections are **constant** over each surface in a scene independent of viewing direction

k_d or diffuse reflection coefficient or diffuse reflectivity (0 to 1)

k_d is nearly 1 for highly reflective surface and k_d is 0 where light absorbs (black surfaces)

Diffuse reflection intensity at any point on the surface as

$$I_{ambDiff} = k_d \cdot I_d$$

Where $I_{ambDiff}$ is ambient light due to diffusion and I_d is light due

assuming diffuse reflections from the surface are scattered with equal probability in all directions independent of the



Viewing direction (called “ideal diffuse reflectors”) also called Lambertian reflectors and governed by Lambert’s Cosine Law.

If “angle of incidence” between incoming light direction and surface normal is 0

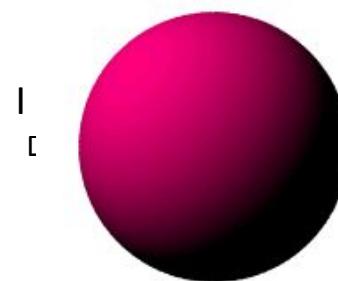
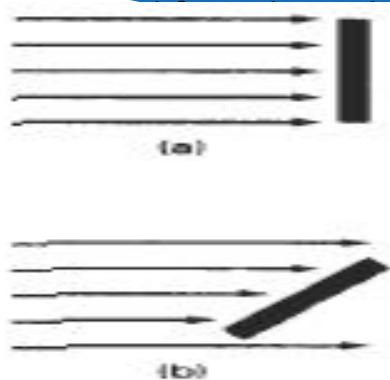
$$\frac{I_{\text{Diff}}}{d} = k_d \cdot I_{\text{Light}} \cos \theta$$

where I_{Diff} is light due to diffusion
 $\frac{I_{\text{Diff}}}{d}$

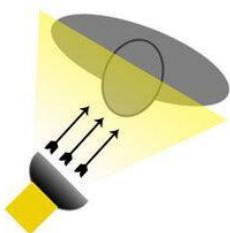
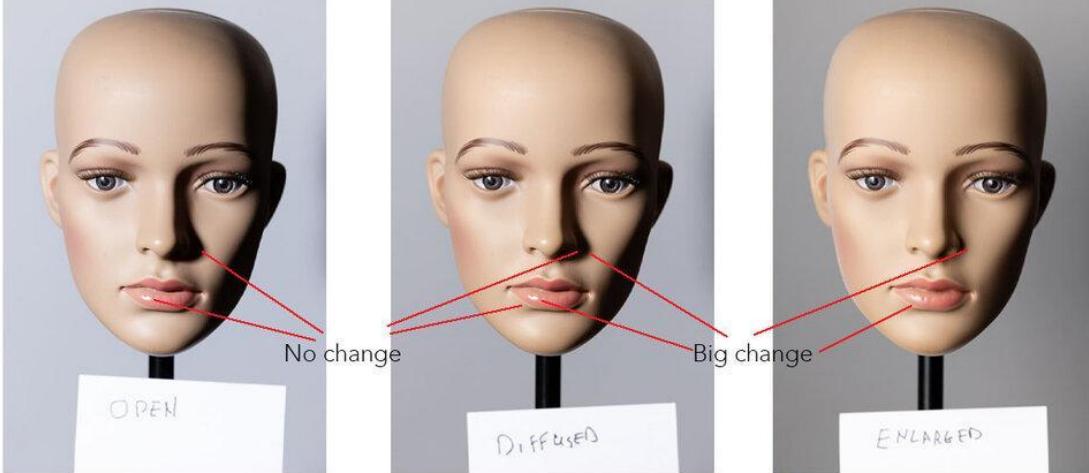
If N is unit normal vector to a surface and L is unit direction vector to the point light source then

$$\frac{I_{\text{Diff}}}{d} = k_d \cdot I_{\text{Light}} (N \cdot L)$$

In addition many graphics packages introduce an ambient reflection coefficient k_a to

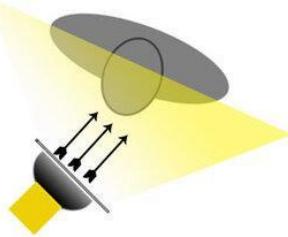


$$I_{\text{diffuse}} = k_d I_{\text{light}} \cos \theta$$
$$+ k_a + k_d \cdot I_{\text{Light}} (N \cdot L)_{\theta}$$
A diagram of a blue rectangular plane with a vertical normal vector N and a yellow arrow representing the light direction L . A small circle indicates the angle θ between them.



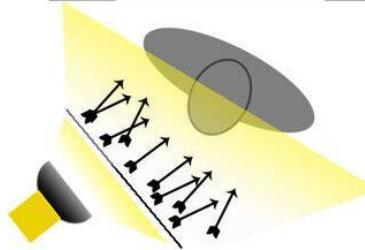
Open reflector

Light rays strike the subject direct with no diffusion. Shadow edge is hard and shadow density is dark (contrast).



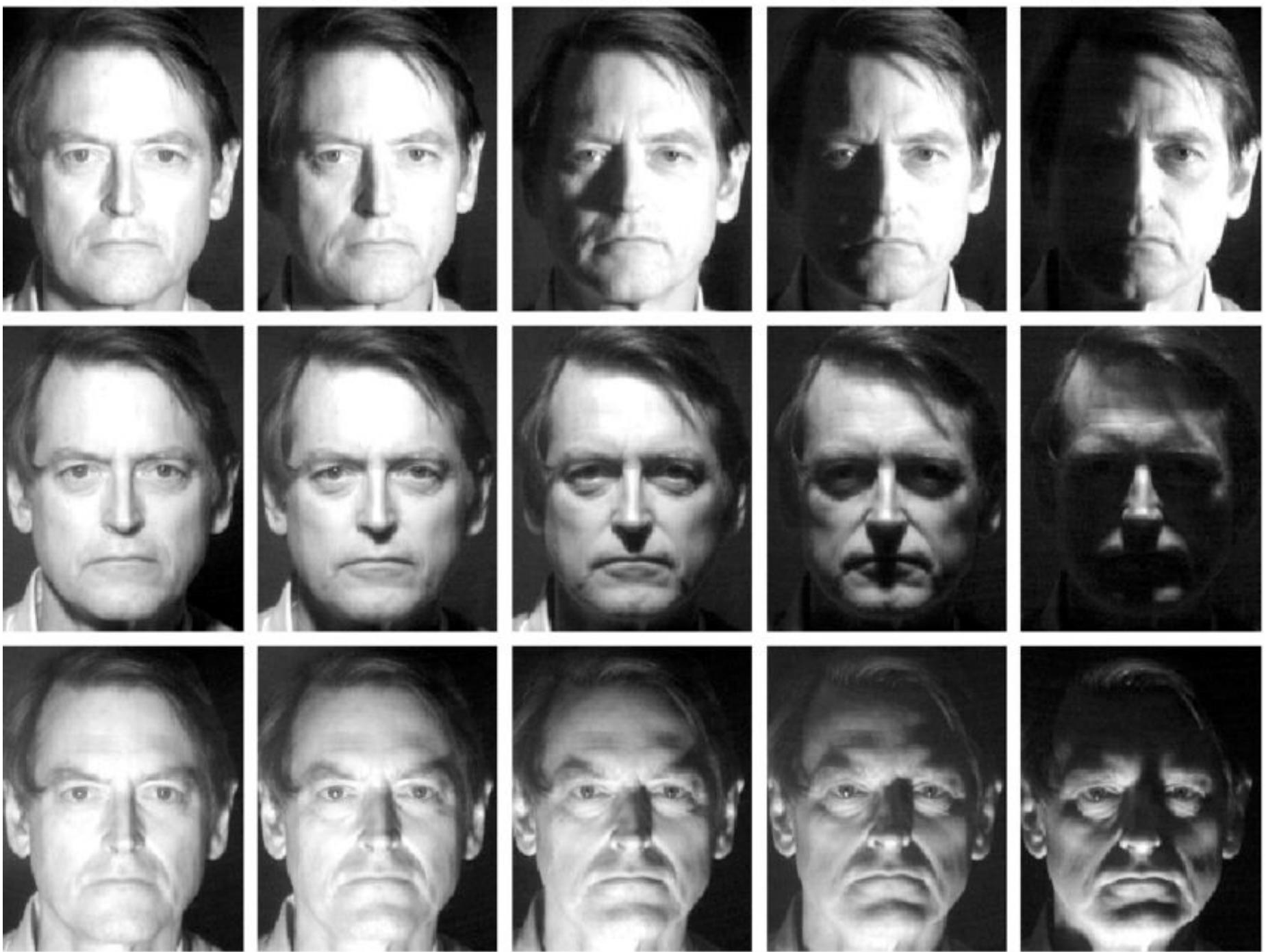
**Open reflector
plus diffusion**

Shadow edge softness change because the size of light is the same. Highlights don't change because the distance is the same. Contrast is lower because of the spread of light and environment.



Open reflector

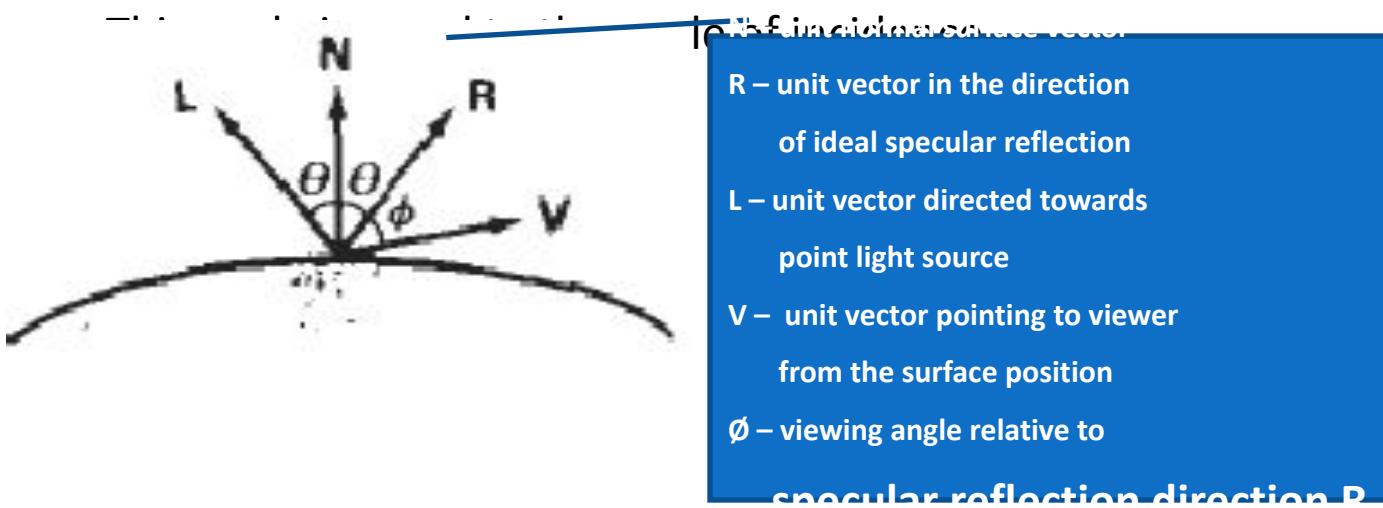




Specular Reflection/Phong Model

When we look at an illuminated shiny surface, such as polished metal, an apple etc we see a highlight or bright spot, at certain viewing direction this phenomenon is called “specular reflection”

It is the result of **total or near total reflection** of the incident light in a **concentrated region** around the “specular reflection angle”.



Specular Reflection/Phong Model

For monochromatic specular reflections intensity variations can be

approximated by SR coefficient $W(\Theta)$

$W(\Theta)$ tends to increase as Θ increases, at $\Theta = 90^\circ$ $W(\Theta) = 1$ and all incident light is reflected.

Fresnel's law of reflection describes specular reflection intensity with Θ and using $W(\Theta)$, Phong specular reflection model as

$$\frac{I_{\text{spec}}}{I_L} = W(\Theta) \cos \phi \quad \text{where } I_L \text{ is intensity of light source}$$

ϕ is viewing angle relative to the specular reflection direction R.

So transparent materials like glass exhibit specular reflection as θ approaches 90° . At $\theta = 0$ about 4 percent of the incident light on a glass surface is reflected.

For ideal reflection (perfect mirror), incident light is reflected along the specular reflection direction i.e. V and R coincide ($\phi = 0$).

Shiny surfaces have narrow ϕ and dull surfaces have wider ϕ

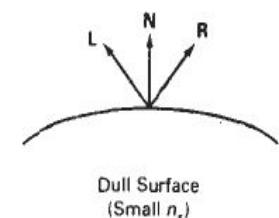
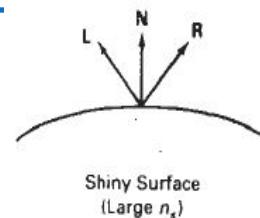
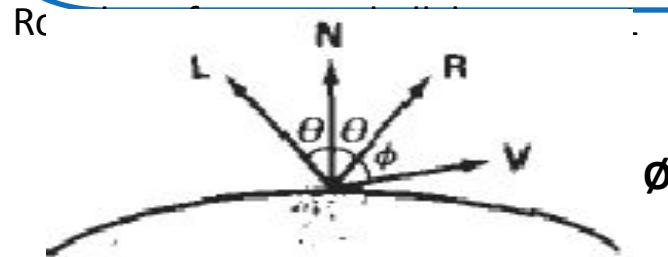
An empirical model for calculating specular reflection range was developed by **Phong Bui Tuong** called “Phong specular reflection” model and it sets the intensity of specular reflection directly proportional to

$$\cos n_s \phi \quad \phi \in 0 \text{ to } 90$$

Specular reflection parameter n_s is determined by type of surface

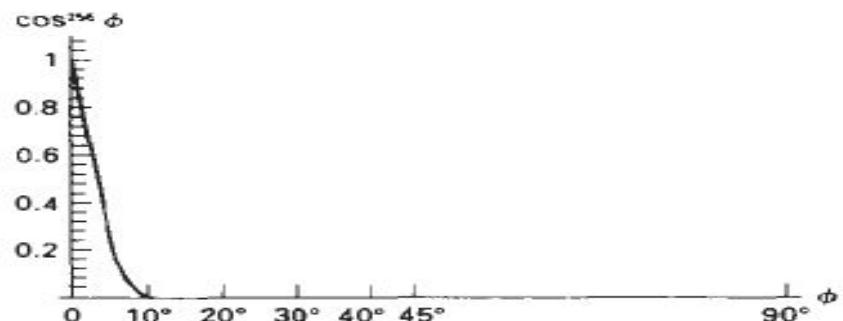
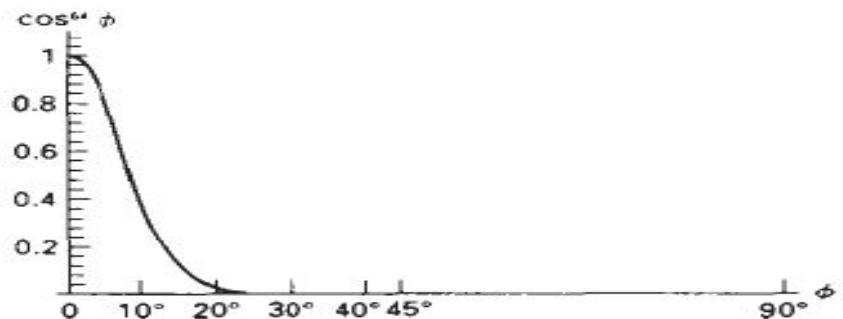
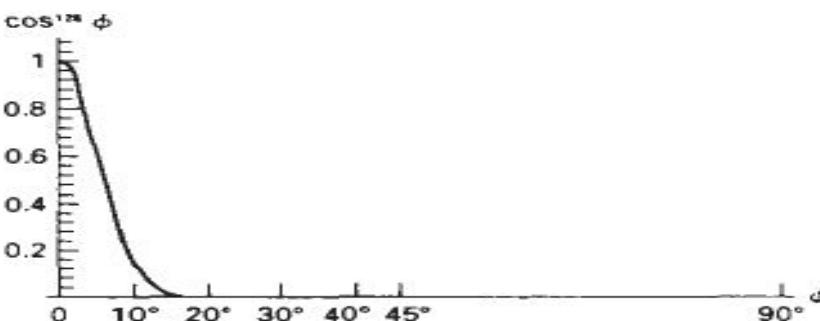
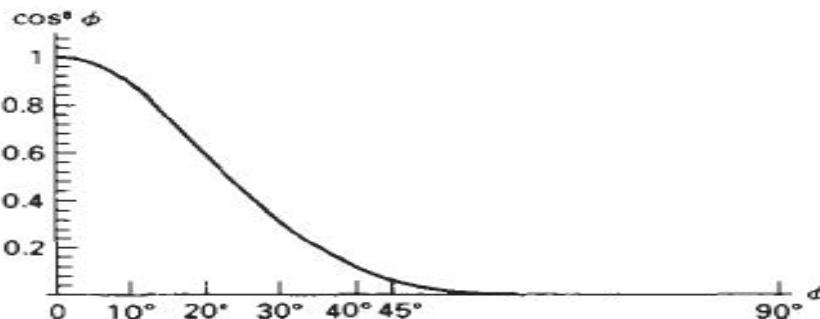
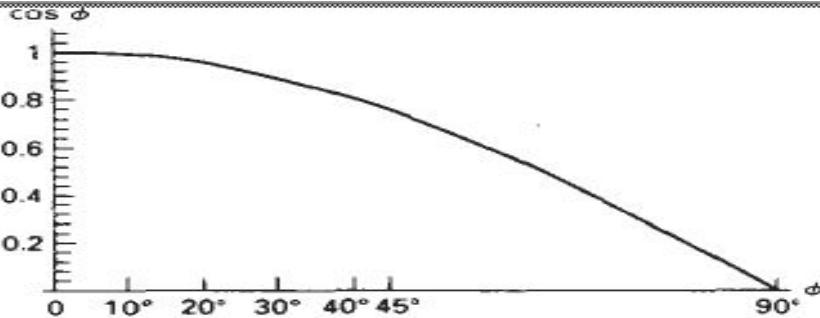
Very shiny surface has large n_s value

Very dull surface has smaller n_s value (down to 1)



Specular Reflection/Phong Model

Intensity of specular reflection depends on material properties of surface,
other factors such as polarization , color of incident light.



Illumination Models and Surface Rendering Methods

Combined diffuse and specular reflections with multiple light sources

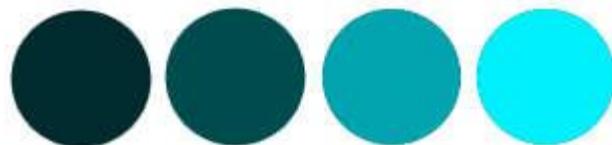
For single point light source

$$I = I_{\text{diffuse}} + I_{\text{spec}} = K_a I_a + K_d I_d (N \cdot L) + K_s I_s (N \cdot H) ns$$

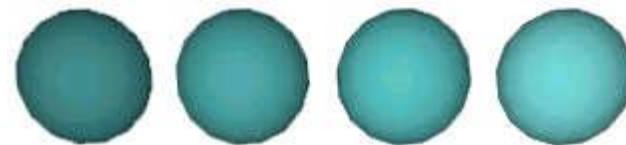
For multiple light sources

$$I = \sum_{i=1}^{a_a} I_i [K_d (N \cdot L)_i + K_s (N \cdot H)_i]$$

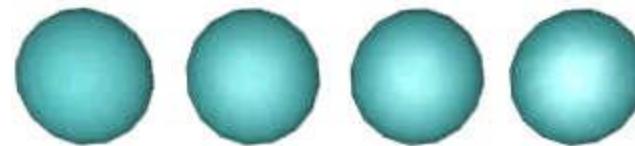
- Ambient



- Diffuse



- Specular



A. Constant Intensity Shading (Flat Shading)

Fast and simple method for rendering of an object with polygon surfaces in CIS also called Flat Shading

Single intensity calculated for each polygon and useful for **quickly displaying** the general appearance of curved surface

This method is accurate if :

The object is a **polyhedron** and is not an approximation of an object with a curved surface

All light sources illuminating the object are **sufficiently far** from the surface

The **viewing position is sufficiently far** from the surface so that V.R is constant over the surface

Even if all conditions are not true, we can still reasonable approximate surface lighting effects using small polygon facets with flat shading and calculate the intensity for each facet at the center of the polygon.

B. Gouraud Shading

Intensity interpolation scheme developed by Gouraud

Renders a polygon surface by linearly interpolating intensity values across the surface

Intensity values for each polygon are matched with the values of the adjacent polygon along the common edge thus eliminating the **intensity discontinuities occur in “flat shading”**

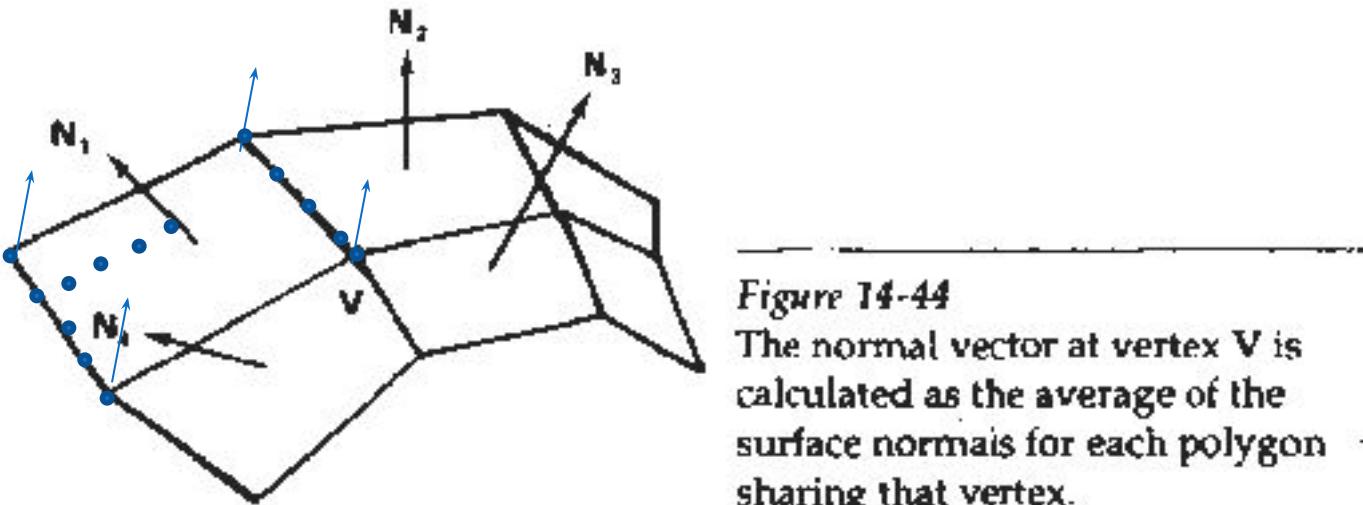


Figure 14-44

The normal vector at vertex V is calculated as the average of the surface normals for each polygon sharing that vertex.

Calculation for each polygon surfaces

- Determine the **average unit normal vector** at each polygon vertex.
- **Apply an illumination model** to each vertex to calculate the vertex intensity
- Linearly **interpolate the vertex intensities** over the surface of the polygon

N_1 normal to ABCD plane , N_2 normal to CDEF plane and so on .

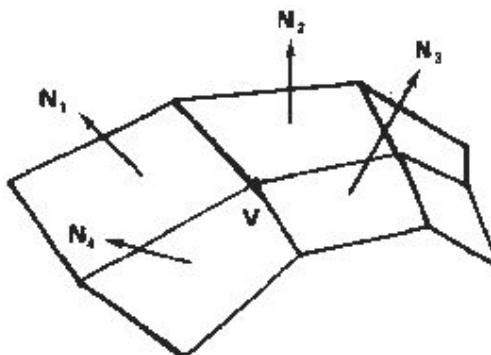


Figure 14-44
The normal vector at vertex **V** is calculated as the average of the surface normals for each polygon sharing that vertex.

For any vertex position V normal unit vector

$$\mathbf{N}_v = \frac{\sum_{k=1}^n \mathbf{N}_k}{\left| \sum_{k=1}^n \mathbf{N}_k \right|}$$

once \mathbf{N}_v is known intensity at vertices can be obtained from lighting model

Next step:

Interpolating intensities along polygon edges

Fast method to find intensity at 4 using 1 and 2
using only vertical displacement

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} \cdot I_1 + \frac{y_1 - y_4}{y_1 - y_2} \cdot I_2$$

Similar process for I_5 , using 3 and 2

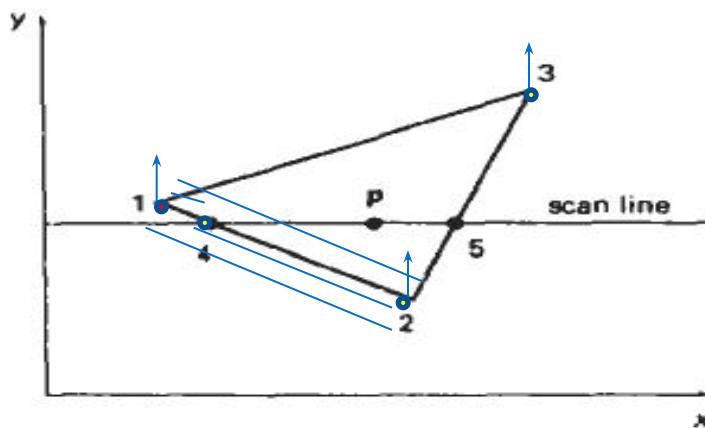


Figure 14-45

For Gouraud shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point p is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

For interior point 'p' interpolated from the bounding intensities at point 4 & 5

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} \cdot I_4 + \frac{x_p - x_4}{x_5 - x_4} \cdot I_5$$

Easier than this is incremental calculations for successive edge intensity values

$$I = \frac{y - y_2}{y_1 - y_2} \cdot I_1 + \frac{y_1 - y}{y_1 - y_2} \cdot I_2$$

for next scan line $y - 1$

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

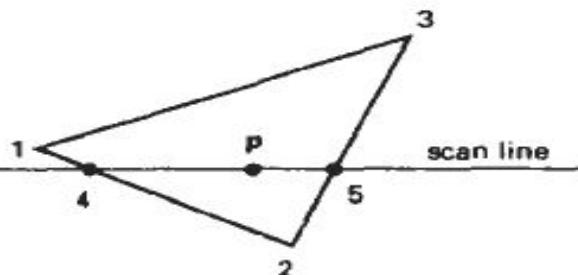
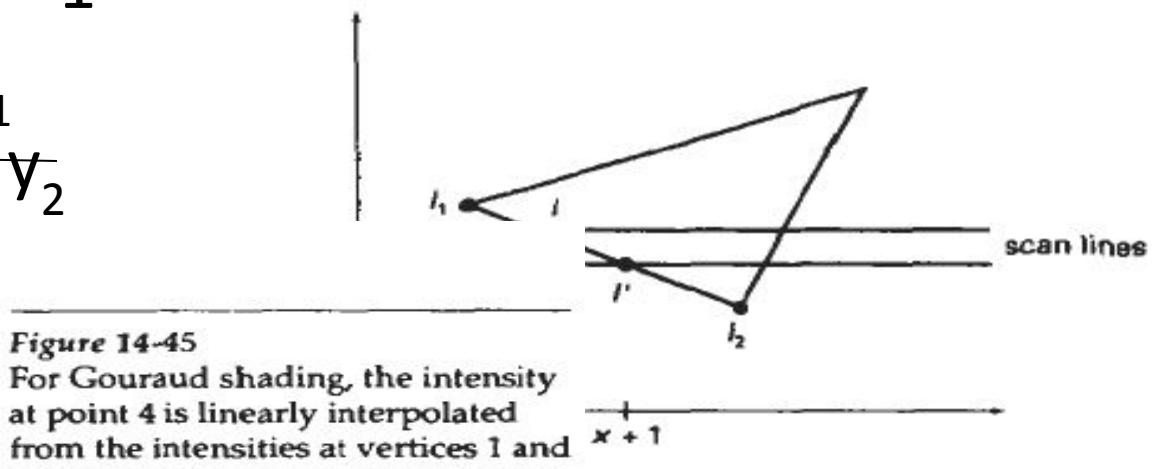


Figure 14-45

For Gouraud shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point p is then assigned an intensity value that is linearly interpolated from

polation of intensity values along a successive scan lines.



Advantages:

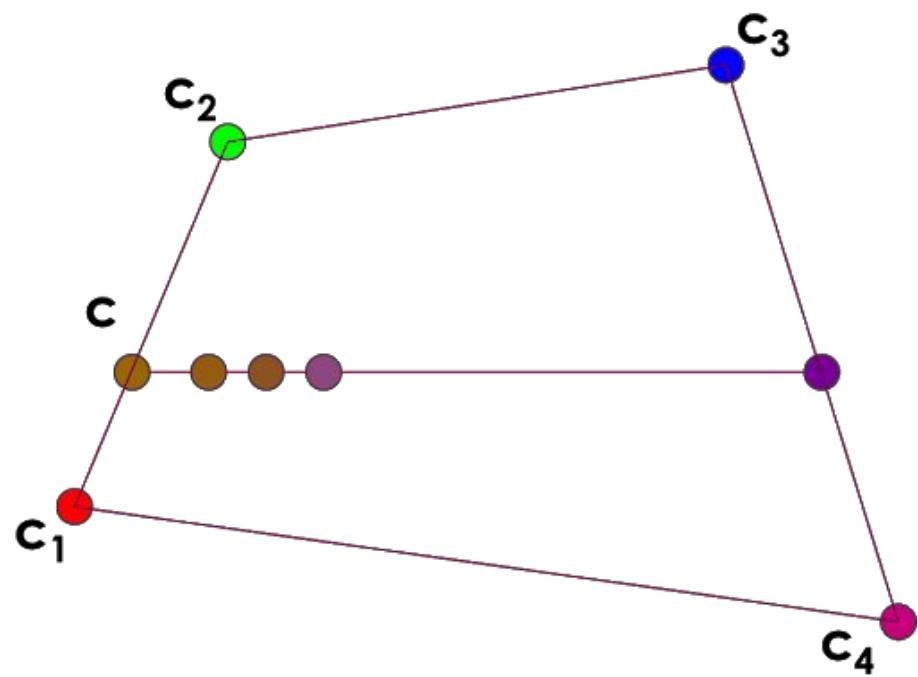
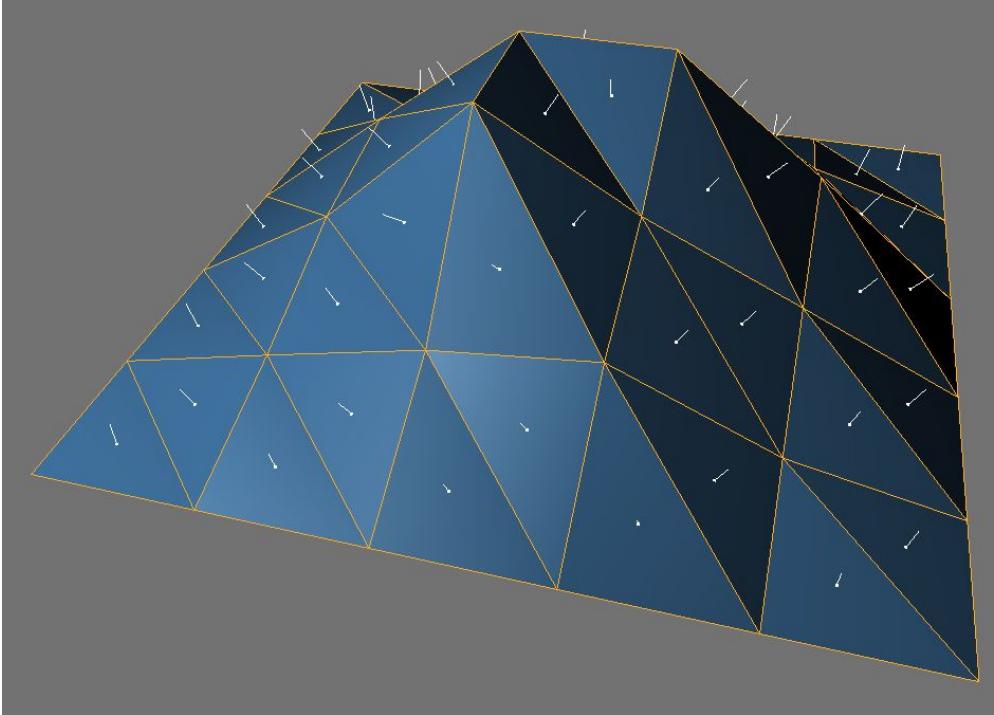
- Removes **discontinuities of intensity at the edge** compared to constant shading model

Disadvantages:

- Highlights on the surface are sometimes displayed with anomalous shapes and linear intensity
- Interpolation **can cause bright or dark intensity streaks called Mach Bands** to appear on the surfaces.

Mach bands can be **reduced** by

- dividing surface into a greater number of polygon faces
- Use Phong shading (requires more calculation)



C. Phong Shading

More accurate method for rendering a polygon surface is to **interpolate normal vector and then apply the illumination model** to each surface point called “Phong Shading” or “**Normal Vector Interpolation Shading**”.

It displays more **realistic highlights** on a surface

Greatly **reduces** Mach band effect.

Steps:

- Determine the **average unit vector normal** at each polygon vertex
- Linearly **interpolate the vertex normal** over polygon surface
- **Apply an illumination model along each scan line** to calculate projected pixel intensities for the surface points

N can be obtained by vertically interpolating between edge end point normal (N_1 and N_2)

$$N = \frac{y - y_2}{y_1 - y_2} \cdot N_1 + \frac{y_1 - y}{y_1 - y_2} \cdot N_2$$

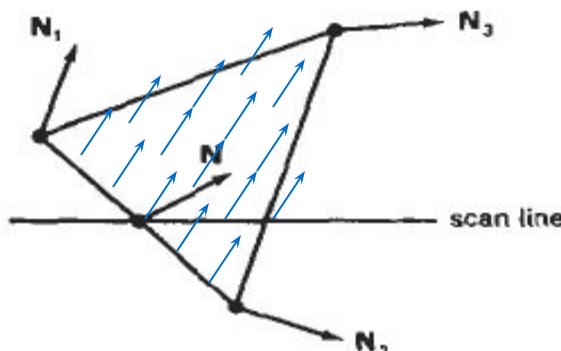


Figure 14-48
Interpolation of surface normals
along a polygon edge

Incremental methods are used to **evaluate normal** between scan lines and along each individual scan line (as in Gouraud) at each pixel position along a scan line the illumination model is applied to determine the surface intensity at that point

Produces **accurate results than the direct interpolation** but it requires considerable **more calculations**

Fast Phong Shading (FPS)

FPS approximates the intensity calculations using a **Taylor** series expansion and **triangular** surface patches

Surface normal at any point (x,y) over a triangle as
 $N = Ax + By + C$

A, B, C are determined from three vertex equations

$$N_k = Ax_k + By_k + C \dots \quad k = 1, 2, 3 \quad (x_k, y_k \text{ vertex position})$$

Omitting reflexivity and attenuation parameters

$$\begin{aligned}
 I_{\text{diff}}(x, y) &= \frac{\mathbf{L} \cdot \mathbf{N}}{|\mathbf{L}| |\mathbf{N}|} \\
 &= \frac{\mathbf{L} \cdot (\mathbf{A}x + \mathbf{B}y + \mathbf{C})}{|\mathbf{L}| |\mathbf{A}x + \mathbf{B}y + \mathbf{C}|} \\
 &= \frac{(\mathbf{L} \cdot \mathbf{A})x + (\mathbf{L} \cdot \mathbf{B})y + \mathbf{L} \cdot \mathbf{C}}{|\mathbf{L}| |\mathbf{A}x + \mathbf{B}y + \mathbf{C}|}
 \end{aligned}$$

Performing the indicated dot products and expanding the vector magnitude yields:

We can write

Where a, b, c, d are used to represent the various dot products e.g. $a = \underline{L} \cdot A$

$| L |$

Finally denominator in eq(i) can be expressed as Taylor series expansion and retain terms up to second degree in x and y. This yields

$$I_{\text{diff}}(x,y) = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0$$

.....(ii)

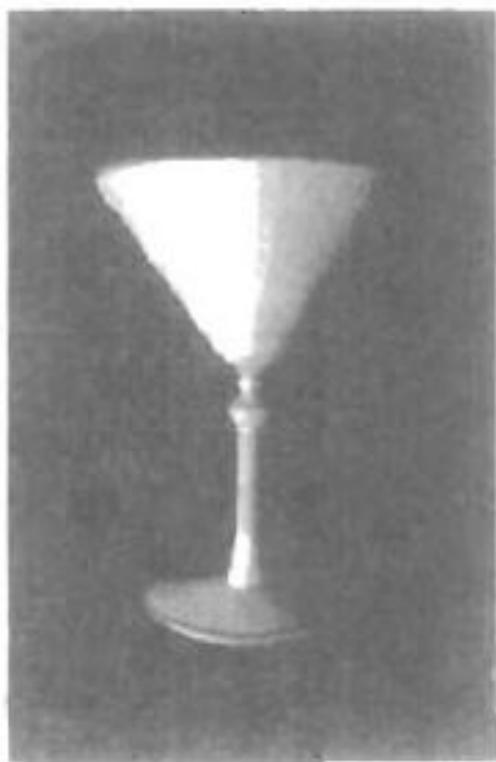
Fast Phong Shading is two times slower than Gouraud shading

Normal Phong shading is 7 times slower than Gouraud

Fast Phong Shading can be extended to include specular reflections



(a)



(b)



(c)

Figure 14-47

A polygon mesh approximation of an object (a) is rendered with flat shading (b) and with Gouraud shading (c).

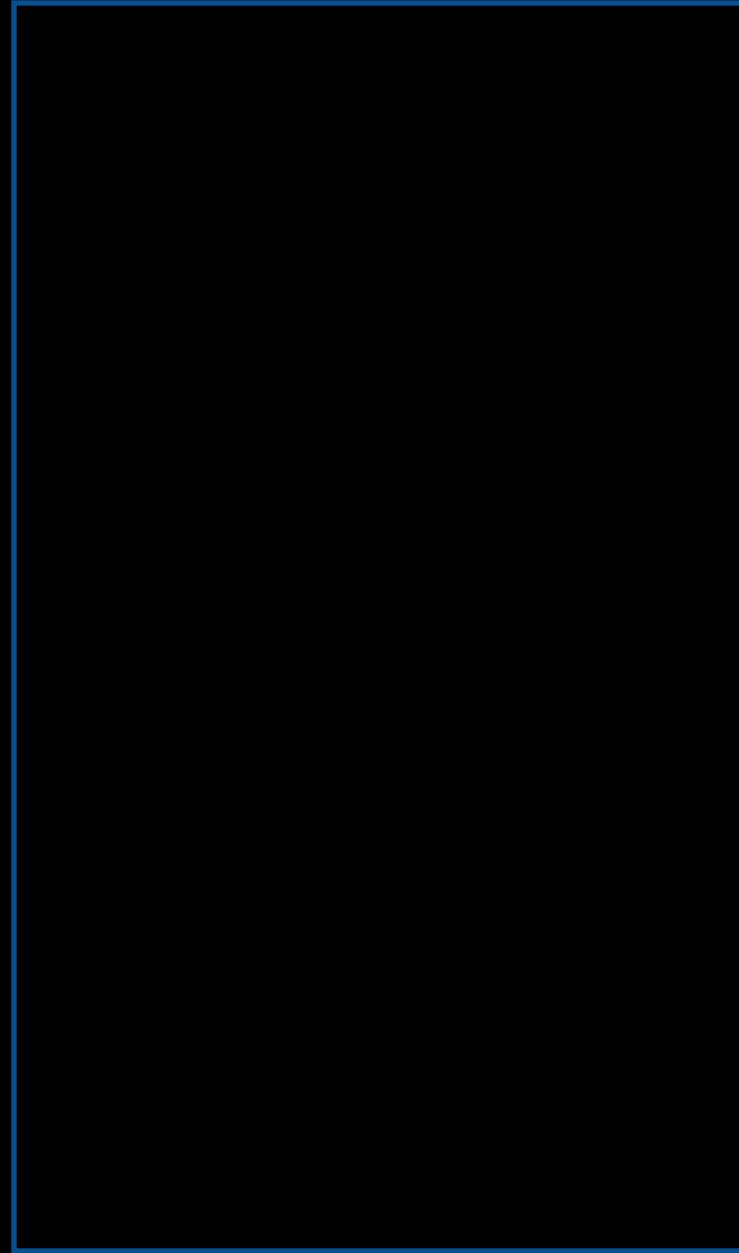
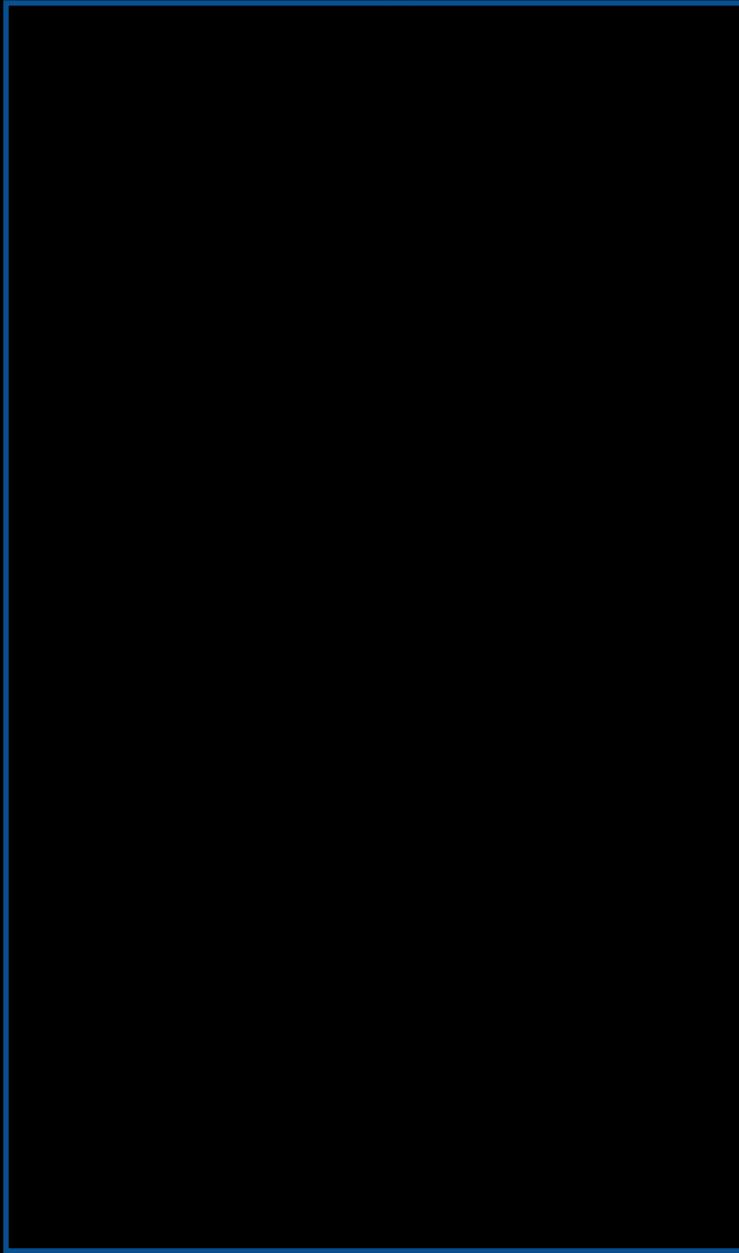
From Computer Desktop Encyclopedia
Reproduced with permission.
© 2001 Intergraph Computer Systems



Flat

Gouraud

Phong

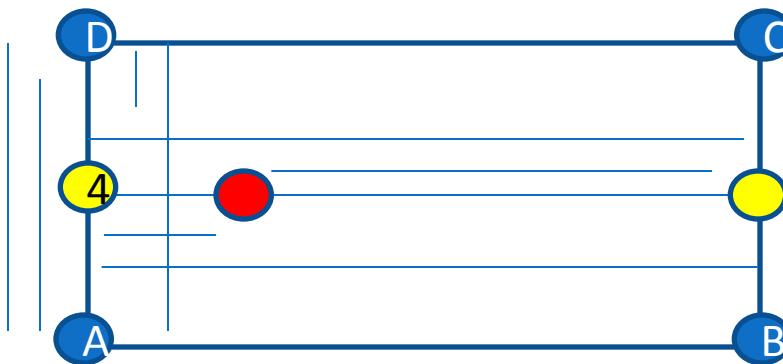


A rectangle plane has vertices A(0,0), B(8,0), C(8,8), D(0,8), find the reflected intensity at point P(4,4) using Gouraud Shading technique.

The average intensities of reflected illumination at the four vertices are :

$$I_A = 6, I_B = 8, I_C = 3, I_D = 5$$

$$I_4 = (-4)/(-8) \times 5 + (-4)/(-8) \times 6 = 2.5 + 3 = 5.5$$



$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} \cdot I_1 + \frac{y_1 - y_4}{y_1 - y_2} \cdot I_2$$

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} \cdot I_4 + \frac{x_p - x_4}{x_5 - x_4} \cdot I_5$$

Graphics File Format

A graphics format **file reader** is responsible for **opening** a file, determining its **validity** and interpreting the **information** contained within it.

A reader may take its input , source image data either **from a file** or **from the data stream of an input device**, such as a scanner or frame buffer.

Graphics File Format

There are two types of reader designs:

- i. Filter
- ii. Scanner.

A filter reads a data source **one character at a time**.

The Scanners **randomly access** across the entire data source.
They can read and reread data **anywhere** within the file.

Filters require small amount of memory while the Scanners require a larger amount of memory.

Reading Graphics File Data

Image data are read from a Graphics File.

Compressed image data is read one byte at a time from file and into memory before it is decompressed.

Uncompressed image data is often stored only as bytes, even when the pixels are two three or four bytes in size.

Writing a Graphics File Data

Inverse of Reading a graphics file

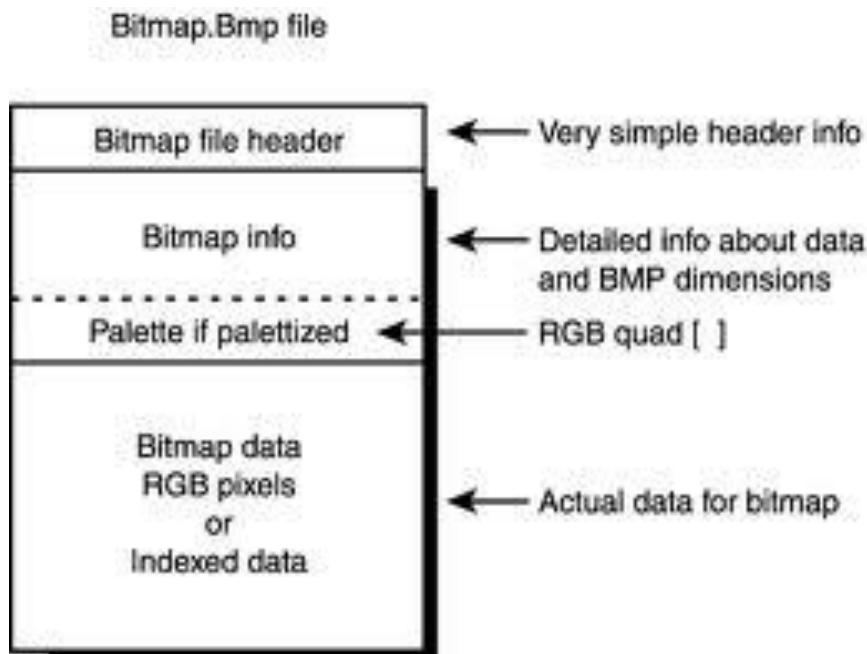
Writers may **send data directly to an output device**, such as a printer or they may create image files and store data in them.

BMP

A Bitmap File (BMP) contains an **exact pixel mapping** of an image which can then be reconstructed by rendering application on the display surface of an output device.

BMP file consists of:

Header Bitmap Data Color Palette other Data



BMP

The Header

Consists of binary or ASCII format data located at the **beginning of** the file., but the information about the bitmap data is stored elsewhere in the file.

A Bitmap Header is composed of

i. File Identifier

A header starts with a unique identification value called a File Identifier, file IP or ID.

ID values are assigned arbitrarily by the creator of the file format, and can be a series of ASCII characters

BMP

ii. File Version

Successive version of bitmap formats differ in header size, bitmap data supported and color capability.

After verifying the file format, an application examine the version value, determine whether it can handle the image data located in the file.

iii. Bitmap Data

In many bit map files Bitmap Data is found immediately after the end of the file header.

Bitmap data is **composed of pixel values**.

BMP

Sequence of values that **logically maps bitmap data** in a file to an image on the display surface of an output device

Pixels on an output device are usually **drawn in scan lines** corresponding to rows spanning the width of the display surface.

Image data is usually first assembled in one or more blocks of memory.

Exactly how the data is arranged then depends on a number of factors, including the **amount of memory** installed, the **amount available** to the application, and the **specifics of the data acquisition** or file write operation in use.

TIFF

Acronym of **Tag Image File Format**

TIFF specification was intended for **storing black and white images**, created by scanners

There are three sections in a TIFF file:

- **Image File Header (IFH)**
- **Image File Directory (IFD)**
- **Bitmap Data**

TIFF

Image File Header is always at the **beginning** of the TIFF file, occupying the first eight bytes in every TIFF file.

IFH contains three fields of information

An identifier, used as byte order identifier and having a size of a word

A version , contains version number of the TIFF format.

TIFF

IFD offset is a 32 bit value. It is offset position of the first image file directory in TIFF file. We use this value to identify the starting position of the image file information

One way to verify whether the file is a TIFF file is by checking the first four bytes of the file

Image File Directory is used to describe the bitmapped data to which it is linked

It contains information of **height , width, and depth of image colors** and **compression types** used and doesn't have a fixed size and position in a TIFF file

Run Length Encoding

An effort to **reduce memory requirements** in raster system , frame buffer is organized as a linked list and intensity information is encoded

Each scan line is stored as a set of integer pairs

It maintains the record of **intensity** and the **numbers of pixels** and the surrounding pixels **with the same intensity values**

One number of each pair indicates an intensity value, and the second number specifies the number of adjacent pixels on scan line that are to have that intensity

Run Length Encoding

Suppose we have scan lines 0, 12, 20 and 40. If the run length encoding for scan line 0 is

Intensity	Run-Length
0	30

It suggests that :

It means that 30 pixels across this line has intensity 0

Run Length Encoding

Suppose we have scan lines 0, 12, 20 and 40. If the run length encoding for scan line 0 is

Intensity	Run-Length
1	1
0	28
1	1
1	1

It suggests that :

First pixel has intensity 1

Next 28 pixels have intensity 0

Last (30^{th}) pixel has intensity 1

Run Length Encoding

Saves storage space if a picture is to be constructed mostly
with long runs of single color each

Disadvantages are **storage requirements actually increase** as the length of runs decrease.

Cell Encoding

Another approach that encodes the raster as a set of rectangular areas

Graphical Data Structure

Graphical Data Structure is required mainly for 2 reasons:

- to **store** graphics data in memory
- to **facilitate data manipulation** and to make its effect quickly visible

The extra tasks of GDS are:

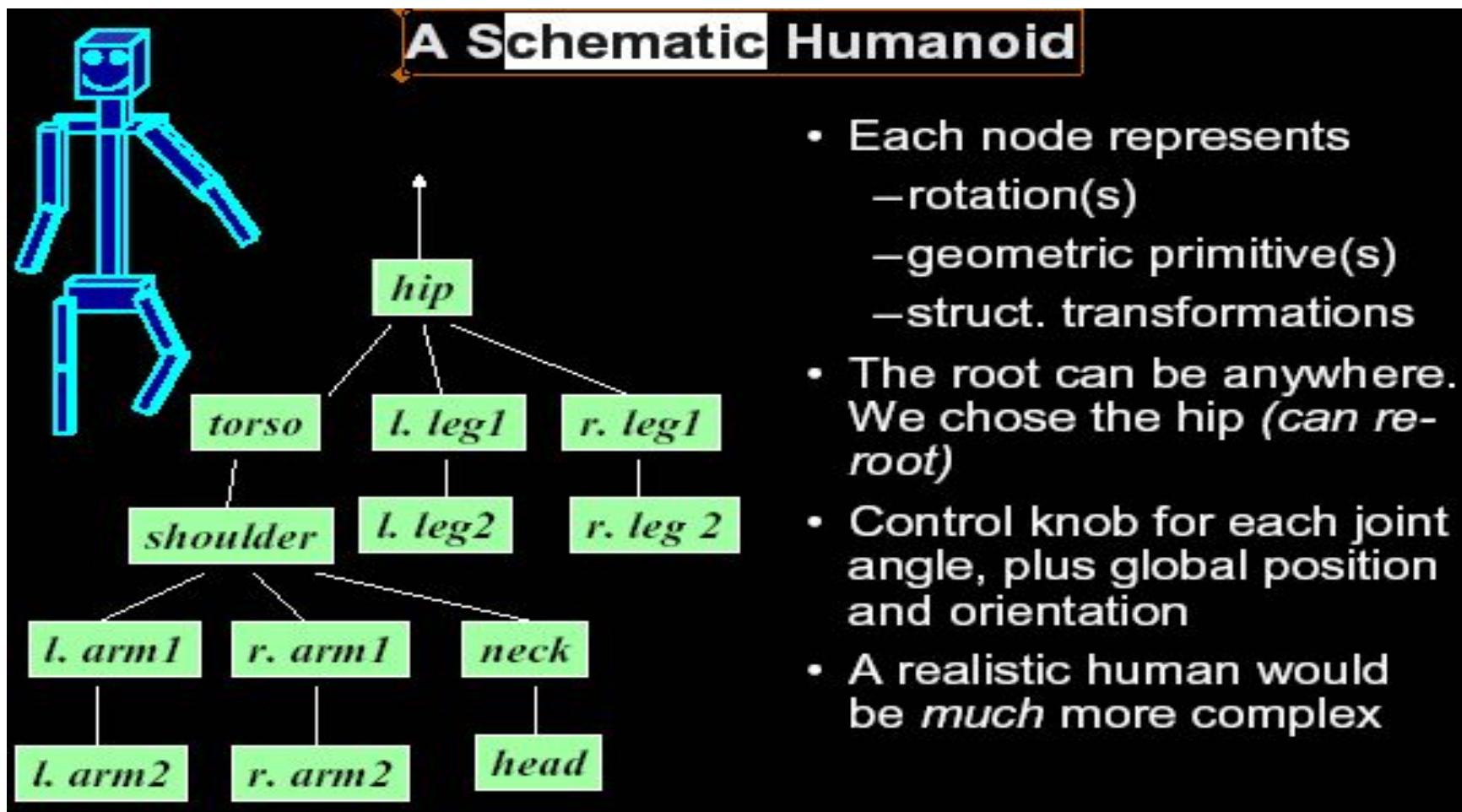
- It should describe pictures **in terms of primitives** (points, lines etc)
- It should **Maintain inter-relationship** between the primitives
- Transformation of **whole /part** of the picture
- Modification of picture by **deleting or adding** one or more elements

Graphical Data Structure

Depending upon the type of the object that is to be displayed the purpose of DS can be chosen.

If a graphical object is a bicycle, then a tree data structure can be implemented to store its data

Bicycle



Graphical Data Structure

Basic Data Structures for Graphics

Static Array

Homogenous data structure

DIMENSION REALARRAY(100) FORTRAN

a = array[1 ... 100] of real PASCAL

float a[100]; C

Operations: Traversal , Insertion, Deletion

Description of objects with vertices can be represented with an array for cubes

Structured Data types

A **structure** combines other data types

in Pascal its called Record

in C its called Struct

```
Struct A{  
    data_type 1  var1;  
    data_type 1  var1;  
    -----  
};
```

Graphical Data Structure

```
struct point{  
    float x,y,z;  
};  
  
struct edge{  
    struct point v1, v2;  
};  
struct object{  
    int no_of_edges , no_of_vertices, no_of_faces;  
    struct point vertices[100];  
    struct edge edges[100];  
    int faces[100][100];  
};
```

Graphical Data Structure

Pointers

Used to effectively represent complex data structures and to deal more concisely with arrays

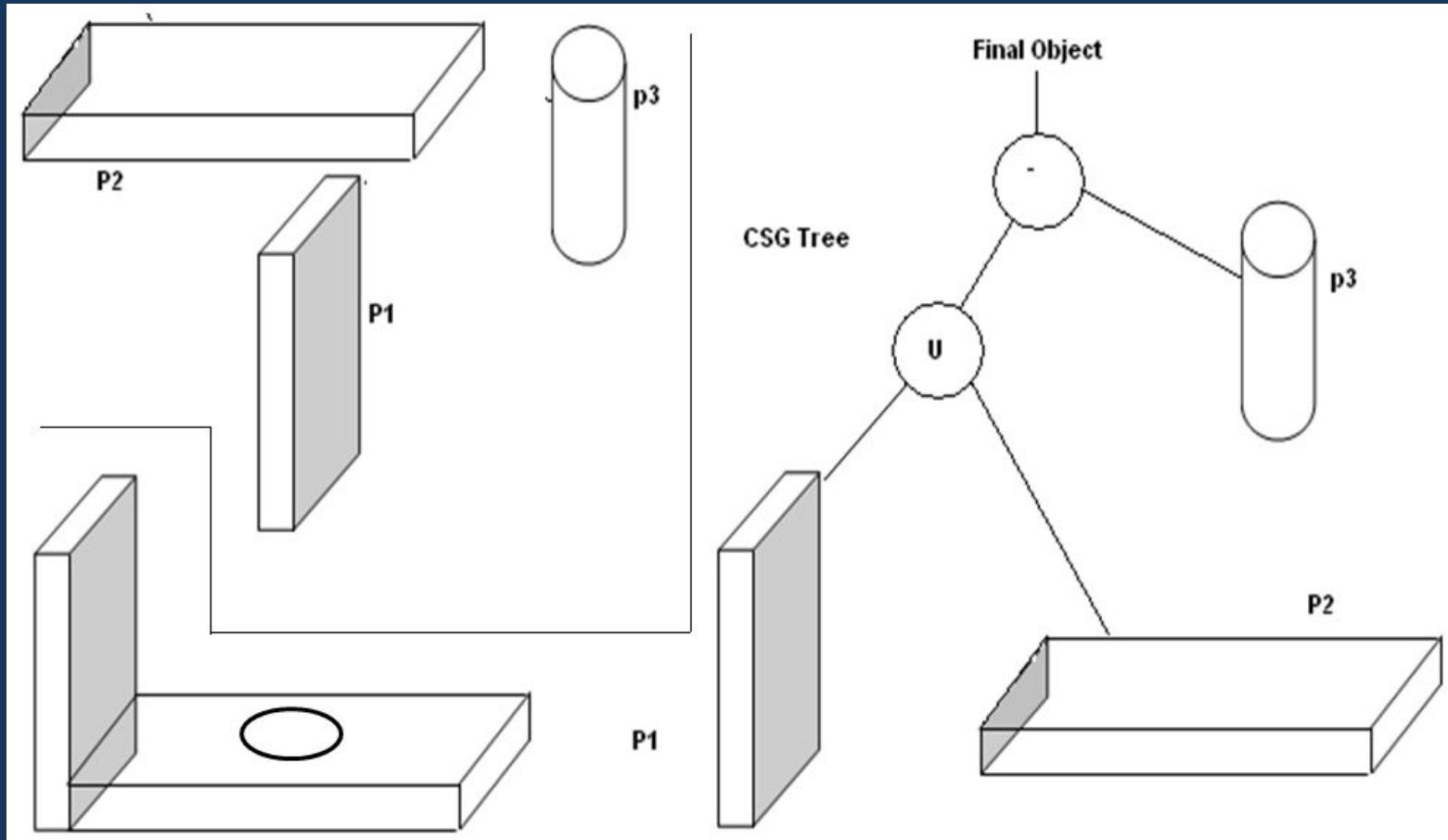
```
struct edge{  
    struct point *pv1, *pv2;  
}e1;  
e1.pv1 = &v1;  
e1.pv2 = &v2;
```

Linked Lists

It is a collection of data elements or nodes placed in a linear order by pointers.

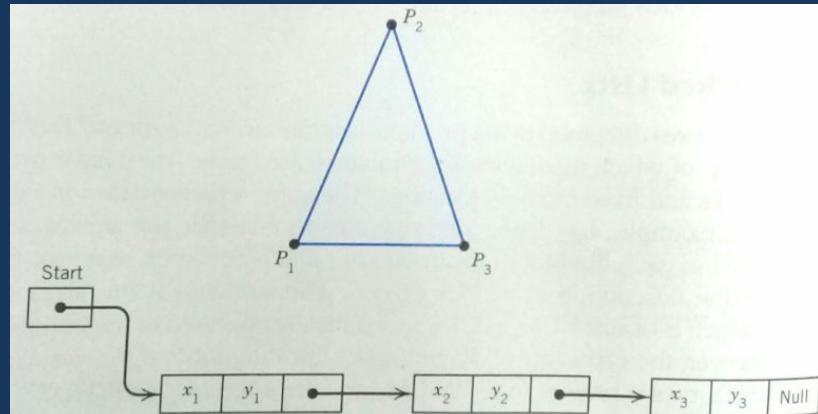
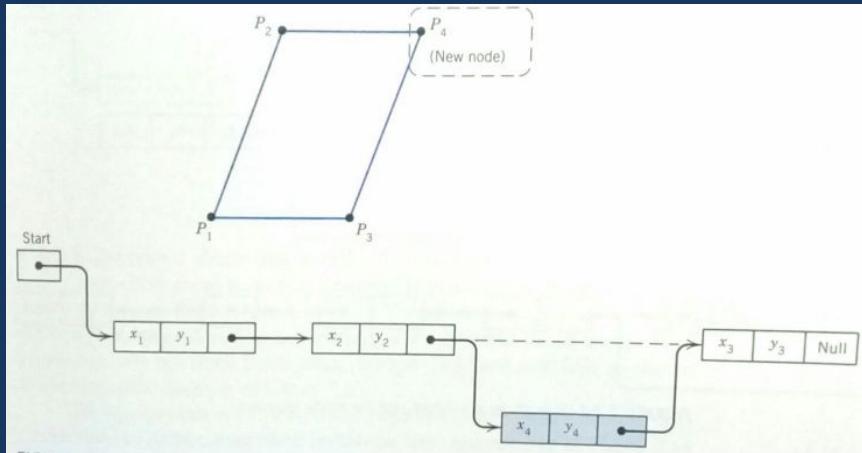
Graphical Data Structure

Tree It is a non linear data structure used to represent data having hierarchical relationship it contains a finite number of nodes and one root



Graphical Data Structure

Linked list data structures can be used to represent object descriptions



OPENGL

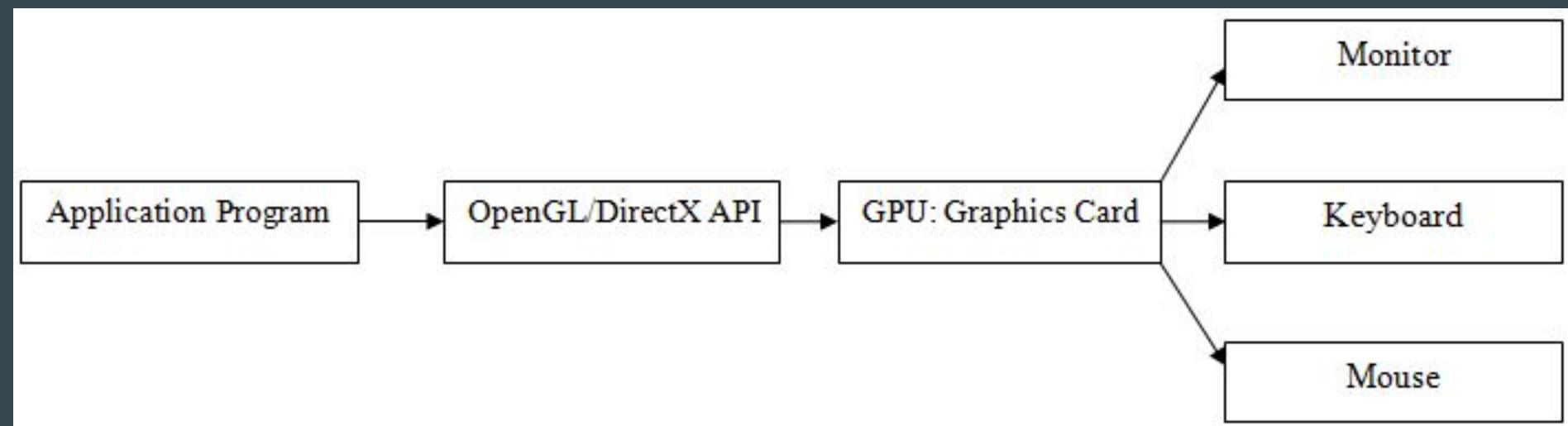
An **operating system and hardware platform independent graphics library** designed to be easily portable yet rapidly executable.

It brings a standard 3D graphics library with the hardware enhanced ability to perform **lighting , shading, texture mapping, hidden surface removal and animation** on to the windows platform

Open GL is available on a variety of hardware platforms and operating systems.

OpenGL was written with the express intention of becoming a **thin software interface to underlying graphics hardware** an arrangement of proven success in the graphics workstation market.

OPENGL



OPENGL

A **software interface** to graphics hardware

About **250 distinct commands**

200 in core OpenGL and 50 in OpenGL Utility Library to produce interactive 3D applications

Designed as **hardware independent interface** to be implemented on different hardware platforms •••

To achieve these **no windowing tasks** or obtaining user inputs are included in open GL

Desired models are built up from a small set of geometric primitives – points, lines, polygons

Features of OpenGL:

Texture Mapping :

The ability to apply an image to a graphics surface, this technique is used to rapidly generate realistic images without having to specify an excessive amount of detail regarding pixel coordinates, textures etc

Z-Buffering:

The ability to calculate the distance from the viewer's location. this makes it easy for the program to automatically remove surfaces or parts of surfaces that are hidden from view

Double Buffering:

Support for smooth animation using double buffering. A smooth animation sequence is achieved by drawing into the back buffer while displaying the front buffer and then swapping the buffers when ready to display the next animation sequence

Lighting Effects:

The ability to calculate the effects on the lightness of a surface's color when different lighting models are applied to the surface from one or more light sources

Features of OpenGL:

Smooth Shading:

The ability to calculate the shading effect that occurs when light hits a surface at an angle and results in subtle color differences across the surface. This effect is important for making model look realistic

Material Properties:

The ability to specify the material properties of a surface. These properties modify the lighting effects on the surface by specifying such this as dullness or shininess of the surface

Alpha Blending:

The ability to specify alpha or opacity value in addition to the regular red , green, blue values. The alpha component is used to specify opacity, allowing the full range from completely transparent to totally opaque. When used in combination with z buffer, Alpha blending gives the effect of being able to see through objects

Developer's Advantage:

Industry Standard:

the OpenGL architecture Review board an independent consortium guides the OpenGL specification OpenGL is the only **true open vendor-neutral, multiplatform graphics standard** with a broad industry support

Stability:

Updating to OpenGL specification are **carefully controlled** and updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure the viability of existing applications

Portability:

applications produce consistent visual display result **on any OpenGL API compliant hardware** regardless of OS or windowing system. So once a program is written for any platform, it can be ported for other platforms as well.

Developer's Advantage:

Evolving:

New hardware innovations are accessible thru the API via the OpenGL extension mechanism. Innovations are phased in to enable developers and hardware vendors to incorporate new features into their product release cycles.

Scalability:

OpenGL applications can be scaled to any class of machine, everything from consumer electronics to PCs, workstations, Super Computers

• • •

Ease of Use:

Efficient OpenGL routines typically result in applications with fewer lines of code than programs created with other graphics libraries or packages. OpenGL driver encapsulates the information about the underlying hardware so the application programmer does not need to be concerned about having to design for specific hardware features.

History

OpenGL was first created as an open and reproducible alternative to **Iris GL** which had been the **proprietary graphics API on Silicon Graphics workstations.**

Although OpenGL was initially similar in some respects to IrisGL the lack of a formal specification and conformance tests made Iris GL unsuitable for broader adoption.

Mark Segal and Kurt Akeley authored the OpenGL 1.0 specification which tried to **formalize the definition of a useful graphics API** and made cross platform non-SGI 3rd party implementation and support viable.

One notable omission from version 1.0 of the API was texture objects.



IrisGL had definition and bind stages for all sorts of objects including materials, lights, textures and texture environments. OpenGL avoided these objects in favor of incremental state changes with the idea that collective changes could be encapsulated in display lists. This has remained the philosophy with the exception that texture objects (**glBindTexture**) with no distinct definition stage are a key part of the API.

History

OpenGL has been through a number of revisions which have predominantly been incremental additions where extensions to the core API have gradually been incorporated into the main body of the API.

OpenGL 1.1: added the `glBindTexture` extension to the core API.

OpenGL 2.0 : addition of the OpenGL **Shading Language** (also called GLSL), a C like language with which the transformation and fragment shading stages of the pipeline can be programmed.

OpenGL 3.0 : adds the concept of **deprecation**: marking certain features as subject to removal in later versions. GL 3.1 removed most deprecated features, and GL 3.2 created the notion of core and compatibility OpenGL contexts.

Official versions of OpenGL released to date are 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5.

Model-View-Controller Architecture

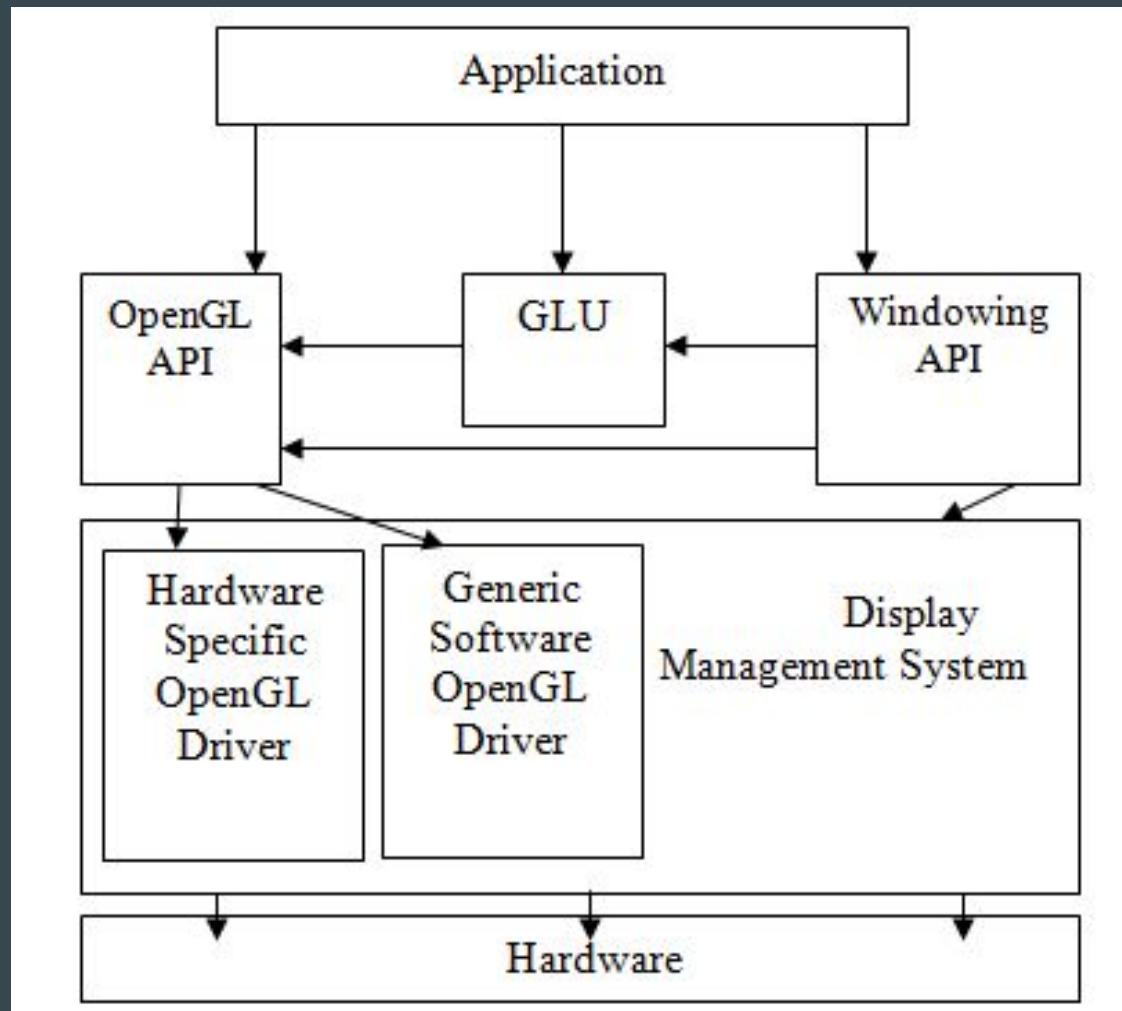
Interactive program design involves breaking the program into three parts:

The **Model**: all the data that are unique to the program reside there. This might be game state, the contents of a text file, or tables in a database.

The **View** of the **Model**: it is a way of displaying some or all of the data to user. Objects in the game state might be drawn in 3D, text in the file might be drawn to the screen with formatting, or queries on the database might be wrapped in HTML and sent to a web browser.

The **Controller**: the methods for user to manipulate the model or the view. Mouse and keystrokes might change the game state, select text, or fill in and submit a form for a new query.

Architecture



OPENGL

GL

Core Open GL
APIs

• • •

OPENGL

```
#include <necessary header files>
main(){
    InitializeWindow();
    Rendering operations(); •••
    UpdateWindowAndCheckForEvents();
}
```

OpenGL Command Syntax

Uses the prefix gl and initial capital letters for each word making up the command name

e.g. glClearColor(-----);

glVertex3f(-----);

Constants begin with GL

...

Use all capital letters and underscores to separate words e.g.
GL_COLOR_BUFFER_BIT

Data types

<u>Data type</u>	<u>Corresponding C language</u>	<u>OpenGL TypeDefinition</u>
8 bit integer	signed char	GLbyte
16 bit integer	short	GLshort
32 bit integer	int or long	GLint, GLsizei
32 bit float	float	GLfloat
64 bit float	double	GLdouble

GL Related Libraries

OpenGL Utility Library (GLU):

Contains several routines that use lower level OpenGL Commands to perform tasks as setting up matrices, for specific viewing orientations and projections etc. e.g.

gluPerspective(.....);

•••

OPENGL

OpenGL Utility Toolkit (GLUT): A **window system independent** toolkit written by Mark Kilgard to hide the complexities of differing system APIs

OpenGL contains **only rendering commands** and no commands for opening windows or reading events from keyboard or mouse

• • •

OPENGL

GLUT provides several routines for opening windows detecting input and creating complicated 3D objects like **sphere** , **torus** , **teapot**

GLUT routines use the prefix **glut**

e.g. `glutCreateWindow(...);`
 `glutInit(...);` •••
 `glutMouseFunc.....);` `glutKeyboardFunc.....);`

Windows Management

Five routines perform tasks necessary for initializing a window

`glutInit(int *argc, char **argv)`

Initializes glut and processes command line arguments

`glutInit();`

- should be called before any other GLUT routine

`glutInitDisplayMode(unsigned int mode)`

specifies whether to use an RGBA or color-index model

specifies whether to use a single or double buffered window and specify a depth buffer

`glutInitDisplayMode(GLUT_SINGLE|GLUT_RGBA|GLUT_DEPTH);`

OPENGL

`glutInitWindowPosition(int x, int y)`

- specifies screen location for the upper left corner of your window

`glutInitWindowSize(int x, int y)`

- specifies the size in pixels of your window

`glutCreateWindow(char *string)` •

- creates a window with the name that you supplies as the string (pointer) variable

The Display Callback

glutDisplayFunc(void (*func)(void))

- most important event callback function
- Whenever GLUT determines that the contents of the window need to be redisplayed the call back function registered by glutDisplayFunc() is executed
- All the routines needed to redraw the scene are placed here

•••

OPENGL

glutMainLoop()

- This is an infinite loop
- All the windows that have been created are now shown infinitely

• • •

Setting up 2D Projection

```
void myReshape(int w, int h) {                                // window is reshaped
    glViewport (0, 0, w, h);          // update the viewport
    glMatrixMode(GL_PROJECTION);    // update projection
    glLoadIdentity();
    gluOrtho2D(0.0, 1.0, 0.0, 1.0);      // map unit square to viewport
    glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();           ••• // request redisplay
}
```

Handling input events

Use the following routines to register callback commands that are invoked when specified events occur

`glutReshapeFunc(void (*func)(int w, int h))`

- Indicates what action should be taken when the window is resized

`glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))`

`glutMouseFunc(void (*func)(int button, int state, int x, int y))`

- Allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released

Managing a background process

`glutIdleFunc((*func)(void))`

- Used to specify a function that's to be executed if no other events are pending
- This routine takes a pointer to a function as its only argument

• • •

OPENGL

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/glu.h>

void main(int argc, char **argv){
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(100,100);
    glutCreateWindow("Open GL Lab Work");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

OPENGL

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/glu.h>

//CLEARING THE BACKGROUND
void display(){
    glClearColor(1.0,0.0,1.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

void main(int argc, char **argv){
    glutInit(&argc,argv);          •••
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(100,100);
    glutCreateWindow("Open GL Lab Work");
    glutDisplayFunc(display);
    glutMainLoop();
}
```

PLOTTING A POLYGON: ADDED INSIDE DISPLAY FUNCTION

```
glShadeModel(GL_SMOOTH);  
glBegin(GL_POLYGON);  
    glVertex3f(0.25, 0.25, 0.0);  
    glVertex3f(0.75, 0.25, 0.0);  
    glVertex3f(0.75, 0.75, 0.0);  
    glVertex3f(0.25, 0.75, 0.0); •••  
glEnd();
```

OPENGL

// TO SPECIFY FILL COLOR

//ADDED INSIDE DISPLAY FUNCTION

```
glClearColor(1.0,0.0,1.0,1.0);
```

```
glClear(GL_COLOR_BUFFER_BIT);
```

```
glShadeModel(GL_FLAT);
```

```
glColor3f(0.0,1.0,0.0);
```

• • •

Getting input from a Mouse

```
GLfloat x = 0.15;

void display(){
    glClearColor(1.0,1.0,1.0,1.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,1.0);
    glBegin(GL_POLYGON);
    glVertex3f(0.25+x,0.25,0.0);
    glVertex3f(0.75+x,0.25,0.0);
    glVertex3f(0.75+x,0.75,0.0);
    glEnd();
    glFlush();
}

void trans(){
    x = x - 0.001;
    glutPostRedisplay();
}
```



OPENGL

```
void reshape(int w, int h){  
    glViewport(0,0,w,h);  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
    glOrtho(0,50,0,50,-1,1);  
    //glOrtho(left,right,bottom,top,near,far);  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
}  
  
void main(int argc, char **argv){  
    glutInit(&argc,argv);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
    glutInitWindowPosition(200,200);  
    glutInitWindowSize(200,200);  
    glutCreateWindow("Triangle");  
    glutDisplayFunc(display);  
    glutReshapeFunc(reshape);  
    glutMainLoop();  
}
```

GL_PROJECTION : Applies subsequent matrix operations to the projection matrix stack.

GL_MODELVIEW : Applies subsequent matrix operations to the modelview matrix stack.

OPENGL

...

OPENGL

```
void gluLookAt( GLdouble eyex, GLdouble eyey, GLdouble eyez,  
    GLdouble centerx, GLdouble centery, GLdouble centerz,  
    GLdouble upx, GLdouble upy, GLdouble upz );
```

Parameters

eyex, eyey, eyez The position of the eye point.

centerx, centery, centerz The position of the reference point.

upx, upy, upz The direction of the up vector.

•••

OPENGL

The **gluLookAt** function creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an up vector.

• • •

OPENGL

...

Graphical Languages

Vector and raster displays are two substantially different hardware technologies for creating images on the screen

Raster display now dominant hardware technology because they support several modern applications like:

- a. **fill area with uniform colors repeated patterns** in two or more colors vector can only simulate filled areas with closely spaced sequences of parallel vectors
- b. raster display **stores images in pixel forms** and can read or write moved copies or any manipulations

Software Standards (Need for Machine Independent Graphical Languages)

Primary goal of standardized graphics software is **portability**

Why machine independent graphics language?

When packages are designed with standard graphics functions **software can be moved easily** from one hardware to another without standards, program designed for one hardware often cant be transferred to another system with out extensive rewriting of the programs.

Software Standards (Need for Machine Independent Graphical Languages)

In international and national standards organizations in many countries have cooperated in an effort to develop **a generally accepted standards** for computer graphics

“Graphical Kernel System(GKS)” was adopted as the first graphics software standard by the international standard organization and others like ANSI

Original GKS was 2D and later on 3D extension was developed

Software Standards

(Need for Machine Independent Graphical Languages)

Second standard “**Programmers Hierarchical interactive graphics standard** (PHIGS) is an extension of GKS with increased capabilities for **object modeling, color specifications surface rendering** and **picture manipulations**.

An extension of PHIGS called PHIGS+ was developed to provide **3D surface shading capabilities** standard graphics functions are defined as a set of specifications that is independent of any programming language

Language binding

Gives the **syntax for accessing the various standard graphics functions** from this language

e.g. PHIGS, GKS function for specifying a sequence of n-1 connected 2D straight line segment polyline (n,x,y)

- FORTRAN polyline can be drawn with

GPLCALLGPR (n , x , y) where x, y are end points

- C

gpolyline (n , pts) where pts is list of end points

PHIGS

PHIGS doesn't provide a standard methodology for a **graphical interface** to output devices nor for storing and transmitting pictures

Separate standards developed

Device interface by **Computer Graphics interface (CGI)**
Computer Graphics Metafile (CGM) for achieving and transporting pictures

PHIGS workstation

A PHIGS workstation can be a single output a single input device an combination of input and output devices a file or even a window displayed on a video monitor

```
OpenPhigs(errorFile, memorySize)
OpenWorkstation(ws, Connection, type) {
    Create and display picture
}
CloseWorkstation(ws)
ClosePhigs
```

Parameter errorfile is to contain any error message that are generated
memoriesize specifies the size of the internal storage area
Workstation identifier (as integer) as ws
connection states the access mechanism for the workstation
Type specifies the particular category such as input, output device a combination
“out in” device or an input or output metafile

1. PHIGS concept and graphical workstation

PHIGS is a high level graphics library **with over 400 functions.**

It allows an application programmer **to describe a model of a scene, to display the model on workstation, to manipulate and to edit the model interactively**

The models are stored in a graphical database known as **centralized structure store (CSS).**

The fundamental entity of data is a **structure element** and these are grouped together into units called **structures.**

Structures are organized as acyclic directed graphs called **structure networks**

PHIGS workstation

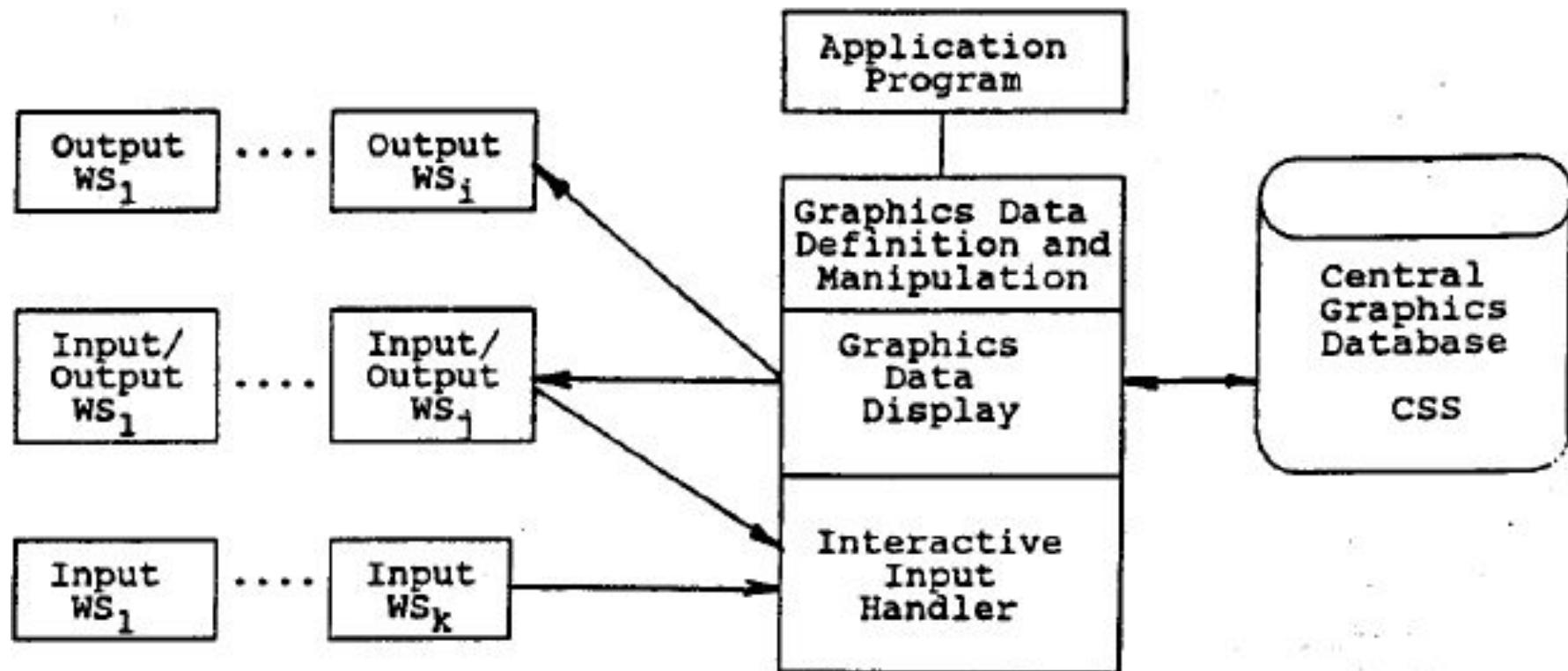


Fig. 1 Structure of the PHIGS

PHIGS workstation

A **PHIGS workstation** represents a unit consisting of zero or one **display surfaces** and zero or more **input devices** such as keyboard, tablet , mouse and light pen.

The workstation presents this devices to the application program as a configuration of abstract devices thereby **shielding the hardware peculiarities**

Workstation type:

OUTPUT – supports output only

INPUT – supports input only

OUTIN – supports output and input

MO – supports output graphical and application data to the external storage

MI- supports input graphical and application data from the external storage to the application

PHIGS

Picture generated by PHIGS are build up of basic pieces called output primitives. Output primitives are generated from structure elements by structural traversal.

POLYLINE set of co-

POLYMARKER set of

POLYMARKER 3

TEXT character str

modeling coordinate

ANNOTATION TEX

x y plane parallel t

FILL AREA single p

uniform color, pat

FILL AREA SET a se

FILL AREA

CELL ARRAY two d

GENERALIZED DRA

capabilities of a w

circular arcs, ellip

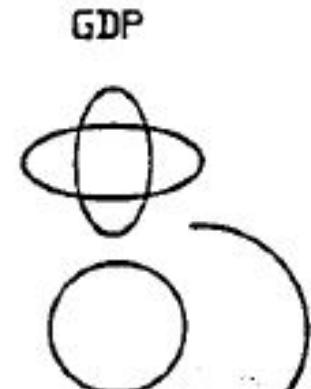
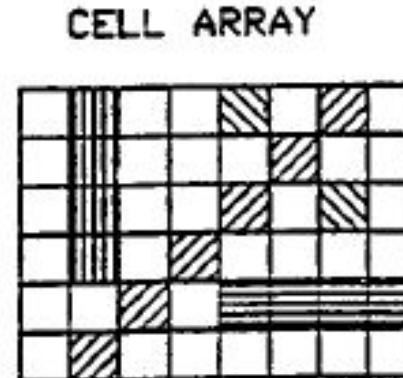
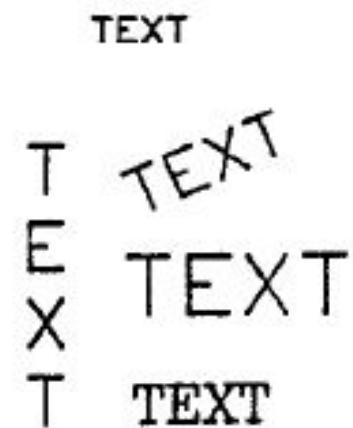
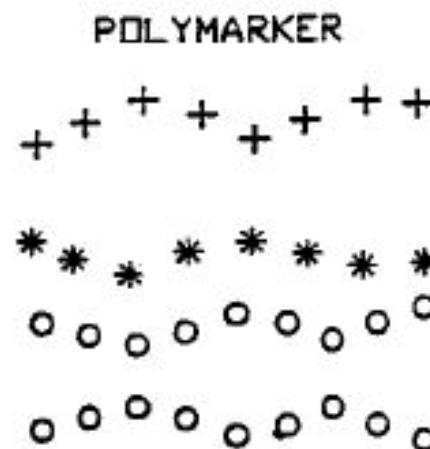
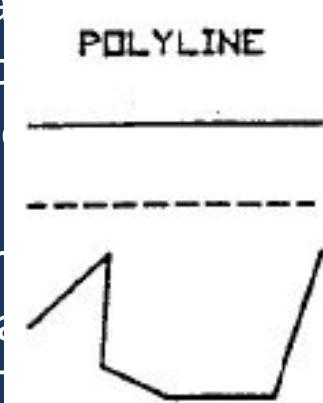


Fig. 2 Examples of output primitives

2. Structure Entity

A structure element is the fundamental entity of data. Structure elements are used to represent application specified graphics data for output primitives, attribute selections, modeling transformations and clipping , invocation of other structures, and to represent application data.

The following types of structure elements are defined in PHIGS

Output primitive structure elements

Attribute specification structure elements

Modeling transformation and clipping structure elements

Control structure element

Editing structure element

Generalized structure element

Application data

Graphical output

3. Structure and structure networks

PHIGS supports the storage and manipulation of data in CSS the CSS contains graphical and application data organized into units called **structures** which may be related each other hierarchically to form structure networks

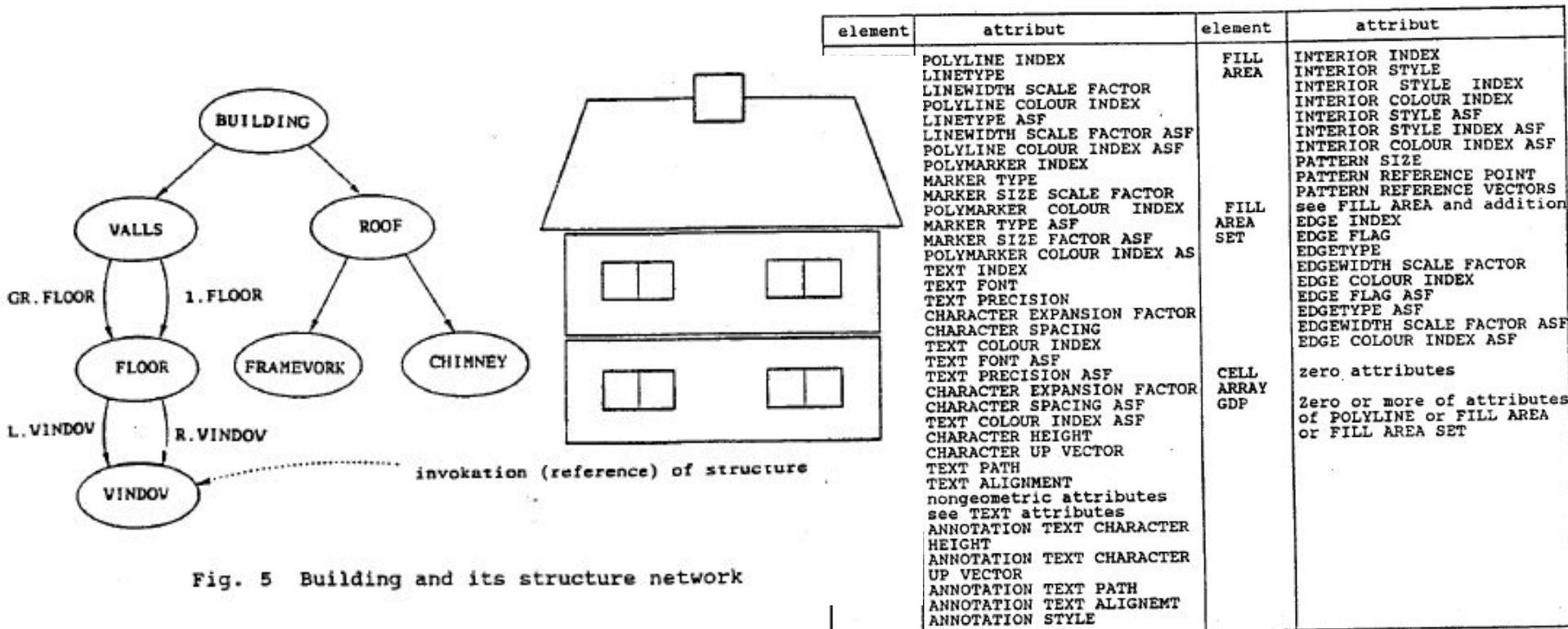


Fig. 5 Building and its structure network

Graphical Output

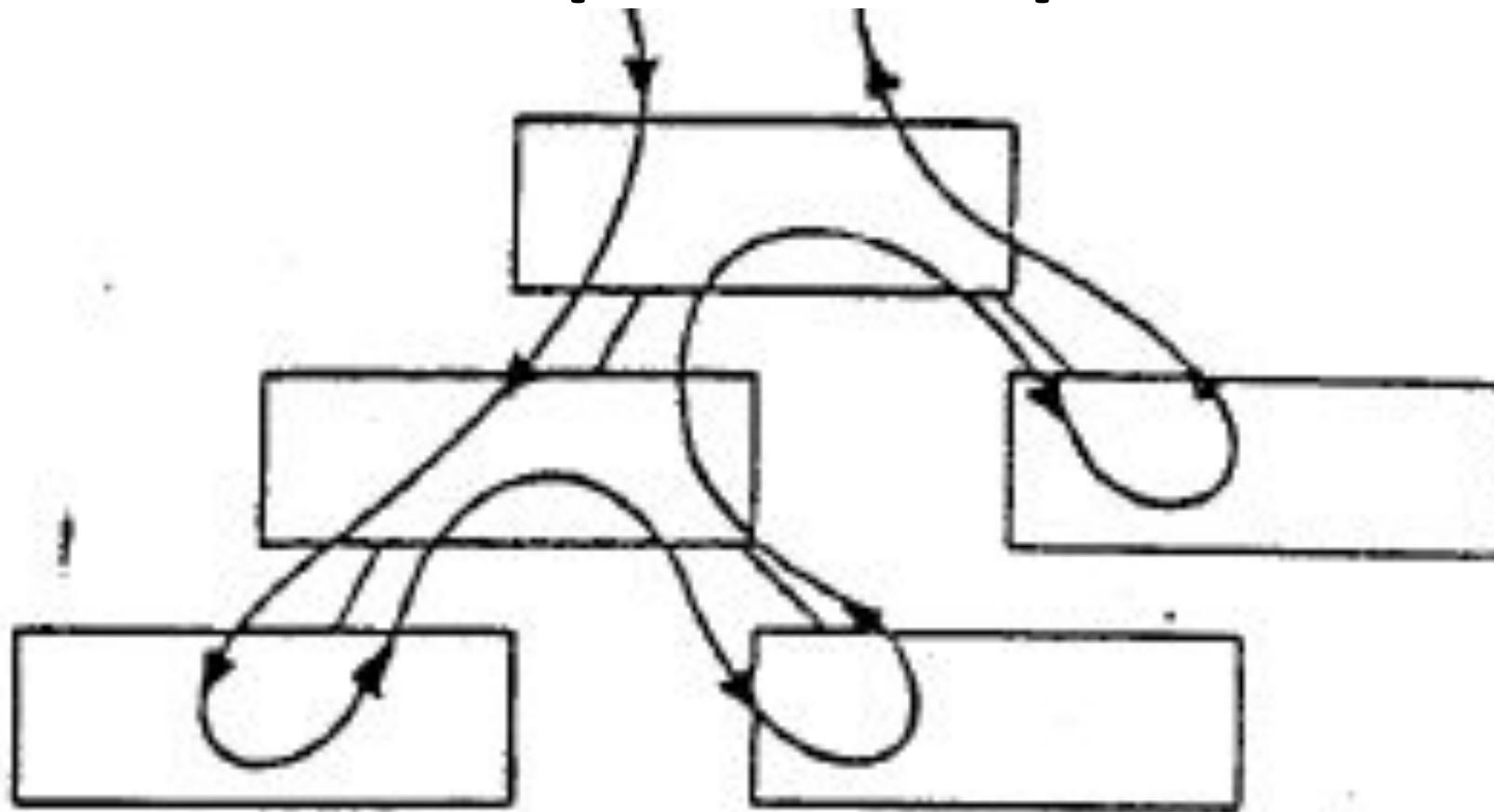


Fig. 6 Traversal process

PHIGS

Structure editing

Each element within a structure can be accessed and modified individually with editing functions to inset new elements, replace elements with new structure elements, delete structure elements.

Functions for positioning element pointer

SET ELEMENT POINTER – set to an absolute position

SET OFFSET ELEMENT POINTER – set relative to current position

SET ELEMENT POINT AT LABEL – set a position of the specified label structure element

The edit mode defined by the function SET EDIT MODE defines whether new elements replace the element pointed to by the element pointer or are added after the element pointed to by the element pointer

Functions for editing:

COPY ALL ELEMENTS FROM STRUCTURE- copy all the elements of a structure into the open structure

DELETE ELEMENT – delete element at which the element pointer is pointing

DELETE ELEMENT RANGE- delete a group of elements between two element positions

DELETE ELEMENT LABELS-delete group of elements delimited by labels

EMPTY STRUCTURE-delete all elements of the structure

PHIGS

Manipulation of structures in CSS

Operations for manipulation of structure

DELETE STRUCTURE- delete structure ;and all references to it

DELETE ALL STRUCTURES- delete all structure from the CSS

DELETE STRUCTURE NETWORK-delete the indicated structure and all its ancestors

CHANGE STRUCTURE IDENTIFIER-change identifier specified structure

CHANGE STRUCTURE REFERENCES-change all references of specified structure

ELEMENT SEARCH –search within a single structure for an element of a particular element type

Structure archival and retrieval

OPEN ARCHIVE FILE , CLOSE ARCHIVE FILE-initiates or terminates access to archive file

ARCHIVE ALL STRUCTURES-storing of structure to archive file

RETRIEVE ALL STRUCTURES- recovers structures from an archive file

DELETE STRUCTURES FROM ARCHIVE-deletes structure in an archive file

4. Coordinate systems and transformations

Structures represent parts of a hierarchical model of modeling scene. Each of these parts has own world space represented by modeling coordinate system .

The relative positioning of the separate parts is achieved by having a single World coordinate space onto which all the defined modeling coordinate systems are mapped by a composite modeling transformation during the traversal process

The world coordinate space can be regarded as a **workstation independent abstract viewing space**

Transformations perform mapping between four coordinate systems:

World coordinates used to define uniform coordinate system for all abstract workstation

View reference coordinate used to define a view

Normalized projection coordinates; used to facilitate assemblies of different views

Device coordinates: one coordinate system per workstation representing its display space

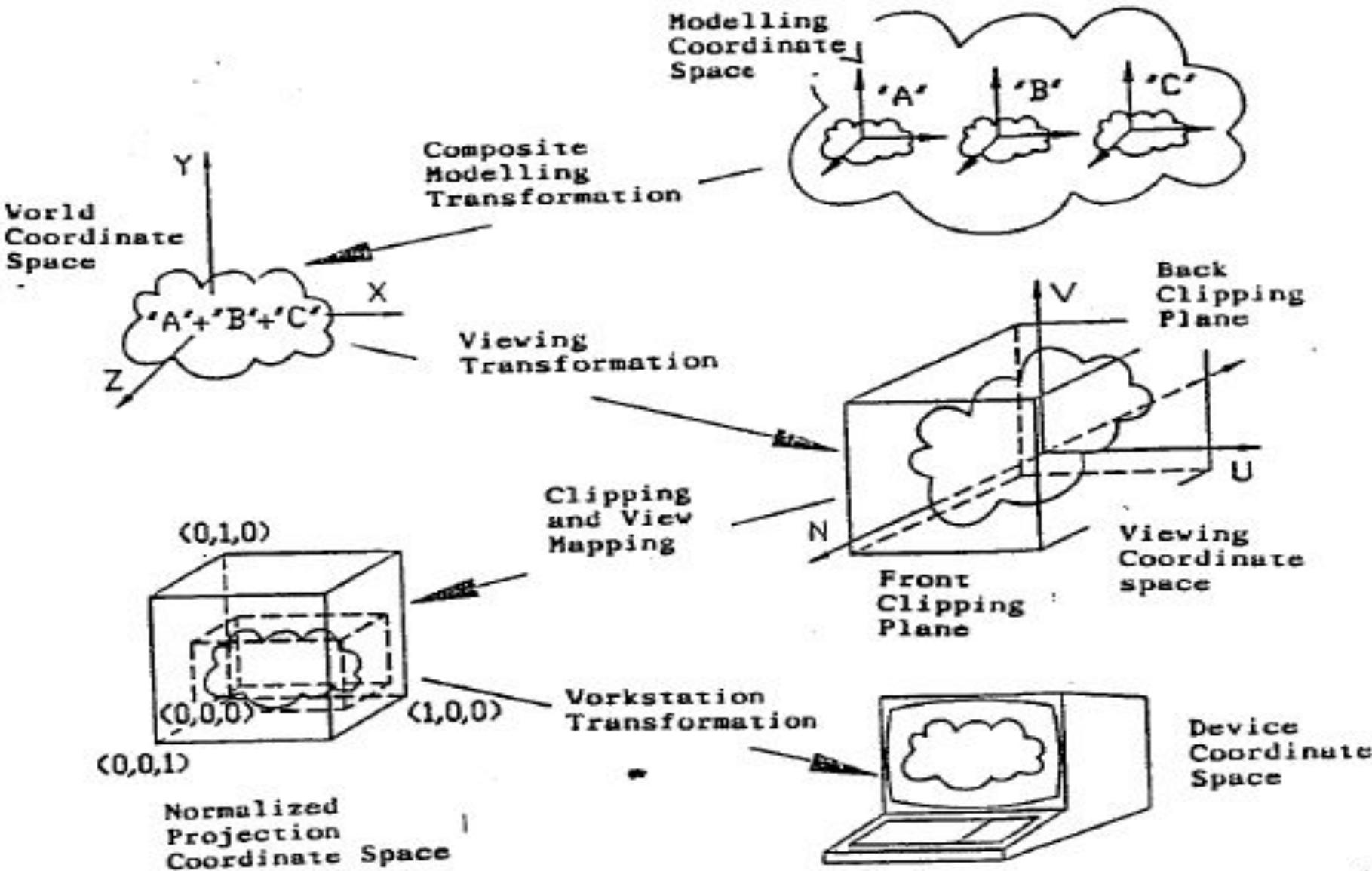


Fig. 7 Coordinate systems and transformations in PHIGS

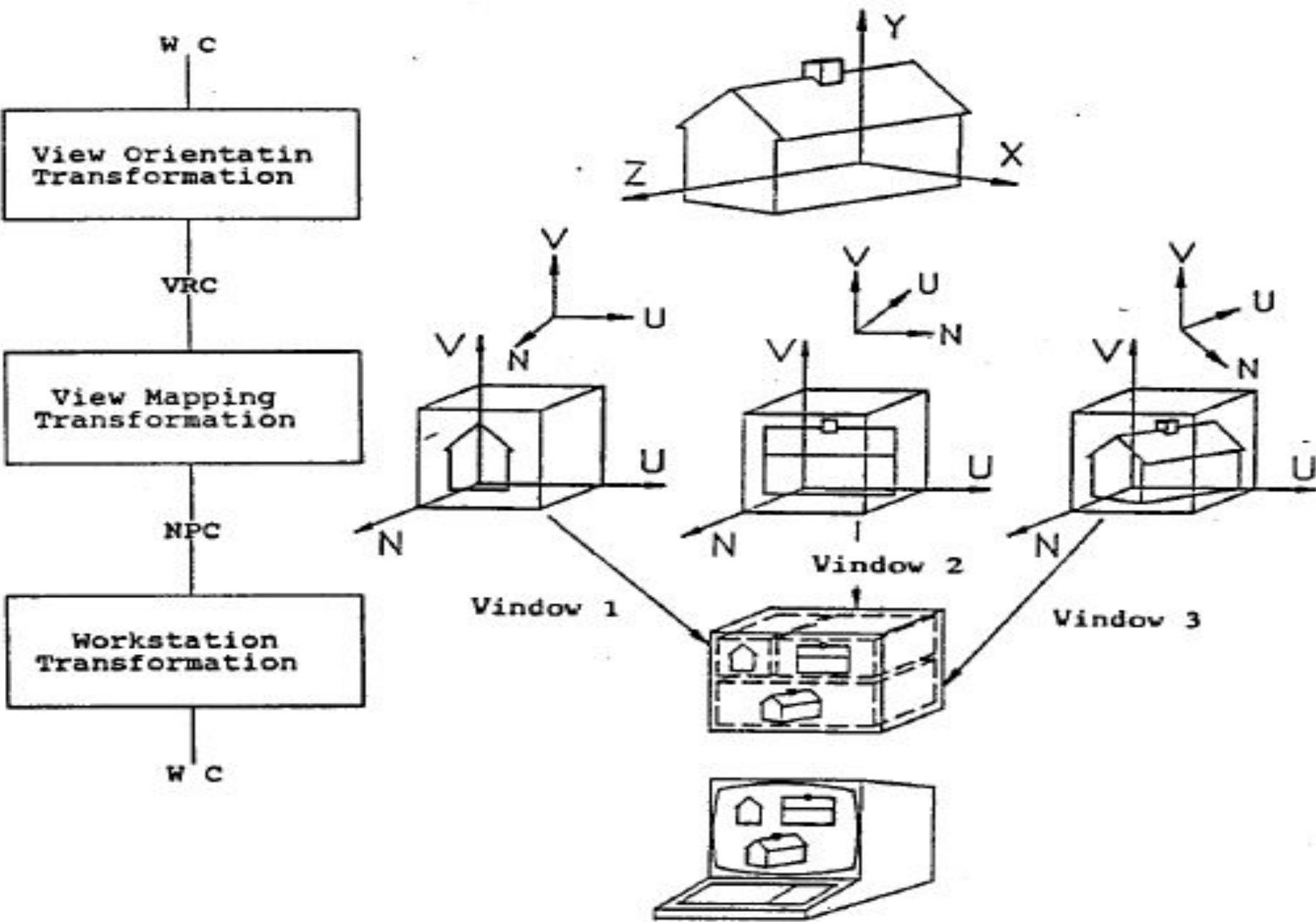


Fig.8 Viewing transformations in PHIGS

Graphical Output

5. Graphical input

An application program gets graphical input from an operator by controlling the activity of one or more logical input devices.

6. Language Interfaces

PHIGS defines only a language independent nucleus of a graphics system

For integration into a language, PHIGS is embedded in a language dependent layer containing the language conventions, e.g. parameter and name assignment

In case of the layer model, each layer may call the functions of the adjoining lower layers. In general the application program uses the application oriented layer, the language dependent layer, other application dependent layers and operations system resources. There are standards of the language dependent layers for the language FORTRAN, PASCAL and C

7. Graphics system PHIGS +

PHIGS+ enables to specify light sources

Parameters of the light sources are described and application program s may set the up

The predefined light sources are ambient, directional, positional, spot light source.

Shading method is an attribute of the graphics primitives.

NONE- No shading

COLOUR-color interpolation shading

NORMAL-normal interpolation shading

DOT PRODUCT- dot product interpolation shading

Light type accepted for shading

NONE – no reflectance calculation performed

AMBIENT-use ambient term

AMB_DIF-use ambient and diffuse terms

AMB_DIF_SPEC- use ambient, diffuse and specular terms

PHIGS+ offers additional output primitives and functions;

COMPUTE FILL AREA SET GEOMETRIC NORMAL-compute geometric normal of the fill area set

FILL AREA SET3 WITH DATA-creates a 3D fill area set structure element that includes color and shading data

QUADRILATERAL MESH3 WITH DATA-creates a 3D quadrilateral mesh primitive with color and shading data

TRIANGLE STRIP3 WITH DATA-creates a 3D triangle strip primitive with color and shading data

NON UNIFORM B-SPLINE CURVE NURBS-crates a structure element containing the definition of a non-uniform B-Spline Curve

Significance of PHIGS

Portability of program

PHIGS is computer and device independent graphics system. The application program that utilize PHIGS can be easily transported between host processors and graphics devices

Sophisticated capabilities save development time

PHIGS manages the storage and display of 2D and 3D graphical data, creates and maintains a hierarchical database

Increased program performance because of fewer error conditions

Application using PHIGS have well defined inputs and outputs that minimizes errors

GKS (the Graphical Kernel System)

- an ANSI and ISO standard.
- **standardizes 2D graphics functionality** at a low level.

The main objective of the Graphical Kernel System, GKS, is the **production and manipulation of pictures** in a computer or graphical **device independent way**.

The primary purposes of the standard are:

- To **provide for portability** of graphics application programs.
- To **aid in the understanding of graphics methods** by application programmers.
- To **provide guidelines for manufacturers** in describing useful graphics capabilities.

Three Basic parts:

1. An **informal exposition** of the contents of the standard includes text positioning, polygonal filling etc
2. A **formalization of the expository material** by way of abstracting the ideas into discrete **functional descriptions** that **contains information as descriptions of input and output parameters, the effect each function, references into the expository material, and a error description** in a language independent way.
3. **Language bindings.** These bindings are an implementation of these abstract functions in a specific computer language such as Fortran or Ada or C.

Five Main GKS Primitives :

In GKS, pictures are considered to be constructed from a number of **basic building blocks**. These basic building blocks are of a number of types each of which can be used to describe a different component of a picture.

The five main primitives in GKS are:

Polyline: Draws a sequence of connected line segments.

Polymarker: Marks a sequence of points with same symbol.

Fill area: Displays a specified area with filled intensity.

Text: Draws a string of characters.

Cell array: Displays an image composed of a variety of colours or grey scales.

Associated with each primitive is a **set of parameters which is used to define particular instances** of that primitive.

Example

The parameters of the **Text** primitive are the **string or characters to be drawn** and the **starting position of that string**.

`TEXT(X, Y, 'ABC')`

will draw the characters ABC at the position (X, Y).

GKS needs to know the **height of a character string** and the **angle** at which it is to be drawn.

These additional data are known as **attributes**.

GKS Output Primitives

GKS standardizes a reasonably complete set of functions for displaying 2D images.

It contains functions for drawing lines, markers, filled areas, text, and a function for representing raster like images in a device-independent manner.

In addition to these basic functions, GKS contains many **functions for changing the appearance** of the output primitives, such as changing **colors**, changing line **thicknesses**, changing **marker types** and **sizes** etc.

The functions for changing the appearance of the fundamental drawing functions (output primitives) are called **attribute setting functions**.

Polyline

It is a GKS function for **drawing line segments**

The polyline function **takes an array of X-Y coordinates** and draws line segments connecting them.

Attributes that control the appearance of a polyline:

Linetype, which controls whether the polyline is drawn as a **solid, dashed, dotted, or dash-dotted line**.

Linewidth scale factor, which controls **how thick** the line is.

Polyline color index, which controls **what color** the line is.

POLYLINE(N, XPTS, YPTS) where XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)).

The Polyline generated consists of N - 1 line segments joining adjacent points starting with the first point and ending with the last.

Polymarker

The Polymarker function **allows drawing marker symbols** centered at coordinate points specified.

Attributes that control the appearance of polymarkers:

Marker: specifies **one of five standardized symmetric characters** to be used for the marker.

The five characters are **dot, plus, asterisk, circle, and cross**.

Marker size: scale factor that controls **how large** each marker is (except for the dot marker).

Polymarker color index: Specifies the **marker color**
GKS provides the primitive polymarker marks a set of points, instead of drawing lines through a set of points.

Polymarker

A Polymarker is generated by the function:

POLYMARKER(N, XPTS, YPTS)

where the arguments are the **same as for the Polyline function**, namely XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)).

Polymarker **places a centered marker** at each point.

GKS recognizes the common use of markers to identify a set of points in addition to marking single points and so the marker function is a polymarker.

POLYMARKER(I9, XDK, YDK)

Text

The text function **allows drawing a text string** at a specified coordinate position.

Attributes that control the appearance of text:

Text font and precision: Specifies **text font** for the characters and **precision** of their representation to adhere to the settings of the other text attributes.

Character expansion factor: Controls the height-to-width ratio of each plotted character.

Character spacing,:: Specifies the additional white space needed to be inserted between characters in a string.

Text color index, which specifies **what color** the text string should be.

Text

Character height: Specifies the height of the characters

Character up vector: Specifies the **angle** of the text drawn

Text path: Specifies in the direction the text should be written (right, left, up, or down).

Text alignment: Specifies **vertical and horizontal centering** options for the text string.

A **text string** may be generated by invoking the function:

TEXT(X, Y, STRING) where (X, Y) is the text position and STRING is a string of characters.

Text

The **character height attribute** determines the height of the characters in the string. Since a character in a font will have a designed aspect ratio, the character height also determines the character width. The character height is set by the function:

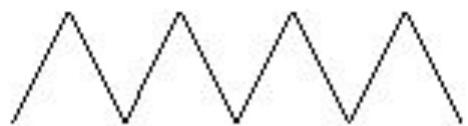
SET CHARACTER HEIGHT(H)

where H is the character height.

The **character up vector** is perhaps the most important text attribute. Its main purpose is to determine the orientation of the characters. However, it also sets a reference direction which is used in the determination of text path and text alignment. The character up vector specifies the up direction of the individual characters. It also specifies the orientation of the character string in that. By default, the characters are placed along the line perpendicular to the character up vector. The function:

SET CHARACTER UP VECTOR(X, Y)

GKS



Polyline



Fill area



Polymarker



Cell array

Example text string

Text