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 as 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,

1. What is the fraction of total refresh time spent in retrace of electron beam for a non interlaced raster system with a resolution of 1280 x 1024 and refresh rate of 60 Hz , the horizontal retrace time is 5 microsecond and vertical retrace time of 500 microsecond?

Here ,

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

where t_h is horizontal retrace

t_v is vertical retrace and r is the refresh rate

n x m = 'm' scan lines and 'n' pixels in each scan line

2. 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?

Aspect ratio = w/h i.e. 1280/12 / 1024/9.6 = 1

Diameter of each pixel = 12/1280 or 9.6/1024 = 0.0094 inches

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

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

Display Devices

Terminologies

Pixel

The **smallest number of phosphor dots that the electron gun can focus** on is called a **pixel**, it comes from the term **picture element**. Each pixel has a **unique address**, which the computer uses to locate the pixel and control its appearance. Some electron guns can focus on pixels as small as a single phosphor dot.

Fluorescence / Phosphorescence

When the electron beam strikes the phosphor-coated screen of the CRT the individual electrons are moving with the kinetic energy proportional to the acceleration voltage.

Some of this energy is dissipated as heat but the rest is transferred to the electron of the phosphor atoms making them jump to *higher quantum energy levels*

In returning to their previous quantum levels these excited electrons give up their extra energy in the form of light at frequencies that is colors predicted by the quantum theory

Any given phosphor has several different quantum levels to which electrons can be excited each corresponding to a color associated with return to an unexcited state

Further, electrons on some levels are less stable and turn to the unexcited state more rapidly than others.

A phosphors fluorescence is the light emitted as these 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

Since fluorescence usually last just a fraction of a microsecond the most of the light emitted is phosphorescence for a give phosphor

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

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 sops flickering and fuses into a steady image is called the critical fusion frequency (CFF)

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

Horizontal scan rate:

The horizontal scan rate is the number of scan lines per second The rate is approximately the product of the refresh rate and the number of scan lines

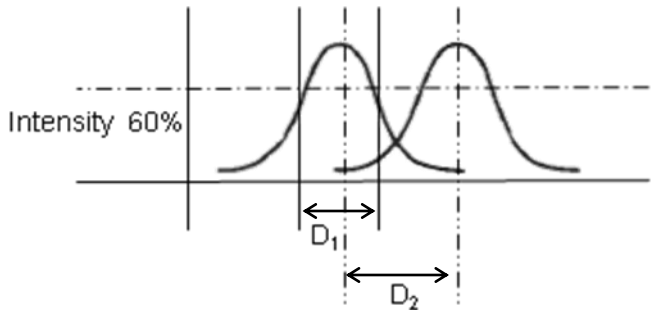
Resolution

Resolution is defined as the maximum number of points that can be displayed horizontally and vertically with out overlap on a display device

A monitor's resolution is determined by the **number of pixels on the screen**, expressed as a matrix. The more pixels a monitor can display, the higher its resolution and the dearer its images appear. For example, a resolution of 640 X 480 means that there are 640 pixels horizontally across the screen and 480 pixels vertically down the screen. The **actual resolution** is determined by the video controller not by the monitor itself—most monitors can operate at several different resolutions e.g. 800 x 600, 1024 x 768, 1152 x 864, 1280 x 1024. As the **resolution increases, the image on the screen gets smaller**.

Factors affecting the resolution are as follows

i. Spot profile The spot intensity has a Gaussian distribution as depicted in figure. So two adjacent spots on the display device appear distinct as log as their separation D2 is greater than the diameter of the spot D1 at which each spot has an intensity of about 60 percent of that at the center of the spot



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 no necessarily a constant and it is not necessarily equal to the resolution of a pix-map, which is allocated in a buffer memory

Color CRTs

Color depends on the light emitted by phosphor.

Two type:

i. Beam Penetration Method

ii. Shadow Mask Method

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

i. A beam of slow electrons excites only the outer red layer

ii. A beam of very fast electrons penetrates thru the red phosphor and excites the inner green layer

iii. When quantity of red is more than green then color appears as orange

iv. 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

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

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

The electron beam form 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

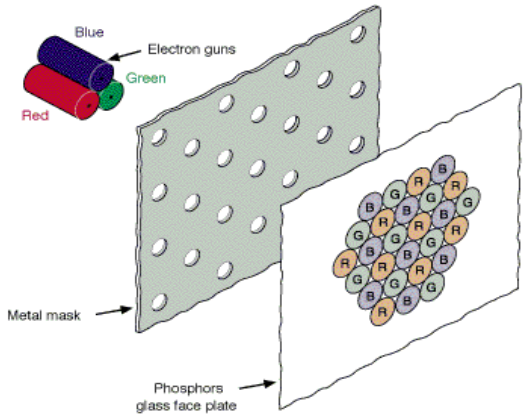
Thus, a wide range of colors can be produced by each triad depending on how strongly each individual phosphor dot in a triad is excited.

Two types:

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. A 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

The necessity of triad has reduced the resolution of a color CRT

The distance between the center of adjacent triads is called a *pitch*

In very high resolution tubes, pitch measures 0.21 mm (0.61 mm for home TV tubes)

The diameter of each electron beam is set at 1.75 times the pitch

For example if a color CRT is 15.5 inches wide and 11.6 inches high and has a pitch of 0.01 inches

The beam diameter is therefore 0.01 x 1.75 = 0.018 inches

Thus the resolution per inch is about 1/0.018 = 55 lines

Hence the resolution achievable for the given CRT is 15.5 x 55 = 850 by 11.6 x 55 = 638

The resolution of a CRT can there fore be increased by decreasing the pitch

But small pitch CRT is difficult to manufacture because it is difficult to set small triads and the shadow mask is more fragile owing to too many holes on it .

Besides the shadow is more likely to warp from heating by the electrons

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

- Cathode Ray Tube (CRT)
- Field emission display (FED)
- Surface-conduction Electron-emitter Display (SED)
- Vacuum Fluorescent Display (VFD)
- Electroluminescent Displays (ELD)
- Light- Emitting Diode Displays (LED)
- Plasma Display Panel (PDP)
- Electrochemical Display (ECD)

Non-Emissive Displays

Light is produced behind the screen and the image is formed by filtering this light

The Non emissive are optical effects to convert the sunlight or light from any other source to graphic form. Liquid crystal display is an example.

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.

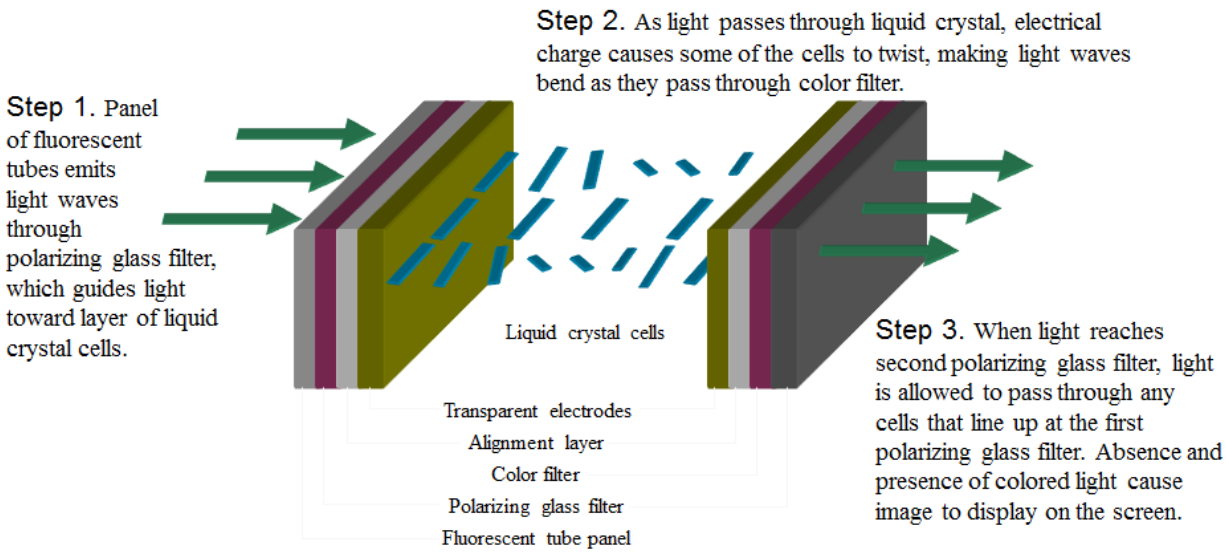
It consumes less electrical energy than CRT and LCD monitors as a cold cathode fluorescent lamp is embedded in the panel.

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

LCD (Liquid Crystal Display)

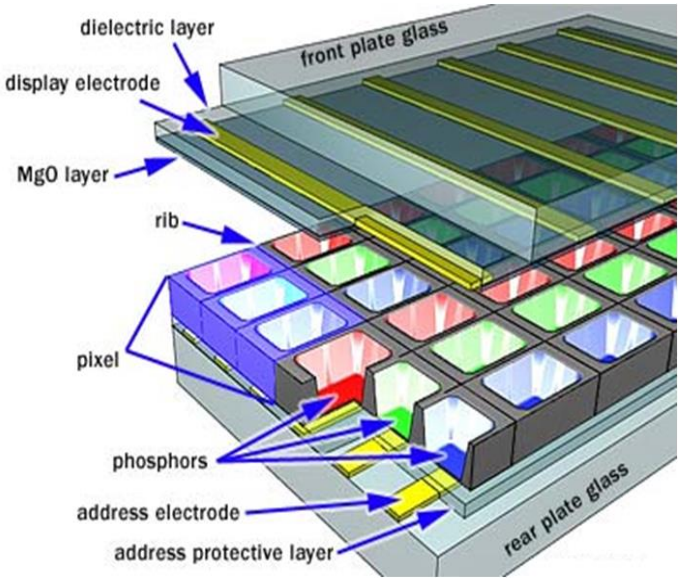
- A type of flat-panel display
- Uses liquid crystals between two sheets of material to present information on a screen
- An electric current passes through the liquid crystals, they twist
- Depending on how much they twist, some light waves are passed through while other light waves are blocked. This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

- 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
- 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
- An importance 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
- Brightness of an LCD monitor is measured in nits
- Nit is a unit of visible light intensity equal to one candela meter
- Resolution and dot pitch determines quality of LCD monitor



Plasma Panel

- 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



Hard Copy Devices

- 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:

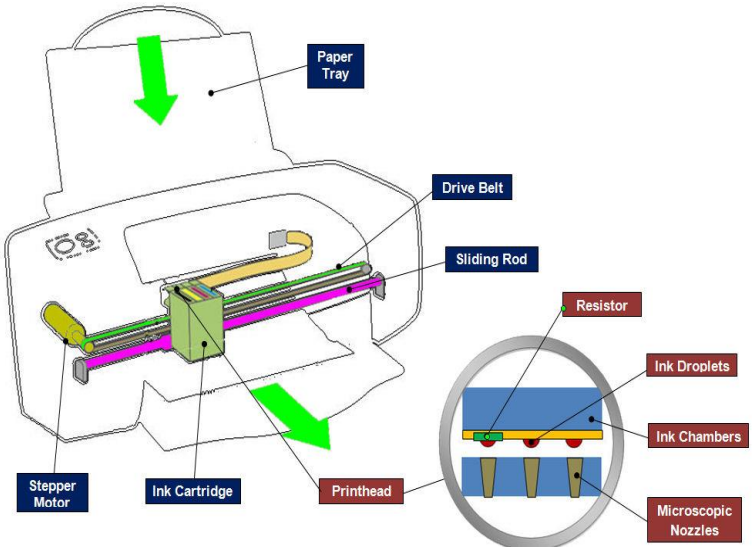
- Character printers prints one character of a text at a time
- Line printer prints one line of the text at a time
- A page printer prints one page of the text a t a time
- According to the technology used printers produce output by either impact or non impact methods

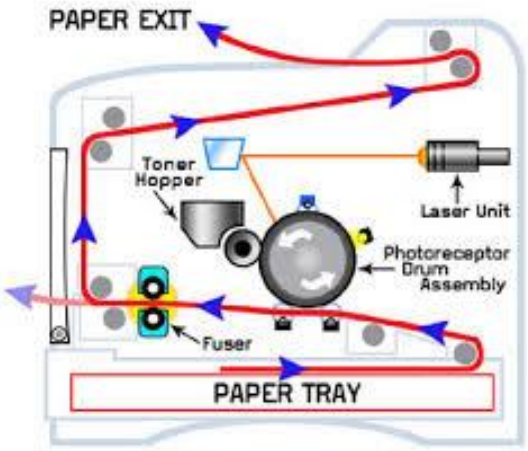
Impact printers Impact printers press the formed character faces against an inked ribbon onto paper

Character impact printers often have a dot matrix print head containing a rectangular array of protruding wire pins with a number of pins depending on the quality of the printer

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.

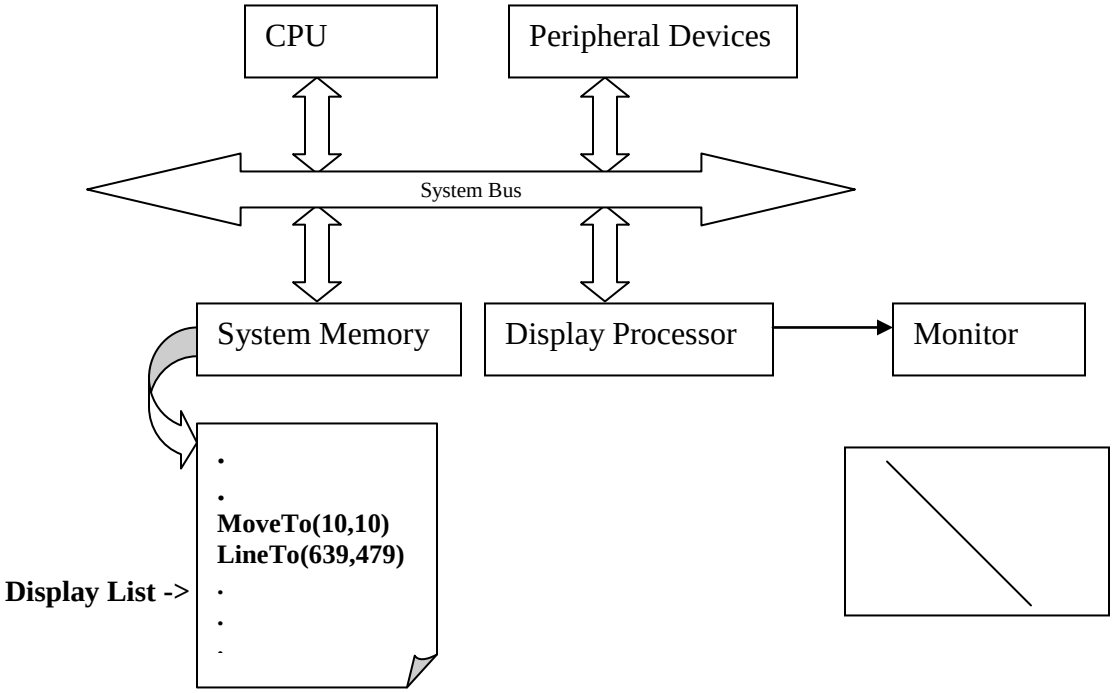




Display Technology

i. Vector Display Technology

Vector display technology was developed in 60's and used as a common display device until 80's , It is also called *random scan, a stroke, a line drawing* or *calligraphic display*



It consists of a central processing unit, a display processor , a monitor , system memory and peripheral devices such as mouse and key board

A display processor is also called a *display processing unit* or *graphics controller*

The application program and *graphics subroutine package* both reside in the system memory and execute on CPU . A graphics subroutine package creates a *display list* and stores in the system memory

A display list contains point and line plotting commands with end point coordinates as well as character plotting commands

The DPU interprets the commands in the display list and plots the respective output primitives such as point, line and characters

As a matter of fact the DPU sends digital point coordinates to a vector generator that converts the digital coordinate values to analog voltages for circuits that displace an electron beam hitting on the CRT's phosphor coating

Therefore the beam is deflected from endpoint to endpoint as dictated by the arbitrary order of the commands in the display list, hence the name *Random Scan Display* Since the light output of the phosphor decays in tens or at most hundreds of microseconds the DPU must cycle thru the display list to refresh the image around 50 times per second to avoid flicker. A portion of the system memory where display list resides is called a *refresh buffer.*

This display technology is used with mono chromatic CRTs or beam penetration color CRTs

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

Disadvantages:

i. A vector display can not fill areas with patterns and manipulate bits

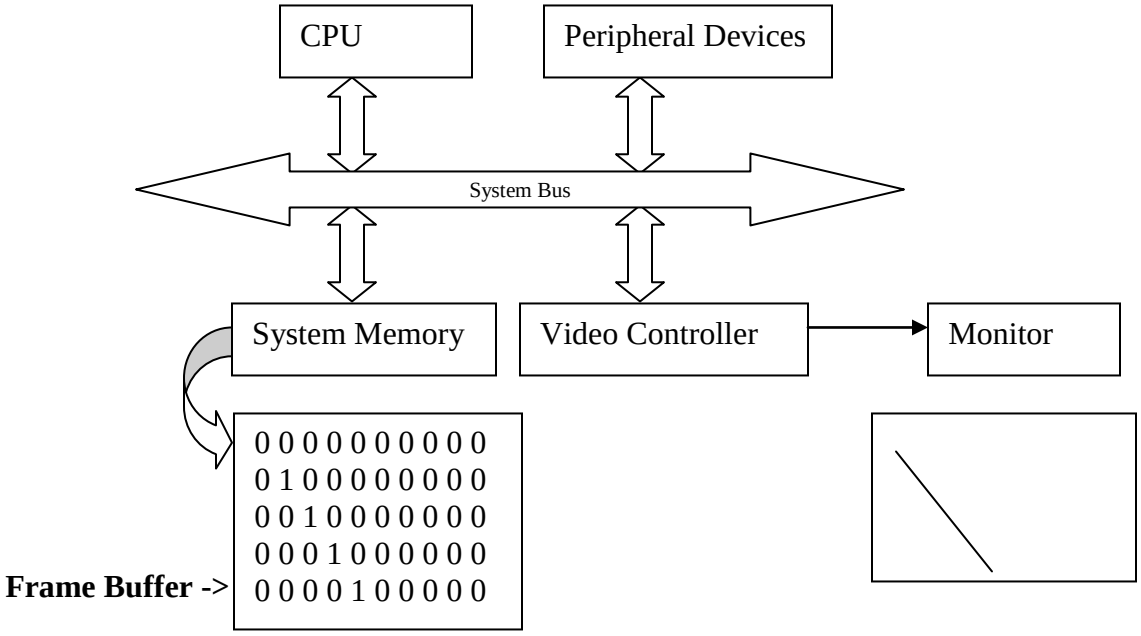
ii. Time required for refreshing an image depends upon its complexity (more the lines , longer the time) the flicker may therefore appear as the complexity of the image increases. The fastest vector display can draw about 100000 short vectors in a refresh cycle without flickering

ii. Raster Display Technology

This technology based on television technology was developed in early 70s

It consists of central processing unit, a video controller, a monitor, system memory and peripheral devices such as mouse and keyboard.

The application program and graphics subroutine package both reside in the system memory and execute on CPU.



When a particular command such as a `line(x1,y1,x2,y2)` is called by the application program the graphics subroutine package sets the appropriate pixels in the *frame buffer*, a portion of the system memory

The *video controller* then cycles thru the frame buffer, one scan line at a time typically 50 times per second.

It brings a value of each pixel contained in the buffer and uses it to control the intensity of the CRT electron beam.

So there exists a one to one relationship between the pixel in the frame buffer and that on the CRT screen

A 640 pixels by 480 lines is an example of *medium resolution* raster display

A 1600 by 1200 is a *high resolution* one

A pixel in a frame buffer may be represented by one bit as in monochromatic system where each pixel on CRT screen is either on '1' or off '0'

Or it may be represented by eight bits resulting 2^8 = 256 gray levels for continuous shades of gray on CRT screen

In color system each of the three color red, green and blue is represented by eight bits producing 2^{24} = 16 million colors

A medium resolution color display having 640 x 480 pixels will thus require (640 x 480 x 24) / 8 = 9kb of RAM

Advantages

i. It has an ability to fill the areas with solid colors or patterns

ii. The time required for refreshing is independent of the complexity of the image

iii. Low cost

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

Video Controller/Video Cards

The quality of the images that a monitor can display is defined by the video card (also called the video controller or the video adapter) and the monitor. The video controller is an intermediary device between the CPU and the monitor. It contains the **video-dedicated memory** and other circuitry necessary to send information to the monitor for display on the screen.

In most computers, the video card is a separate device that is plugged into the motherboard. In many newer computers, the video circuitry is built directly into the motherboard, eliminating the need for a separate card.

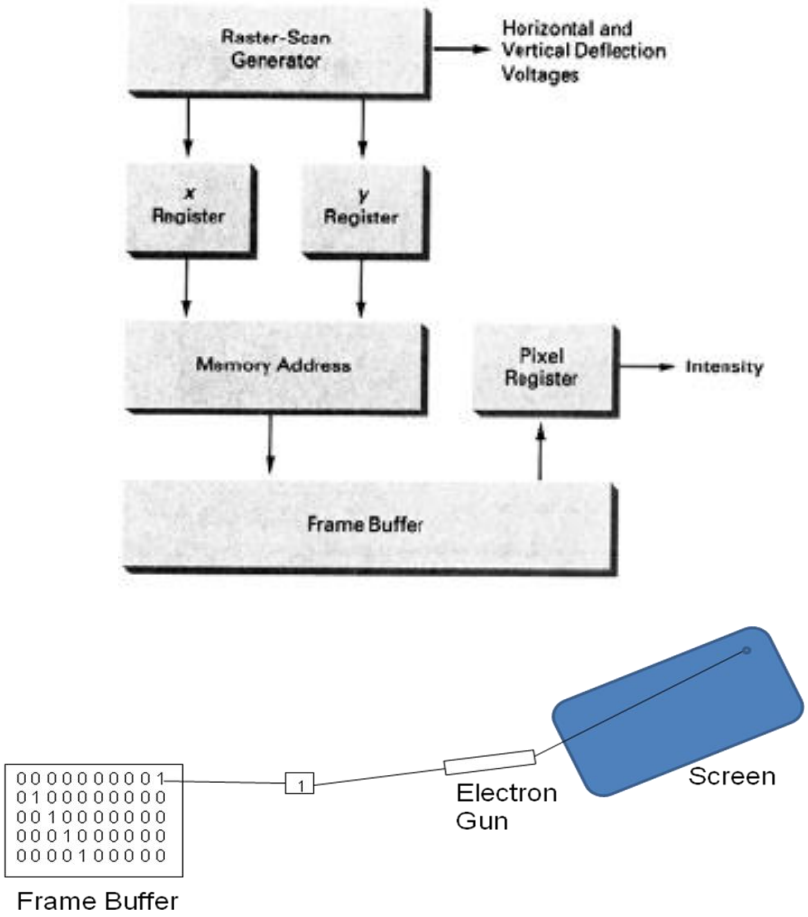
The screen changes constantly as a user works—the screen is updated many times each second, whether anything on the screen actually changes or not.

If the user wants more colors or a higher resolution, the amount of data can be much higher For example, for "**high color**" (24 bits, or 3 bytes per pixel will render 16 million colors) at a resolution of 1024 x 768, the computer must send 2,359,296 bytes to the monitor for each screen.

Today's video controllers feature their own built-in microprocessors , which frees the CPU from the burden of making the millions of calculations required for displaying graphics. The speed of the video controller's chip determines the speed at which the monitor can be refreshed.

Video controllers also feature their own built-in video RAM, or VRAM (which is separate from the RAM that is connected to the CPU). VRAM is dual-ported, meaning that it can send a screen full of data to the monitor and at the same time receive the next screen full of data from the CPU.

Basic Raster Scan Video Controller Operation



Frame Buffer

Frame buffer is the portion of memory used to store intensity information of physical pixels to be displayed on the screen. There is a one to one mapping of the physical pixel on the screen and its logical representation in the memory (frame buffer) in the form of pixel intensity values. In case of a raster system, there is at least one bit assigned for each physical pixel on the screen but if more bits are assigned to the pixel then different intensities can be generated out of the single pixel.

Color Manipulation Technique

In general, the size of the frame buffer depends upon the total number of bits assigned per pixel and the total resolution of the screen

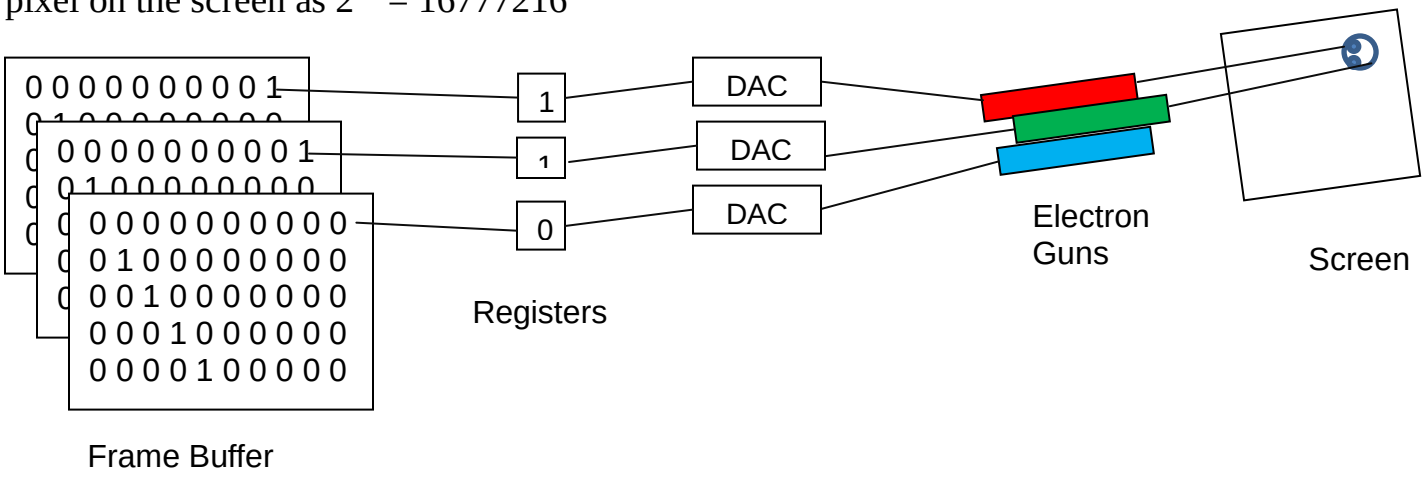
```
Frame Buffer size  = total resolution * bits assigned per pixel
```

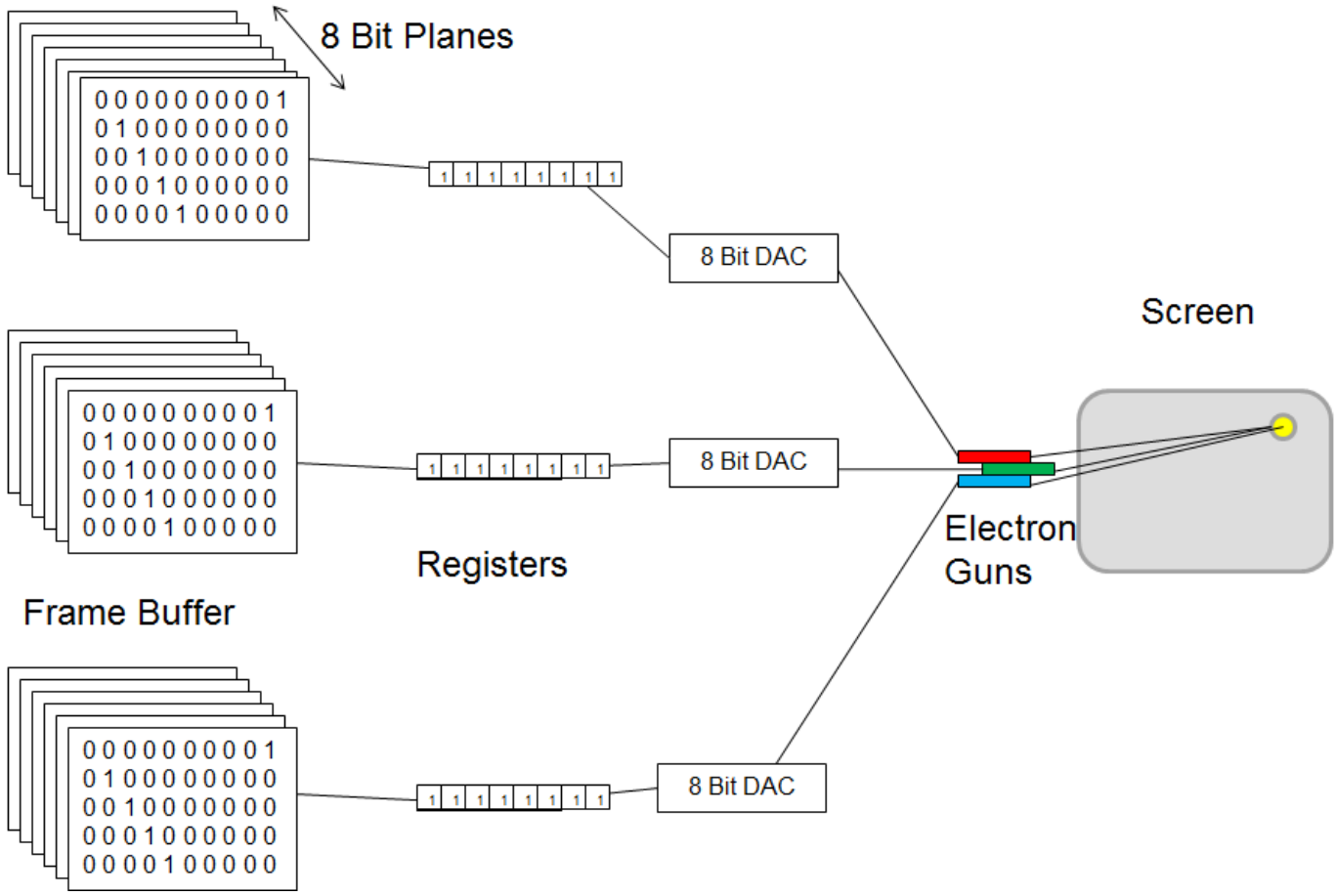
So if the total resolution of the screen is 640 * 480 and 8 bits are assigned per pixel then the total size of the frame buffer will be 640 * 480 * 8

The total number of intensities that can be produced out of a single pixel on the screen depends upon the total number of bits assigned for that pixel

```
Total number of intensities    =    2 number of bits assigned per pixel
that can be produced out of
a single pixel
```

So a 24 bit video card has the ability to produce 16 million different intensities out of a single pixel on the screen as 2^{24} = 16777216



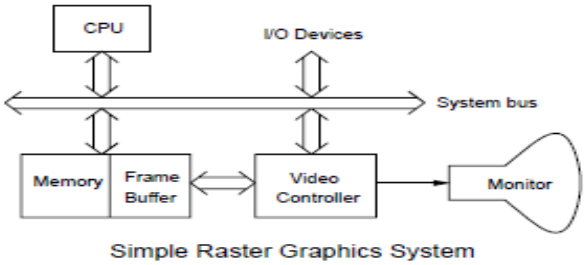

8 Bit Planes
0 0 0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
1 1 1 1 1 1 1 1
8 Bit DAC
Screen
8 Bit DAC
Registers
Frame Buffer
Electron
Guns
8 Bit DAC

Graphics Systems and Model

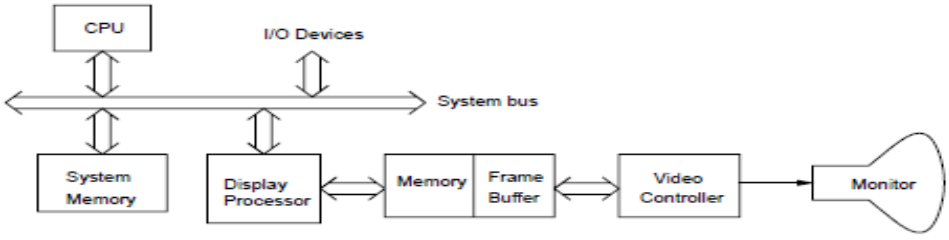
Raster Images: Raster image is an image of a 2-dimensional array of square (or generally rectangular) cells called *pixels* (short for "picture elements"). Such images are sometimes called *pixel maps*.
The simplest example is an image made up of black and white pixels, each represented by a single bit (0 for black and 1 for white). This is called a *bitmap*. For gray-scale (or *monochrome*) raster images raster images, each pixel is represented by assigning it a numerical value over some range (e.g., from 0 to 255, ranging from black to white). There are many possible ways of encoding color images.

Graphics Devices: The standard interactive graphics device is called a *raster display*. As with a television, the display consists of a two-dimensional array of pixels. There are two common types of raster displays.
Video displays: consist of a screen with a phosphor coating, that allows each pixel to be illuminated momentarily when struck by an electron beam. A pixel is either illuminated (white) or not (black). The level of intensity can be varied to achieve arbitrary gray values. Because the phosphor only holds its color briefly, the image is repeatedly rescanned, at a rate of at least 30 times per second.
Liquid crystal displays (LCD's): use an electronic field to alter polarization of crystalline molecules in each pixel. The light shining through the pixel is already polarized in some direction. By changing the polarization of the pixel, it is possible to vary the amount of light which shines through, thus controlling its intensity.

Irrespective of the display hardware, the computer program stores the image in a two-dimensional array in RAM of pixel values (called a *frame buffer*). The display hardware produces the image line-by-line (called *raster lines*). A hardware device called a *video controller* constantly reads the frame buffer and produces the image on the display. The frame buffer is not a device. It is simply a chunk of RAM memory that has been allocated for this purpose. A program modifies the display by writing into the frame buffer, and thus instantly altering the image that is displayed. An example of this type of configuration is shown below.



Simple Raster Graphics System



Raster Graphics with Display Processor

More sophisticated graphics systems, which are becoming increasingly common these days, achieve great speed by providing separate hardware support, in the form of a *display processor* (more commonly known as a *graphics accelerator* or *graphics card* to PC users). This relieves the computer's main processor from much of the mundane repetitive effort involved in maintaining the frame buffer. A typical display processor will provide assistance for a number of operations including the following:

Transformations: Rotations and scalings used for moving objects and the viewer's location.

Clipping: Removing elements that lie outside the viewing window.

Projection: Applying the appropriate perspective transformations.

Shading and Coloring: The color of a pixel may be altered by increasing its brightness. Simple shading involves smooth blending between some given values. Modern graphics cards support more complex procedural shading.

Texturing: Coloring objects by "painting" textures onto their surface. Textures may be generated by images or by procedures.

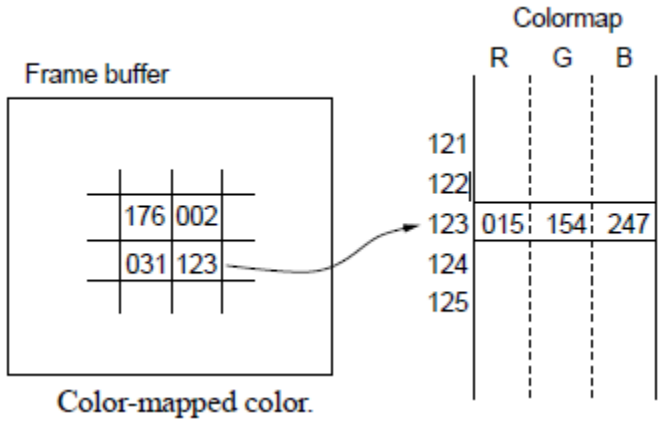
Hidden-surface elimination: Determines which of the various objects that project to the same pixel is closest to the viewer and hence is displayed.

Color: The method chosen for representing color depends on the characteristics of the graphics output device (e.g.,whether it is *additive* as are video displays or *subtractive* as are printers). It also depends on the number of bits per pixel that are provided, called the *pixel depth*. For example, the most method used currently in video and color LCD displays is a *24-bit RGB* representation. Each pixel is represented as a mixture of red, green and blue components, and each of these three colors is represented as a 8-bit quantity (0 for black and 255 for the brightest color).

In many graphics systems it is common to add a fourth component, sometimes called *alpha,* denoted A. This component is used to achieve various special effects, most commonly in describing how opaque a color is. In some instances 24-bits may be unacceptably large. For example, when downloading images from the web, 24-bits of information for each pixel may be more than what is needed. A common alternative is to used a *color map*, also called a *color look-up-table* (LUT). (This is the method used in most gif files, for example.) In a typical instance, each pixel is represented by an 8-bit quantity in the range from 0 to 255. This number is an index to a 256-element array, each of whose entries is a 234-bit RGB value. To represent the image, we store both the LUT and the image itself. The 256 different colors are usually chosen so as to produce the best possible reproduction of the image. For example, if the image is mostly blue and red, the LUT will contain many more blue and red shades than others.

A typical photorealistic image contains many more than 256 colors. This can be overcome by a fair amount of clever trickery to fool the eye into seeing many shades of colors where only a small number of distinct colors

exist. This process is called *digital halftoning*. Colors are approximated by putting combinations of similar colors in the same area. The human eye averages them out.



Color-mapped color.

Computer Graphic

Assignment 01

2-5 suppose an RGB raster system is to be designed using an 8-inch by 10-inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for the frame buffer?

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

2-6 how long would it take to load a 640 by 480 frame buffer with 12 bits per pixel, if 105bits 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 x 480 x 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 x 1024 x 24 bits = 31457280 bits

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

2-8 consider two raster systems with resolutions of 640 by 480 and 1280 by 1024. How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second? What is the access time per pixel in each system?

The access time per pixel is 1 / (640x480x60)sec

The access time per pixel is 1 / (1280x1024x60)sec

※accurate time (1/60 – 639 x T_{horiz} – T_{vert})/640x480 sec

2-12 what is the fraction of the total refresh time per frame spent in retrace of the

electron beam for a noninterlaced raster system with a resolution of 1280 by 1024, a refresh rate of 60 Hz, a horizontal retrace time of 5 microseconds, and a vertical retrace time of 500 microseconds?

1sec = 10^6 usec

Refresh rate = 60Hz = 1/60 sec to scan = 16.7 msec

The time for horizontal retrace = 1024 x 5 usec

The time for vertical retrace = 500 usec

Total time spent for retrace = 5120 + 500 = 5620 usec = 5.62 msec

The fraction of the total refresh time frame spent in retrace = 5.62 / 16.7 = 0.337

2-13 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 x 512

2 Assuming that a certain RGB raster system has 512*512 frame buffer with 12 bit per pixel and color lookup table with 24 bit for each entry

1 How many distinct color choice we have available

2 How many different color could we display at any one time?

3 How much storage spent altogether for the frame buffer and the color lookup table?

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 X 512 X 12 bit = 3145728 bit

The storage spent for the color lookup table is 2^{12} X 24 bit = 98304 bit

So the total storage spent altogether is 3145728 + 98304 = 3244032 bit

Hardware Concepts

Input devices

Tablet

Tablet a tablet is a **digitizer**. In general, a digitizer is a device which is used to 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

A tablet is a flat surface and it's size varies from 6 by 6 inches up to 48 by 72 inches or more

The accuracy of the tablets usually falls below 0.2 mm

There are three types of tablets

i. Electrical Tablet

A grid of wires on ¼ to ½ inch centers 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.

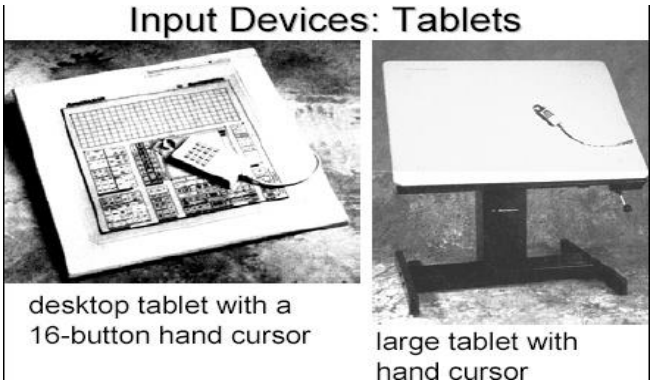
The signal strength is also used to determine roughly how far the stylus is from the tablet

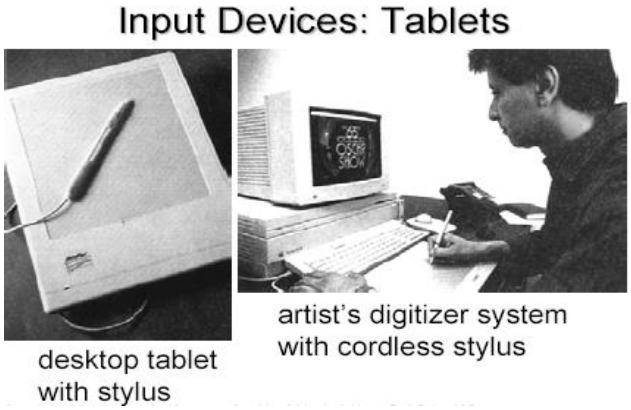
When the stylus is within ½ inch from the tablet it taken as near other wise it is either "far" or "touching"

When the stylus is near or touching, a cursor is usually shown on the display to provide visual feedback to the user

A signal is sent to the computer when the tip of the stylus is pressed against the tablet or when any button on the puck is pressed

The information provided by the tablet repeats 30 to 60 times per second





ii. Sonic Tablet

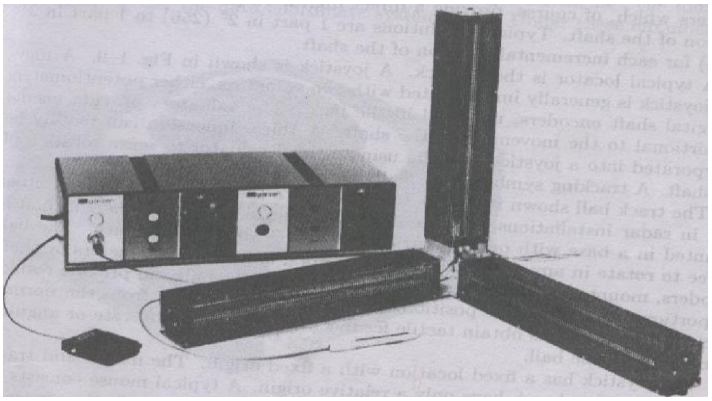
The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area

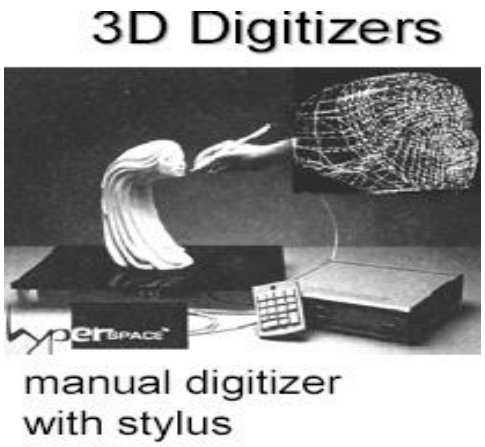
An electrical spark at the tip of the stylus creates sound bursts.

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

The main advantage of sonic tablet is that it doesn't require a dedicated working area as the microphones can be places 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





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

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

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.

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

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

iii. Electrical Touch Panel

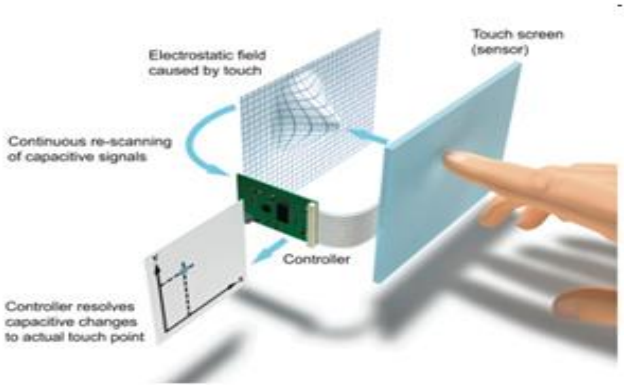
It consists of slightly separated two transparent panel one coated with a thin layer of conducting material and the other with resistive material

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

Input Devices: Touch Screens

plasma panels with touch screens



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

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 the screen images as it is pointer 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

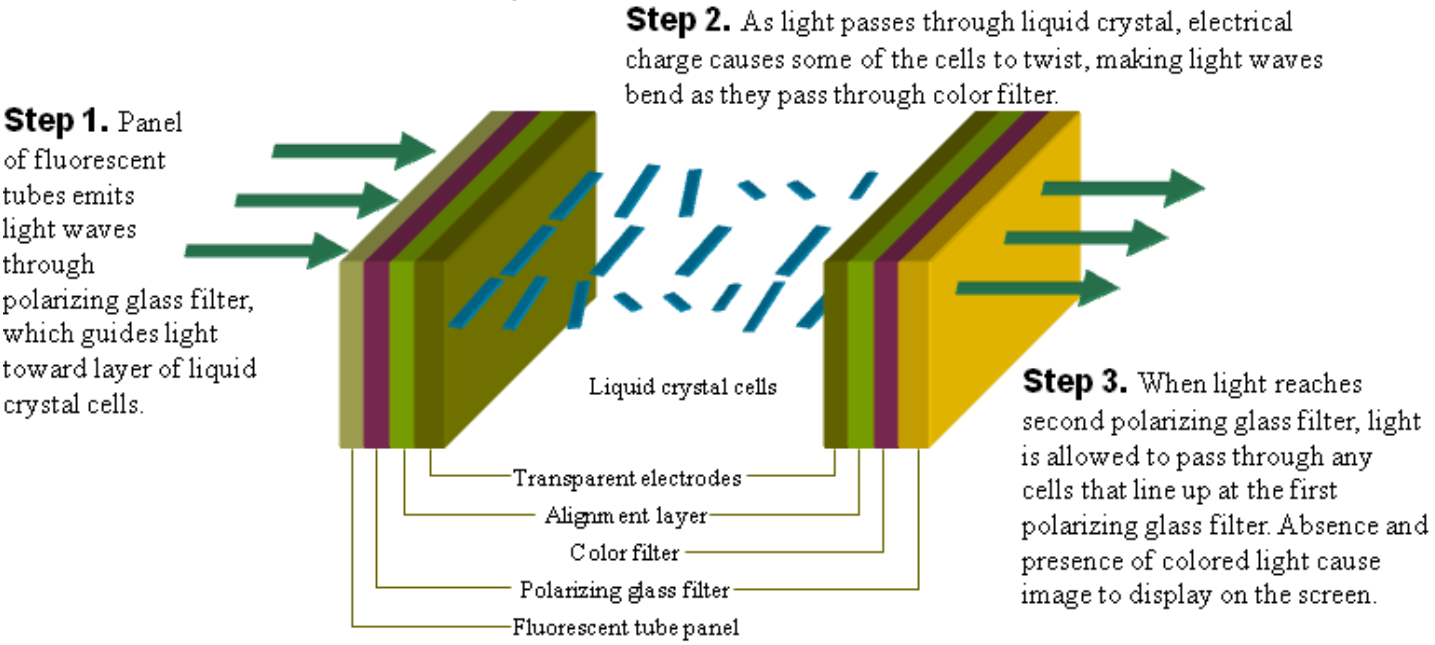
iv. It gives sometimes false reading due to back ground lighting in a room

Input Devices: Light Pen

LCD (Liquid Crystal Display)

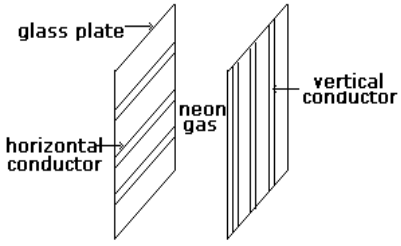
- A type of flat-panel display
- Uses liquid crystals between two sheets of material to present information on a screen

- 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
- 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



Plasma Panels (gas-discharge display)

- Region between two glass plates is filled with a mixture of gases such as neon, xenon.
- A series of vertical conducting ribbons is placed on one glass panel and a set of horizontal ribbons is built into other gas panel.
- Firing voltages applied to a pair of horizontal and vertical conductors cause gas at intersection of the two conductors to break down into a glowing plasma of electrons and ions
- By controlling the amount of voltage applied at various points on grid, each point acts as a pixel(intersection of conductors) to display an image.
- Picture definition is stored in a refresh buffer and firing voltages are applied to refresh pixel positions 60 times per second.



Voice System

- Consists of speech recognizer which analyze the sound of each person
- It must consist of dictionary of words (frequency pattern) spoken words are converted into frequency pattern.

Three Dimensional Viewing Devices

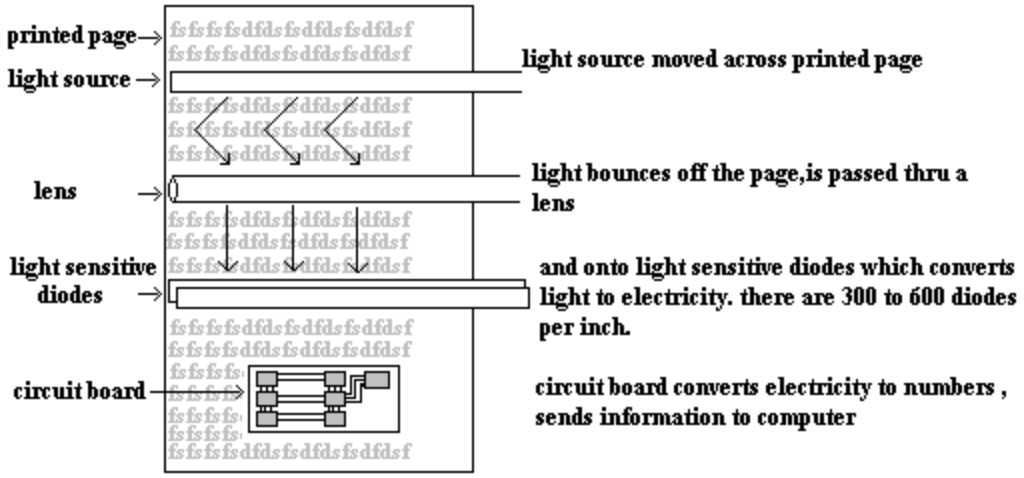
- A technique that reflects a CRT image from a vibrating, flexible mirror.
- As the varifocal mirror vibrates, it changes focal length.
- These vibrations are synchronized with the display of an object on a CRT so that each point on the object is reflected from the mirror into a spatial position corresponding to the distance of that point from a specified viewing position.
- This allows us to walk around an object or scene and view it from different sides.

Stereoscopic views

- Another method for representing three dimensional objects by overlapping images by 60%.
- Does not produce true 3D image but provides 3D effect by presenting a different view to each eye of an observer so that scenes appears to have depth.
- Two views of a scene generated from a viewing direction corresponding to each eye.
- A component in virtual reality (a computer generated simulation of real or imagined physical space, ultimate multimedia experience) systems.

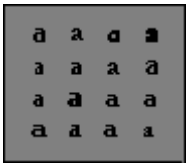
Scanner

- Converts any printed image of an object into electronic form by shinning light onto the image and sensing the intensity of light's reflection at any point.
- Color scanners use filters to separate components of color into primary additive colors (red, green, blue) at each point.
- R G B are primary additive colors because they can be combined to create any other color.
- Image scanners translate printed images into electronic format that can be stored into a computer memory.
- Software is then used to manipulate the scanned electronic image.
- Images are enhanced or manipulated by graphics programs like Adobe.



Optical Character Recognition (OCR)

- For text document we use Optical Character Recognition software to translate image into text that is editable.
- When a scanner first creates an image from a page the image is stored in computer's memory as bitmap.
- A bitmap is a grid of dots, each dot represented by one or more bits.
- OCR software translates the array of dots into text that the computer can interpret as number and letters by looking at each character and trying to match the character with its own assumption about how the image should look like.



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 a t 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
Character impact printers often have a dot matrix print head containing a rectangular array of protruding wire pins with a number of pins depending on the quality of the printer

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

The print head contains an ink cartridge which is made up of a number of ink filled firing chambers each attached to a nozzle thinker that n a human hair

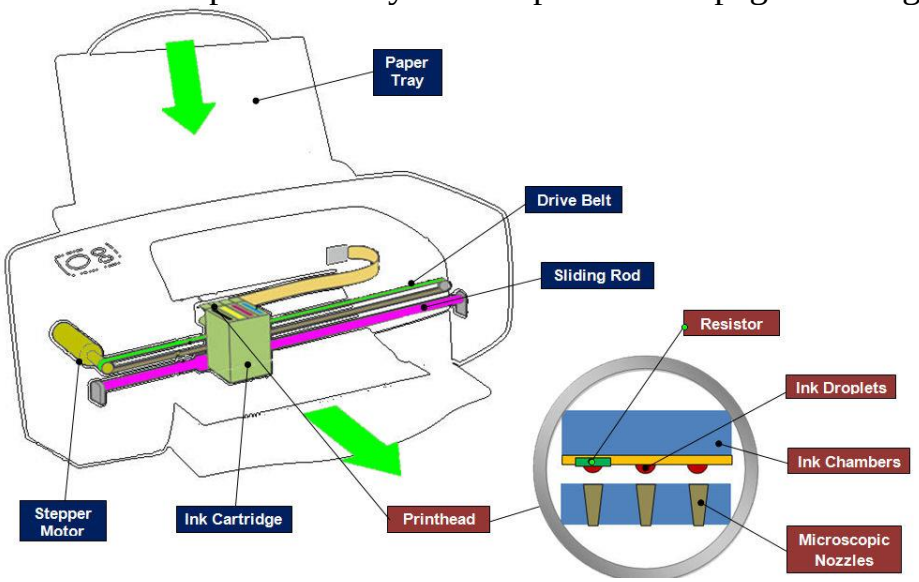
When an electric current is passed thru a resistor the resistor heats a thin layer of ink at the bottom of the chamber

Causing ink to boil and form a vapor bubble that expands and pushes ink thru the nozzle to form a droplet at the tip of the nozzle

The pressure of vapor bubble forces the droplet to move to the paper

A color ink jet printer employs four ink cartridges: one each for cyan, magenta, yellow and black

The ink of desired color can be placed at any desired point of the page in a single pass



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 charge photo conductive material

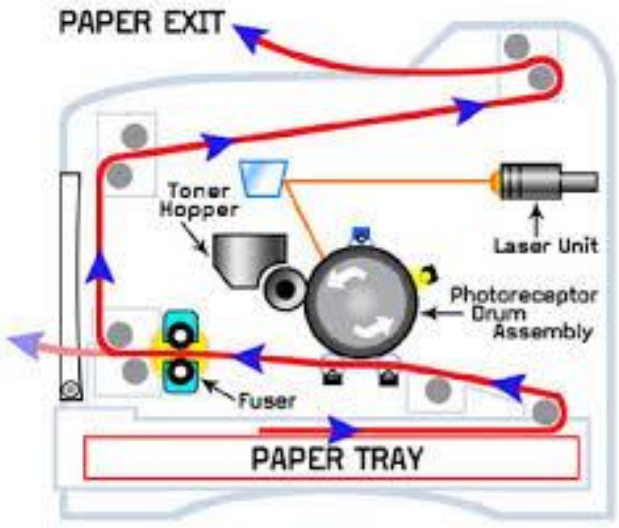
In a laser device a laser beam creates a charge distribution on a rotating drum coated with a photo electric material such as selenium. Toner is applied to drum and then transferred to paper.

Just as the electron gun in a monitor can target any pixel, the laser in a laser printer can aim at any point on a drum, creating an electrical charge.

Toner, composed of tiny particles of oppositely charged ink, sticks to the drum in the places the laser has charged

With pressure and heat, the toner is transferred off the drum onto the paper.

The paper then moves to a fusing station where toner is permanently fused on the paper with page



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)

Drum plotters

A drum plotter contains as long cylinder and a pen carriage

Paper is placed over the drum and the drum rotates back and forth to give up and down movement

The pen is mounted horizontally on the carriage that moves horizontally along with the carriage left to right or right to left on the paper to produce drawings

The pen and drum both mover under the computer control to produce the desired drawing

Several pens with different color dinks can be mounted on the carriage for multicolor drawing

Inkjet plotters

Many plotters us ink jets in place of ink pens

The paper is placed on a drum and the ink jets with different colored ink are mounted on a carriage

Such plotters are capable of producing multicolor large drawings

LCD (Liquid Crystal Display)

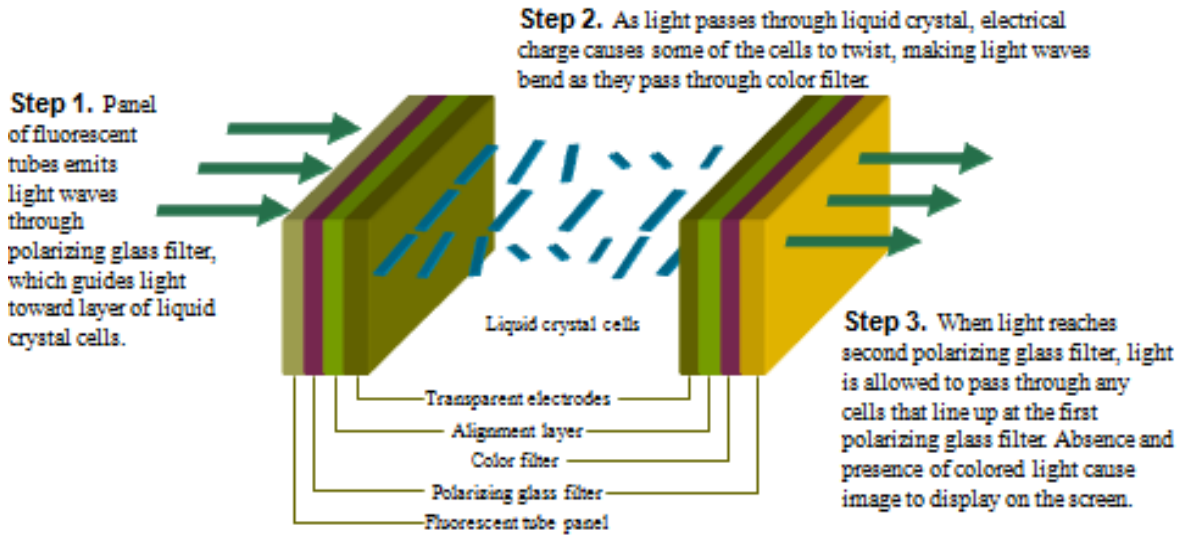
It is a type of flat-panel display

It uses liquid crystals between two sheets of material to present information on a screen

The LCD monitor creates images with a special kind of liquid crystal that is normally transparent but becomes opaque when charged with electricity.

As 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



LCD monitors produce color using either passive-matrix or active-matrix technology

Active Matrix LCD

The active matrix LCD technology , also known as a TFT (thin-film transistor) display, **assigns a transistor to each pixel (or** to each liquid crystal cell**)**, and each pixel is turned on and off individually.

This enhancement allows the pixels to be refreshed much more rapidly and display is of high-quality color that is viewable from all angles (wider viewing angle)

Active matrix displays use **thin-film transistor** (TFT) technology, which employs as many as four transistors per pixel.

Passive Matrix Display

The passive matrix LCD relies on transistors for each row and each column of pixels, thus creating a grid that defines the location of each pixel. The color displayed by a pixel is determined by the electricity coming from the transistors

The passive matrix LCD relies on transistors for each row and each column of pixels, thus creating a grid that defines the location of each pixel. The color displayed by a pixel is determined by the electricity coming from the transistors

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.

(they have a narrow viewing angle)

Passive-matrix displays are less expensive than active-matrix displays

Another disadvantage is that they don't refresh the pixels very quickly.

If you move the pointer too quickly, it seems to disappear

Animated graphics can appear blurry on a passive matrix monitor.

Most passive matrix screens now use dual-scan LCD technology, which scans the pixels twice as often.

An importance 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 Plasma Panels (gas-discharge display)

A flat-panel display that uses gas plasma technology

Region between two glass plates is filled with a mixture of gases such as neon, xenon.

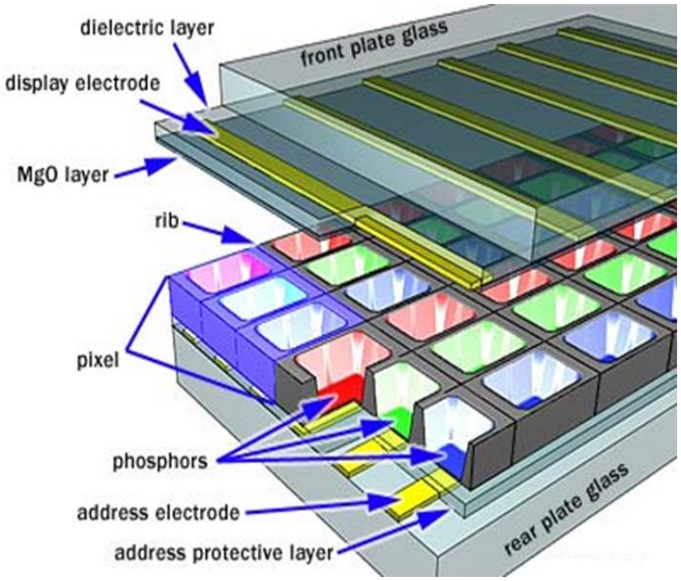
A series of vertical conducting ribbons is placed on one glass panel and a set of horizontal ribbons is built into other gas panel.

Firing voltages applied to a pair of horizontal and vertical conductors cause gas at intersection of the two conductors (release ultraviolet (UV) light) to break down into a glowing plasma of electrons and ions to form an image.

By controlling the amount of voltage applied at various points on grid, each point acts as a pixel(intersection of conductors) to display an image.

Picture definition is stored in a refresh buffer and firing voltages are applied to refresh pixel positions 60 times per second.

Larger screen sizes and higher display quality than LCD, but much more expensive



Scanners

 # Convert any printed image of an object into electronic form by shinning light onto the image and sensing the intensity of light's reflection at any point.

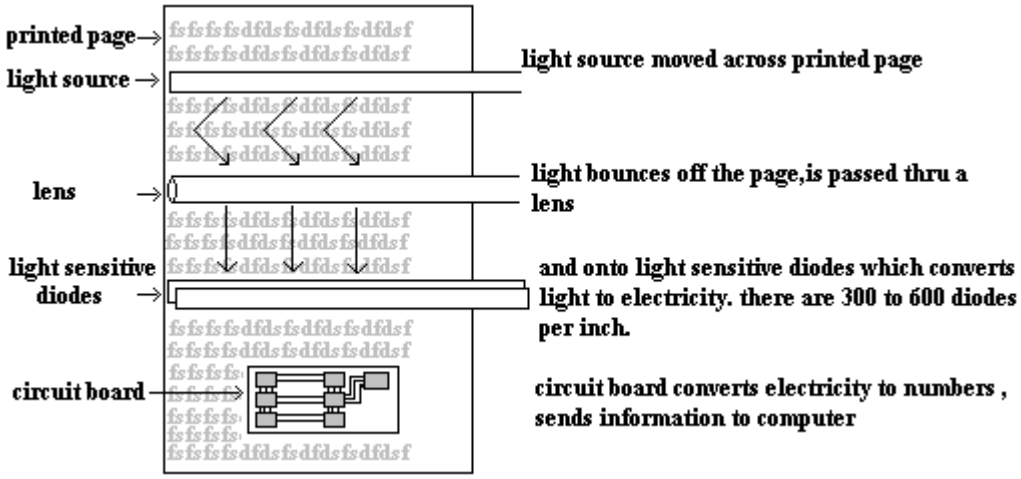
 # Color scanners use filters to separate components of color into primary additive colors (red, green, blue) at each point.

 # R G B are primary additive colors because they can be combined to create any other color.

 # Image scanners translate printed images into electronic format that can be stored into a computer memory.

 # Software is then used to manipulate the scanned electronic image.

 # Images are enhanced or manipulated by graphics programs like Adobe.



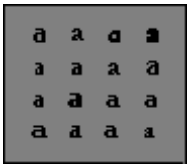
Optical Character Recognition (OCR)

 # For text document we use Optical Character Recognition software to translate image into text that is editable.

 # When a scanner first creates an image from a page the image is stored in computer's memory as bitmap.

 # A bitmap is a grid of dots, each dot represented by one or more bits.

 # OCR software translates the array of dots into text that the computer can interpret as number and letters by looking at each character and trying to match the character with its own assumption about how the image should look like.



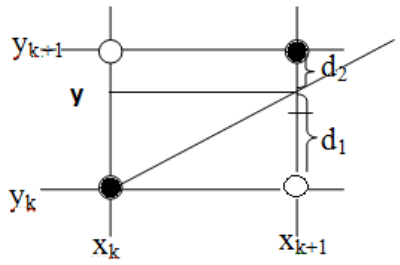
Bresenham's Line Drawing Algorithm for Lines with Slope <=1

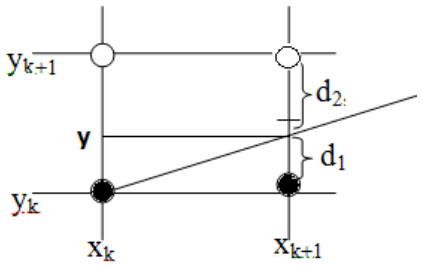
For the line with slope less than equal to one, the pixel positions are determined by sampling at unit 'x' interval i.e. $x_{k+1} = x_k + 1$, with the starting pixel at (x_0, y_0) from left hand.

For any k^{th} step, 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 $(x_{k+1} , y_k + 1)$

At x_{k+1} label vertical pixel separations from ideal line path as d_1 and d_2, 'y' coordinate at x_{k+1} will be

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





The distance of lower pixel from the ideal location $d_1 = y - y_k$ or $d_1 = m (x_k+1) + c - y_k$

The distance of the ideal location from the upper pixel $d_2 = y_k +1 - y$ or $d_2 = y_k +1 - m (x_k+1) - c$

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

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

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

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

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

$P_k = \Delta x.(d_1 - d_2) = 2 \Delta y. x_k - 2\Delta x. y_k + b$ …. (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 $k+1^{th}$ step

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

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

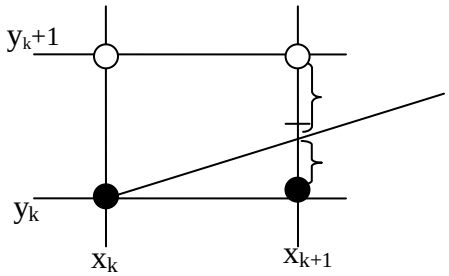
Case 1:

if $P_k <= 0$ then the pixel on scanline

'y_k' is closer to the line path and $y_{k+1} = y_k$

i.e. from equation (iii)

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



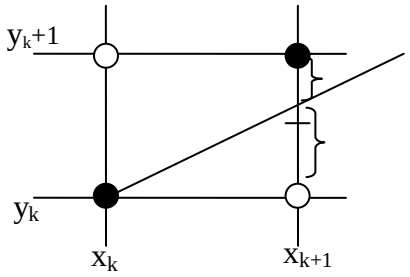
Case 2:

if $P_k > 0$ 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)

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



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

$d_1 - d_2 = 2m(x_0+1) + 2c - 2y_0 - 1$

$= 2mx_0 + 2c - 2y_0 + 2m - 1$

or $d_1 - d_2 = 2m - 1$ since, $2mx_0 + 2c - 2y_0 = 0$

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

Algorithm

For |m| <=1

i. Read xa,ya,xb,yb (Assume −1 <= m <=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 $p_0 = 2\Delta y - \Delta x$

iv. At each x_k along the line starting at k = 0 perform the following 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

Here first we initialize the decision parameter and set the first pixel. Next, during each iteration, we increment 'x' to the next horizontal position, then use the current value of the decision parameter to select the bottom or top pixel (`increment y`) and update the decision parameter and at the end set the chosen pixel.

Advantages

It is a faster incremental algorithm that makes use of integer arithmetic calculations only , avoids floating point computations

So it uses faster operations such as addition/subtraction and bit shifting

CPU intensive rounding off operations are avoided

Disadvantages

It is used for drawing basic lines, antialiasing is not a part of this algorithm so for drawing smooth lines it is not suitable

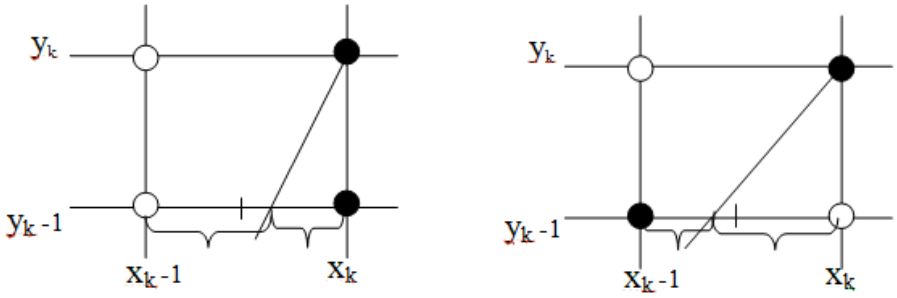
Bresenham's Line Drawing Algorithm for Lines with Negative slope and Magnitude of the slope greater than one (-2,-6) (-4,-9)

For the line with slope greater than equal to one and negative slope, the pixel positions are determined by sampling at unit 'y' interval i.e. $y_{k+1} = y_k + 1$, with the starting pixel at (x_0, y_0) from right hand.

For any k^{th} step, 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, y_k-1) or $(x_k -1, y_k - 1)$

At x_{k+1} label vertical pixel separations from ideal line path as d_1 and d_2, 'y' coordinate at x_{k+1} will be

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



The distance of right pixel from the ideal location $d_1 = x - x_k$ or $d_1 = ((y_k-1) + c)/m - x_k$

The distance of the ideal location from the left pixel $d_2 = x_k -1 - x$ or $d_2 = x_k -1 - ((y_k-1) - c)/m$

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

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

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

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

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

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

$P_k = \Delta y.(d_1 - d_2) = 2.\Delta x.\ y_k - 2.\Delta y.x_k + b$ …. (i) where $b = -2\Delta x + \Delta x.2c + \Delta y$ and P_k is the decision parameter at the k^{th} step

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

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

Now subtracting (i) and (ii)

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

Since the slope of the line is greater than one, we sample in decreasing 'y' direction i.e. $y_{k+1} - y_k = -1$ so,

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

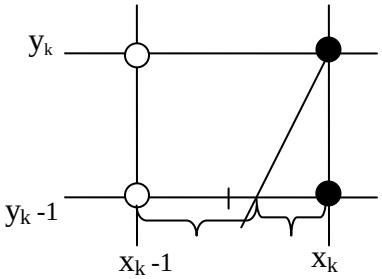
Case 1:

if $P_k < 0$ then the pixel on scanline

'x_k' is closer to the line path and $x_{k+1} = x_k$

i.e. from equation (iii)

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



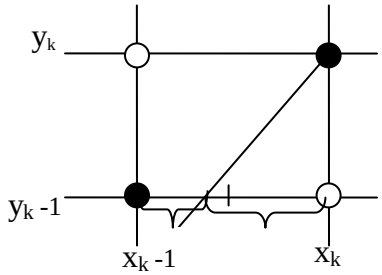
Case 2:

if $P_k >= 0$ then the pixel on scanline

'$x_k - 1$' is closer to the line path and $x_{k+1} = x_k - 1$

i.e. from equation (iii)

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



The Initial Decision Parameter $P_0 = ?$

We have,

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

if the line passes thru (x_0, y_0) then

$d_1 - d_2 = 2\ ((y_0 - 1) - c)/m - 2x_0 + 1$

$= 2y_0/m - 2/m - c/m - 2x_0 + 1$

or $d_1 - d_2 = -2/m + 1$ since, $2y_0/m - c/m - 2x_0 = 0$

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

Circle

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

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

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

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

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

Select next pixel along the circle path according to the sign of circle function evaluated at the midpoint between two candidate pixels.

Start at (0,y) take unit steps in 'x' direction (sample in 'x' direction $x_{k+1} = x_k + 1$)

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

$$P_k = F_{circle}(x_k + 1 , y_k - ½)$$

$$= (x_k + 1)^2 + (y_k - ½)^2 - r^2 \quad \text{---------- (i)}$$

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

$$P_{k+1} = F_{circle}(x_{k+1} + 1 , y_{k+1} - ½)$$

$$= [(x_k + 1) + 1]^2 + (y_{k+1} - ½)^2 - r^2 \quad \text{---------- (ii)}$$

Now subtracting eq (i) and (ii),

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

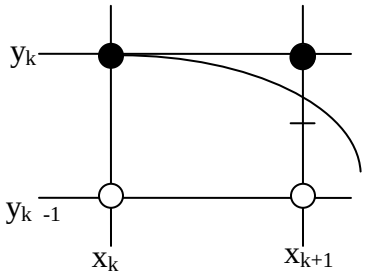
where y_{k+1} is either y_k or $y_k - 1$ depending on the sign of P_k.

<u>Case 1:</u>

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 $P_{k+1} = P_k + 2x_{k+1} + 1$ ---------- (a)



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

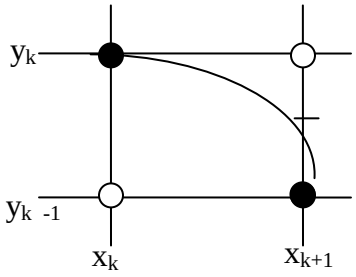
or $2x_{k+1} = 2x_k + 2$

Case 2:

If $P_k >= 0$ then the mid point is outside or on the boundary of the circle, so we select the pixel on scan line '$y_k - 1$' then $y_{k+1} = y_k - 1$

i.e. from equation (iii)

or $P_{k+1} = P_k + 2x_{k+1} - 2y_{k+1} + 1$ --------- (b)



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

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

The initial decision parameter P_0 is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$

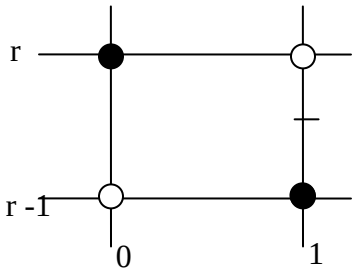
Next pixel to plot is either (1,r) or (1,r - 1)

So, midpoint coordinate position is (1, r - ½)

$F_{circle}(1, r - ½) = 1 + (r - ½)^2 - r^2$

Thus,

$P_0 = 5/4 – r$



If the radius 'r' is specified as an integer, we can simply round P_0 to $P_0 = 1 – r$ (for 'r' an integer)

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

   Otherwise, 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 the circular path centered on (x_c, y_c) and plot the coordinate values:

   $x = x + x_c$, $y = y + y_c$
6. Repeat steps 3 through 5 until x >= y.

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.

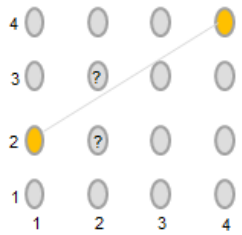
In random scan system, point plotting commands are stored in display list and coordinate values in these instructions are converted to deflection voltages that position the electron beam at that screen location to be plotted during each refresh cycle.

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.

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 (causes stair cases or jaggies to be formed)

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

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



Digital Differential Analyzer Algorithm (DDA)

A scan conversion algorithm for lines computes the coordinates of the pixels that lie on a near an ideal infinitely thin straight line imposed on a 2D Raster grid.

It traces out successive (x,y) values by simultaneously incrementing x and y by small steps proportional to the first derivative of x and y.

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

For points (x1, y1) and (x2,y2) slope m = y2 – y1/x2 - x1 or $\Delta y/\Delta x$

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

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

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

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

For Lines with Positive Slope

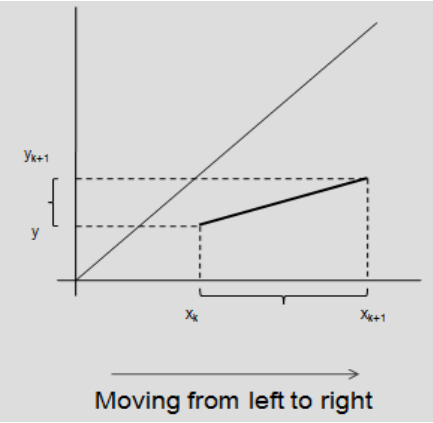
i. For Lines with |m| <=1 (Moving from Left to Right)

The simplest strategy for scan conversion of lines is to compute the slope 'm' as $\Delta y/\Delta x$ to increment x by 1 pixel starting with the leftmost point to calculate the next value of y_i, for each x and to intensify the pixel at $(x_i,|y_i|)$.

Then

$$m = y_{k+1} – y_k / x_{k+1} – x_k$$

$$y_{k+1} = y_k + m.(x_{k+1} – x_k)$$



And if $\Delta x = 1$ then $y_{k+1} = y_k + m$. Thus the unit change in x changes y by m , which is the slope of the line. For all points (x_k,y_k) on the line, we know that if $x_{k+1} = x_k + 1$ then $y_{k+1} = y_k + m$ i.e. the values of x and y are defined in terms of their previous values

Similarly

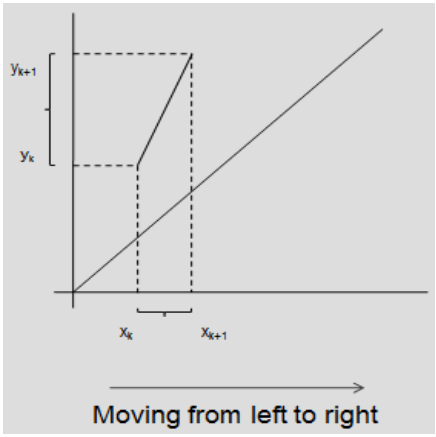
ii. For Lines with slope > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = 1$ and compute successive x value as

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

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

The x value computed must be rounded off to the nearest whole number



iii. For Lines with slope < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = -1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number

iv. For Lines with slope > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

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

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

The x value computed must be rounded off to the nearest whole number

For Lines with Negative Slope

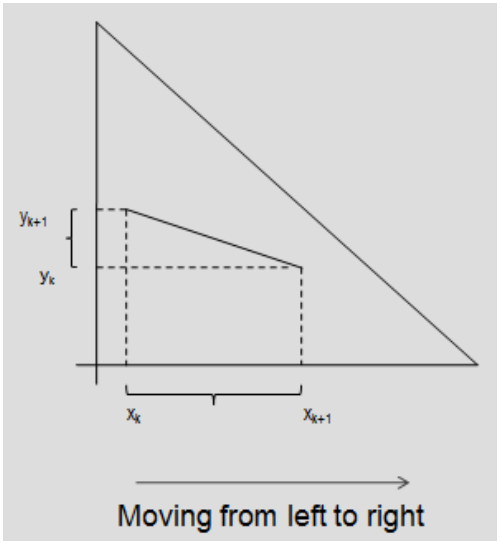
v. For Lines with |m| <=1 (Moving from Left to Right)

Perform unit increment in x direction $\Delta x = 1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = 1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number



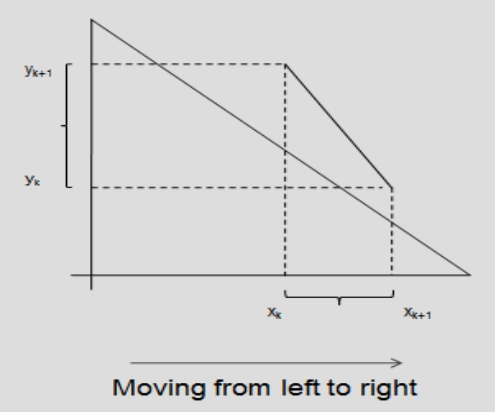
vi. For Lines with |m| > 1 (Moving from Left to Right)

Perform unit increment in y direction $\Delta y = -1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = -1$ and compute successive x value as

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

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

The x value computed must be rounded off to the nearest whole number



vii. For Lines with |m| < 1 (Moving from Right to Left)

Perform unit increment in x direction $\Delta x = - 1$ as $\Delta x > \Delta y$ i.e. $x_{k+1} - x_k = -1$ and compute successive y value as

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

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

The y value computed must be rounded off to the nearest whole number

viii. For Lines with |m| > 1 (Moving from Right to Left)

Perform unit increment in y direction $\Delta y = 1$ as $\Delta y > \Delta x$ i.e. $y_{k+1} - y_k = 1$ and compute successive x value as

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

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

The x value computed must be rounded off to the nearest whole number

This algorithm is based on floating point arithmetic so it is slower than Bresenham's line drawing algorithm for drawing lines as Bresenham's line drawing algorithm is based on integer arithmetic approach.

Algorithm:

```
void lineDDA (int xa, int ya, int xb, int yb){
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
        if (abs (dx) > abs (dy))  steps = abs (dx) ;
        else steps = abs (dy);
      xIncrement = dx / (float) steps;
      yIncrement = dy / (float) steps
      setpixel (ROUND(x), ROUND(y));
      for (k=O; k<steps; k++){
         x += xIncrement;
         y += yIncrement;
        setpixel (ROUND(x), ROUND(y));
      }
}
```

Ellipse

Definition: An ellipse is defined as set of points such that sum of the distances from the two fixed points is the same for all points. If the distance to two fixed points from any point P(x,y) on ellipse are d_1 , d_2 then general equation of an ellipse is $d_1 + d_2 = \text{constant}$

Or expressing distance d_1 , d_2 in terms of focal coordinates $F_1 = (x_1, y_1)$ and $F_2 = (x_2, y_2)$

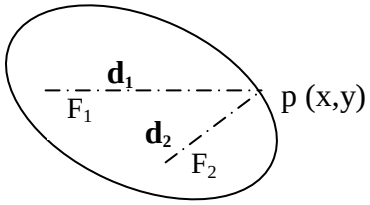
We have,

$$\sqrt{(x - x_1)^2 + (y - y_1)^2} + \sqrt{(x - x_2)^2 + (y - y_2)^2} = \text{constant}$$

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

$$x^2 / r^2_x + y^2 / r^2_y = 1$$



or $F_{ellipse}(x,y) = r^2_y x^2 + r^2_x y^2 - r^2_x r^2_y$

now

$F_{ellipse}(x,y)$	< 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 $F_{ellipse}(x,y)$ serves as the decision parameter

We select next pixel along the ellipse path according to the sign of ellipse function evaluated at the midpoint between two candidate pixels.

Start at $(0, r_y)$ take unit steps in 'x' direction until we reach boundary region 1 and 2 then switch to unit steps in 'y' direction for remainder of curve in the first quadrant.

At each step test the value of the slope of the curve. The ellipse slope is given by

$$2 r^2_y x + 2 r^2_x y \, dy/dx = 0$$

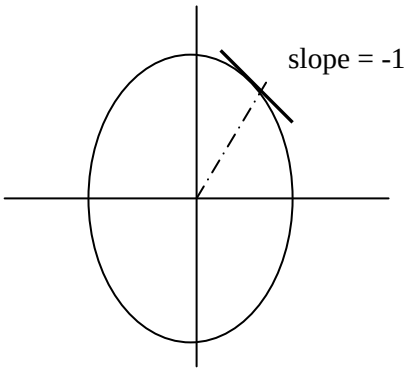
$$dy/dx = - 2 r^2_y x / 2 r^2_x y$$

At the boundary between region 1 and 2 $dy/dx = -1$

$$\text{so, } 2 r^2_y x = 2 r^2_x y$$

and we move out of the region 1 when

$$2 r^2_y x >= 2 r^2_x y$$



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

$$P1_k = F_{ellipse}(x_k + 1 , y_k - ½)$$

$$= r^2_y (x_k + 1)^2 + r^2_x (y_k - ½)^2 - r^2_x r^2_y \quad \text{---------- (i)}$$

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

$$P1_{k+1} = F_{ellipse}(x_{k+1} + 1 , y_{k+1} - ½)$$

$$= r^2_y [(x_k + 1) +1]^2 + r^2_x (y_{k+1} - ½)^2 - r^2_x r^2_y \quad \text{---------- (ii)}$$

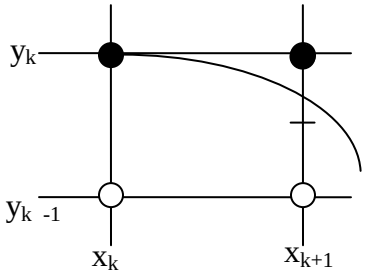
Now subtracting eq (i) and (ii),

$$P1_{k+1} = P1_k + 2r^2_y (x_k + 1) + r^2_y + r^2_x [(y_{k+1} - ½)^2 - (y_k - ½)^2] \quad \text{--------- (iii)}$$

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

<u>Case 1:</u>

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 $2r^2_y x_{k+1} + r^2_y$ i.e. from equation (iii)



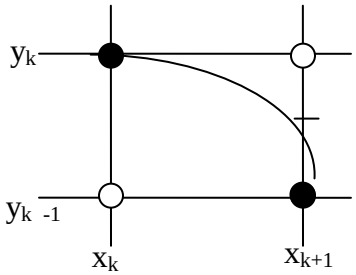
Or $$P1_{k+1} = P1_k + 2r^2_y x_{k+1} + r^2_y \quad \text{---------- (a)}$$

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

or $2r^2_y x_{k+1} = 2r^2_y x_k + 2 r^2_y$

<u>Case 2:</u>

if $P1_k >= 0$ 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 $2r^2_y x_{k+1} - 2r^2_x y_{k+1}$ i.e. from equation (iii)



Or $$P1_{k+1} = P1_k + 2r^2_y x_{k+1} - 2r^2_x y_{k+1} + r^2_y \quad \text{---------- (b)}$$

Where $2r^2_x y_{k+1} = 2r^2_x y_k - 2 r^2_x$

or $2r^2_y x_{k+1} = 2r^2_y x_k + 2 r^2_y$

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 $(1, r_y - ½)$

$$F_{ellipse}(1, r_y - ½) = r^2_y + r^2_x (r_y - ½)^2 - r^2_y r^2_y$$

Thus,

$$P1_0 = r^2_y + ¼ r^2_x - r^2_x r_y$$

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. we choose either $(x_k, y_k - 1)$ **or** $(x_{k+1}, y_k - 1)$

So, midpoint coordinate position is $(x_k + ½, y_k - 1)$

$$F_{ellipse}(x_k + ½, y_k - 1)$$

$$\text{Or, } P2_k = r^2_y (x_k + ½)^2 + r^2_x (y_k - 1)^2 - r^2_x r^2_y \quad \text{---------- (iv)}$$

now, at next sampling position, the next pixel to plot will either be

$(x_{k+1}, y_{k+1} - 1)$ **or** $(x_{k+1} + 1, y_{k+1} - 1)$

thus,

$$F_{ellipse}(x_{k+1} + ½, y_{k+1} - 1)$$

$$\text{Or, } P2_{k+1} = r^2_y (x_{k+1} + ½)^2 + r^2_x (y_{k+1} - 1)^2 - r^2_x r^2_y$$

$$= r^2_y (x_{k+1} + ½)^2 + r^2_x [(y_k - 1) - 1]^2 - r^2_x r^2_y \quad \text{---------- (v)}$$

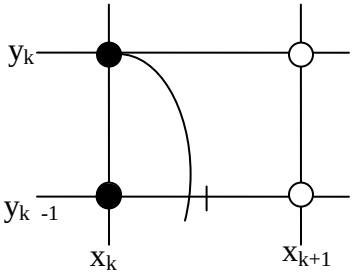
Now subtracting eq (iv) and (v),

$$P2_{k+1} = P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [(x_{k+1} + ½)^2 - (x_k + ½)^2] \quad \text{---------- (vi)}$$

where x_{k+1} **is either** x_k **or** $x_k + 1$ **depending on the sign of** $P2_k$**.**

Case 1:

if $P2_k > 0$ **then the mid point is outside the boundary of the ellipse, so we select the pixel at '**x_k**'**



Or $P2_{k+1} = P2_k - 2r^2_x (y_k - 1) + r^2_x$

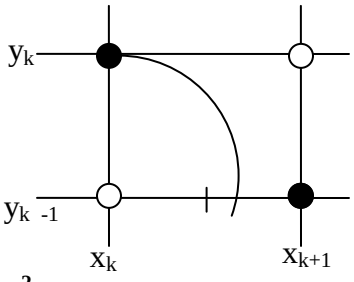
$$= P2_k - 2r^2_x y_{k+1} + r^2_x \quad \text{---------- (c)}$$

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

or $2r^2_x y_{k+1} = 2r^2_x y_k - 2r^2_x$

Case 2:

if $P2_k$ < = 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)



Or $P2_{k+1} = P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [(x_{k+1} + ½)^2 - (x_k + ½)^2]$

$= P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [(x_k + 1) + ½)^2 - (x_k + ½)^2]$

$= P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [(x_k + 3/2)^2 - (x_k + ½)^2]$

$= P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [x^2_k + 3x_k + 9/4 - x^2_k - x - 1/4]$

$= P2_k - 2r^2_x (y_k - 1) + r^2_x + r^2_y [2x_k + 2]$

$= P2_k - 2r^2_x y_{k+1} + r^2_x + 2r^2_y x_{k+1}$ --------- **(d)**

Where $2r^2_x y_{k+1} = 2r^2_x y_k - 2 r^2_x$

or $2r^2_y x_{k+1} = 2r^2_y x_k + 2 r^2_y$

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

$P2_0 = F_{ellipse}(x_0 + ½, y_0 - 1)$

$= r^2_y (x_0 + ½)^2 + r^2_x (y_0 - 1)^2 - r^2_x r^2_y$

FILLED-AREA PRIMITIVES

Objects can be filled with solid-color or patterned polygon area in some graphical packages.

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

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

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.**

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

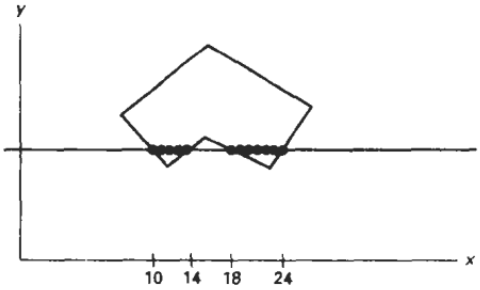
Set each pixel lying on scan line running from left edge to right with same pixel value, each span from x_{max} to x_{min}

```
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);
```

Scan line Polygon Fill Algorithm

For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges.

These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color.

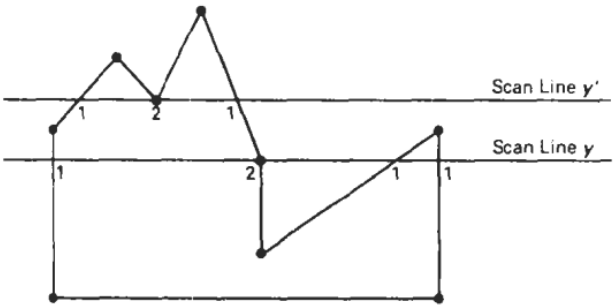


Four pixel intersection positions with the polygon boundaries define two stretches of interior pixels from x = 10 to x = 14 and from x = 18 to x = 24.

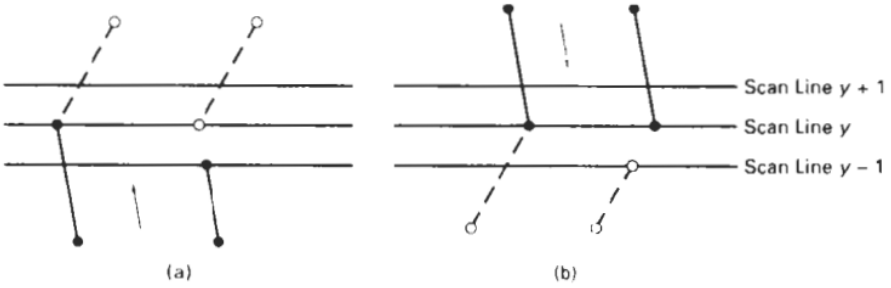
For a scan line passing through a vertex with two edges intersecting at that position, adding two points to the list of intersections for the scan line gives a solution.

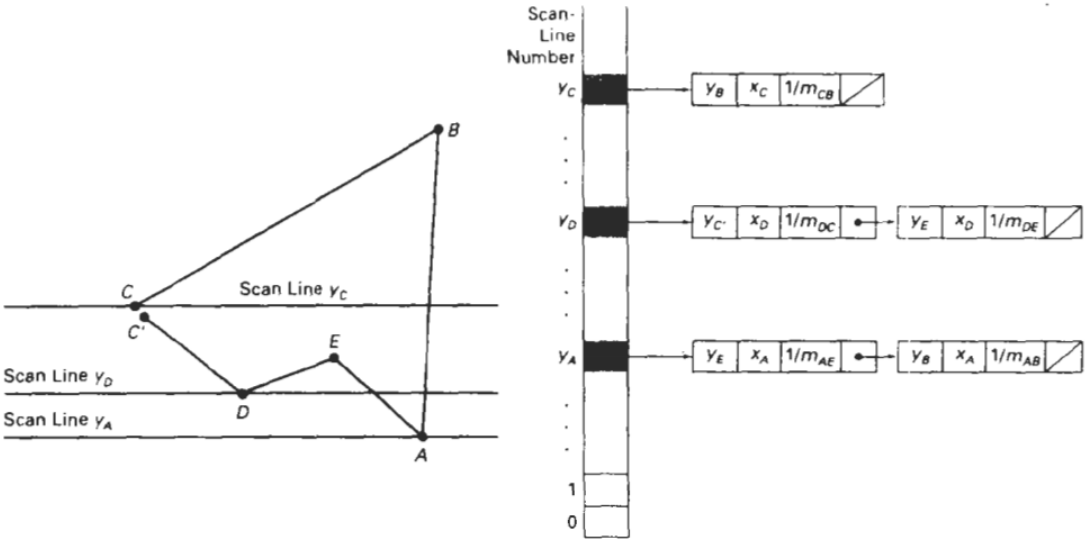
Scan line y intersects five polygon edges. Scan line y', however, intersects an even number of edges although it also passes through a vertex.

For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line. But for scan line y', the two intersecting edges are both above the scan line



One way to resolve the question as to whether we should count a vertex as one intersection or two is to shorten some polygon edges to split those vertices that should be counted as one intersection



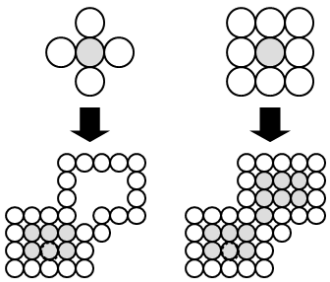


Boundary-Fill Algorithm

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

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

This approach requires boundary to be specified, but interior pixels can be of multiple colors



Four connected approach considers neighboring pixels as the ones that are to the right, left, bottom and top of the seed pixel

Eight connected approach also considers diagonal pixels as neighboring pixels so it includes pixels to the right, left, bottom , top then four more diagonal ones right-

top, right-bottom, left-top , left-bottom

Algorithm:

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

```
void boundaryFill (int x, int y, int fill, int boundary) (
int  current = getpixel (x, y);
      if ((current != boundary) && (current != fill)) {
            setcolor (fill) ;
            setpixel (x, y);
            boundaryfill (x+1, y, fill, boundary);
            boundaryFill (x-1, y, fill, boundary);
            boundaryFill (x, y+1, fill, boundary);
            boundaryFill (x, y-1, fill, boundary) ;
      }
}
```

Flood-Fill Algorithm

Used for filling an area that is not defined within a single color boundary.

Such areas can be filled by replacing a specified interior color instead of searching for a boundary color value.

This approach does not require boundary to be specified, but interior pixels must be of the same color

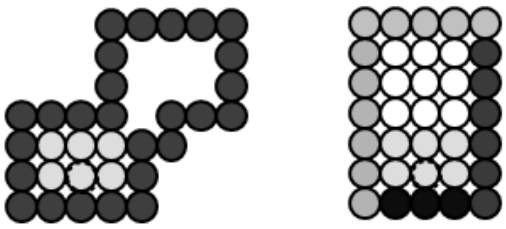
Algorithm:

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.

```
void floodFill (int x, int y, int fillcolor, int oldcolor) {
      if (getpixel (x, y) == oldcolor) {
            setcolor (fillcolor);
            setpixel (x, y);
            floodFill (x+1, y, fillColor, oldColor);
            floodfill (x-1, y, fillcolor, oldcolor);
            floodFill (x, y+1, fillcolor, oldcolor);
            floodFill (x, y-1, fillColor, oldcolor);
      }
}
```



2D

A point is represented in 2 dimension by its coordinates

These two values are specified as elements of 1 row 2 column matrix

In two dimension [x, y] **In three dimension [x, y,z]**

Or alternatively a point is represented by a 2 row 1 column matrix

In two dimension $\begin{bmatrix} x \\ y \end{bmatrix}$ **In three dimension** $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$

and are called *Position Vectors*.

Vector has single direction and length and may be denoted by [D_x ,D_y].

D_x indicates how far to move along x axis direction.

D_y indicates how far to move along y axis direction.

Vectors tell us how far and what direction to move but hot where to start. e.g. command for pen to move so far from its current position in given direction.

A series of points each of which is a position vector relative to some coordinate system is stored in computer as matrix or array of numbers.

Position of these points is controlled by manipulating matrix that defines the points.

A straight line is transformed by transforming its end points then redrawing the line between the transformed end points.

A polygon is transformed by transforming its vertices then redrawing the line between the transformed vertices.

A curve is transformed by transforming its control point such as it's center point in case of a circle and then redrawing the curve using the transformed control points.

Clipping in Raster World

It is a procedure that identifies those operations of picture that are either inside or outside is called clipping.

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.

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 a picture to be selected for copying, moving, erasing or duplicating etc.

Displaying only those parts of picture that are within window area, discard everything outside window.

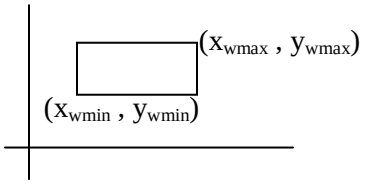
World coordinate clipping removes those primitives outside window from further consideration thus eliminating process necessary to transfer those primitives to device space.

View port clipping requires transformation to device coordinate be performed for all objects including those outside window area.

Point Clipping

Assuming the clip window is a rectangle, the lower left corner of the window is define by $x = x_{wmin}$ and $y = y_{wmin}$ and the upper left corner of the window is define by $x = x_{wmax}$ and $y = y_{wmax}$ a point p=(x,y) if visible for display if the following inequalities are satisfied:

$$x_{wmin} \leq x \leq x_{wmax}$$
$$y_{wmin} \leq y \leq y_{wmax}$$



Applied to scenes involving explosions or sea foam that are modeled with points distributed in some region of the scene.

Cohen Shutherland Line Clipping Algorithm

It is based on a coding scheme

Makes clever use of bit operations to perform this test efficiently

For each end point , a 4 bit binary code is used

The 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 co-related 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.

The region codes are shown

1001	**1000**	**1010**
0001	**0000**	**0010**
0101	**0100**	**0110**

Algorithm:

Step 1: Establish the region codes for all line end points:

Bit 1 is set to 1 if $x < x_{wmin}$ otherwise set it to 0 Bit 3 is set to 1 if $y < y_{wmin}$ otherwise set it to 0

Bit 2 is set to 1 if $x > x_{wmax}$ otherwise set it to 0 Bit 4 is set to 1 if $y > y_{wmax}$ otherwise set it to 0

Step 2: Determine which lines are completely inside the window and which are completely outside , using the following tests:

a. If both end points of the line have region codes 0000 the line is completely inside the window
b. 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)

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

slope (m) = $(y_2 - y_1)/(x_2 - x_1)$

a. If bit 1 is 1 then the line interests with the left boundary and $y_i = y_1 + m * (x - x_1)$ where $x = x_{wmin}$
b. If bit 2 is 1 then the line interests with the right boundary and $y_i = y_1 + m * (x - x_1)$ where $x = x_{wmax}$
c. If bit 3 is 1 then line interests with the bottom boundary and $x_i = x_1 + (y - y_1)/m$ where $y = y_{wmin}$
d. If bit 4 is 1 then line interests with the upper boundary and $x_i = x_1 + (y - y_1)/m$ where $y = y_{wmax}$

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

Homogenous coordinate

Consider the effect of general 2 by 2 transformation applied to the origin

So for the origin, Origin is invariant under general 2 by 2 transformation.

This limitation is overcome by homogenous coordinates

It is necessary to be able to modify position of origin i.e. to transform every point in 2 dimension plane.

This can be accomplished by translating origin or any other point in 2 dimension plane

If $x^* = ax + by + m$
$y^* = bx + dy + n$

In homogenous coordinate representation we add third coordinate to a point .

Instead of representing by a pair of number (x , y) each point is represented by a triple (x , y , h).

We say that 2 sets of homogenous coordinates (x, y, h) and (x*, y*,h*) represent the same point if and only if one is multiple of another i.e. (2,3,6) , (4,6,12) are same points represented by different coordinate triples.

In order to transform a point (x, y) into homogenous representation we choose a non zero number 'n' and form a vector [hx , hy , h] and h is called scale factor or homogenous coordinate parameter.

For point [2,3] in 2 dimensional space it's representation in homogenous coordinated will be

[2, 3, 1] for h = 1
[4, 6, 2] for h = 2
[-2,-3,-1] for h = -1

Affine Transformation: a transformation that preserves collinearity (i.e. points lying on a line initially still lie on a line after transformation) and ratios of distances
It is a combination of single transformations such as translation, rotation or reflection on an axis

Hence the general transformation matrix is of the form

$$[T] = \begin{bmatrix} a & b & m \\ c & d & n \\ 0 & 0 & 1 \end{bmatrix}$$

hence for translation

$P^* = T(t_x, t_y) \cdot P$

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

now every point in 2 dimension plane evern origin ($x = y = 0$) can be transformed.
Similarly, rotation transformation equation about coordinate origin EW

$P^* = R(0) \cdot P$

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 0 & -\sin 0 & 0 \\ \sin 0 & \cos 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x\cos 0 - y\sin 0 \\ x\sin 0 + y\cos 0 \\ 1 \end{bmatrix}$$

Scaling transformation relative to coordinate origin is

$P^* = S(s_x, s_y) \cdot P$

$$\begin{bmatrix} x^* \\ y^* \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ 1 \end{bmatrix}$$

POLYGON CLIPPING

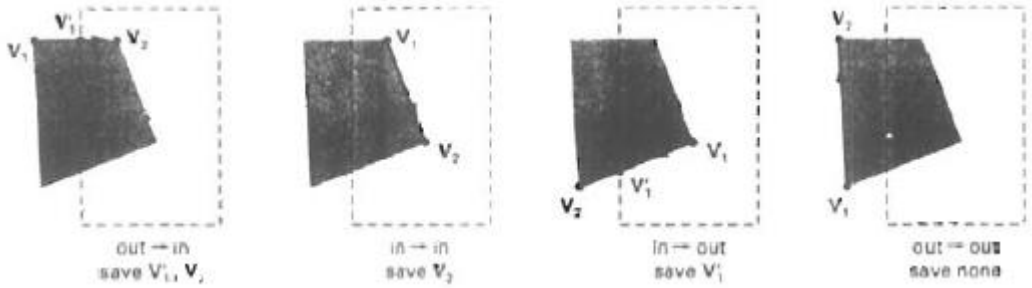
Polygons can be clipped by modifying the line-clipping algorithm. **A** polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments, depending on the orientation of the polygon to the clipping window. The portion needs to be displayed is a bounded area after clipping. Polygon clipping algorithm generates one or more closed areas that are then scan converted for the appropriate area fill. The output of a polygon clipper should be a sequence of vertices that defines the clipped polygon boundaries.

The Sutherland-Hodgeman Polygon-Clipping Algorithm

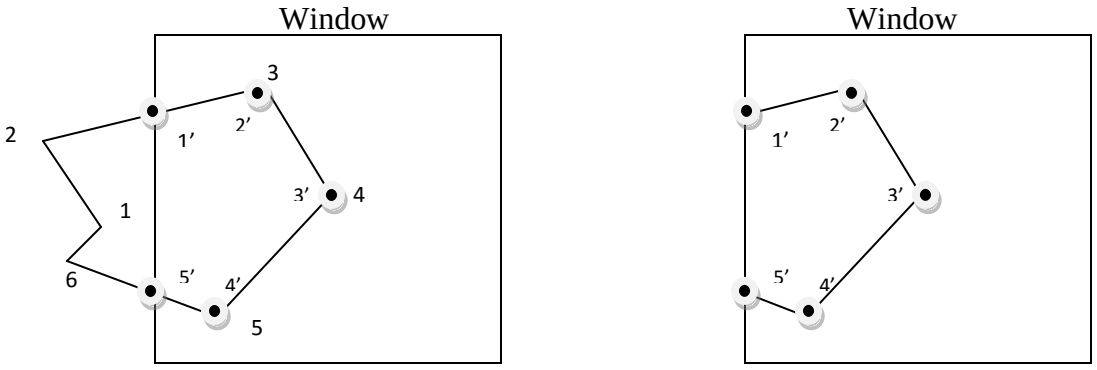
A polygon can be clipped by processing the polygon boundary as a whole against each (clip) window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary (left, right, bottom and top) in turn.
Beginning with the initial set of polygon vertices, first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then be successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper. At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper

Four possible cases when processing vertices in sequence around perimeter of a polygon:

As each pair of adjacent polvgon 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.



Example



Before Clipping

After Clipping

i. Process area in figure against left window boundary .

ii. a. Vertices 1,2 are outside of boundary

b. Moving along to vertex 3 which is inside calculate the intersection . Save both intersection point and vertex 3

c. Vertices 4 and 5 are determined to be inside save them both

d. 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

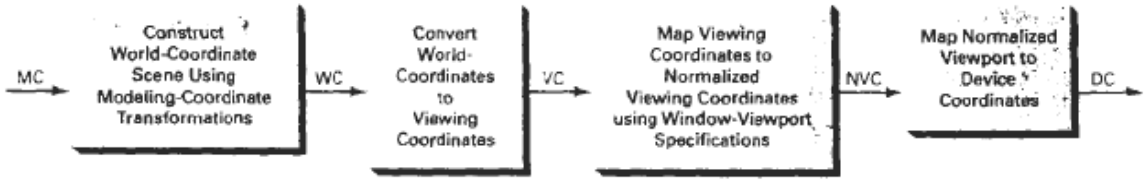
The final set of vertices of the clipped polygon are 1' 2' 3' 4' 5'

THE VIEWING PIPELINE: 2D

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.



First, we construct the scene in World Coordinates from objects modeled in their individual coordinate systems called **Modeling Coordinate System**

Next. to obtain a particular orientation for the window, we can set up a two-dimensional **Viewing-Coordinate System** in the world-coordinate plane, and define a window

In the viewing-coordinate system, the viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations for rectangular windows. Once the viewing reference frame is established, descriptions from world coordinates are transferred to viewing coordinates. So **clipping** and **2D transformations** take place in bringing objects from World Coordinate System to Viewing Coordinate System.

Then a viewport in normalized coordinates (in the range from **0** to 1) is defined and the viewing-coordinate description of the scene are mapped to **normalized coordinates**.

At the final step, the parts of the picture that are outside the viewport are clipped, and the contents of the viewport are transferred to **device coordinates**.

By changing the position of the viewport, we can view objects at different positions on the display area of an output device. Also, by varying the size of viewports, we can change the size and proportions of displayed objects. We achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport.

3D Viewing Transformation

At the first step, a scene is constructed by transforming object descriptions from **modeling coordinates** to **world coordinates**.

Next, a view mapping convert: the world descriptions to **viewing coordinates**.

At the projection stage, the viewing coordinates are transformed to **projection coordinates**, which effectively converts the view volume into a rectangular parallelepiped.

Then, the parallelepiped is mapped into the unit cube, a normalized view volume called the **normalized projection coordinate system**.

The mapping to normalized projection coordinates is accomplished by transforming points within the rectangular parallelepiped into a position within a specified three-dimensional viewport, which occupies part or all of the unit **cube.**

Finally, at the workstation stage, normalized projection coordinates are converted to **device coordinates** for display. The normalized view volume is a region defined by the planes

$$x = 0, \quad x = 1, \quad y = 0, \quad y = 1, \quad z = 0, \quad z = 1$$

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. We can express the three-dimensional transformation matrix for these operations in the form

$$\begin{bmatrix} D_x & 0 & 0 & K_x \\ 0 & D_y & 0 & K_y \\ 0 & 0 & D_z & K_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We can express the three-dimensional transformation matrix for these operations in the form

$$D_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$D_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

$$D_z = \frac{zv_{max} - zv_{min}}{z_{back} - z_{front}}$$

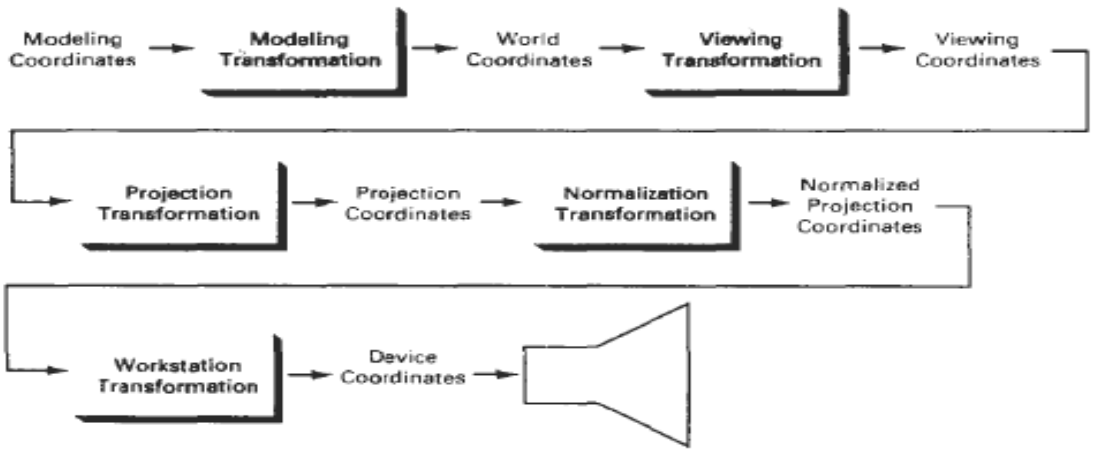
Factors D_x D_y , and D_z are the ratios of the dimensions of the viewport and regular parallelepiped view volume in the *x,* y, and z directions where the view-volume boundaries are established by the window limits ***(xw$_{min}$, xw$_{max}$, , yw$_{min}$, yw$_{max}$)*** and the positions z_{front} and z_{back} of the front and back planes.

Viewport boundaries are set with the coordinate values ***xv$_{min}$, xv$_{max}$, , yv$_{min}$, yv$_{max}$,*** z***v$_{min}$*** and ***zv$_{max}$***. The additive translation factors $\mathbf{K_x}$, $\mathbf{K_y}$, and $\mathbf{K_z}$ in the transformation are

$$K_x = xv_{min} - xw_{min} D_x$$

$$K_y = yv_{min} - yw_{min} D_y$$

$$K_z = zv_{min} - z_{front} D_z$$



Window /View port

Windows are areas on screen where graphical information can be displayed

View ports refer to rectangular areas inside window that display graphical data.

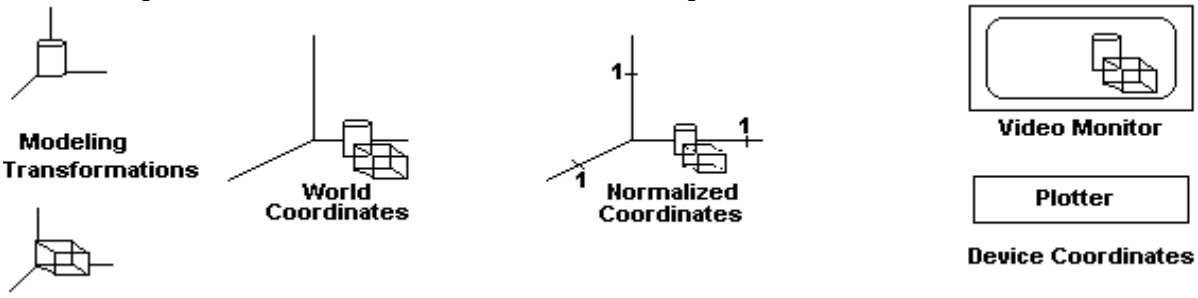
View ports are or can be of different sizes but are always smaller than size if window.

For practical applications we need a transformation to translate and scale window to any size by moving it to specified rectangular area on screen

This area is commonly called the view port

The choice of window decides what we want to see on display and choice of view port decides where we want to see on display screen

This concept allows user to divide screen into virtual partitions.



The display hardware divides screen into a number of pixels arranged in a grid with each pixel are associated its x , y coordinates.

Top left corner of screen is called origin of coordinate system

Value of 'x' coordinate increases from left to right

Value of 'y' coordinate increases from top towards bottom.

These coordinates are reference d to as device for screen coordinates.

For the VGA graphics card the size of display grid is 640 x 480.

Similarly suitable coordinate system can be selected for objective in space

The coordinates so defined are world coordinates

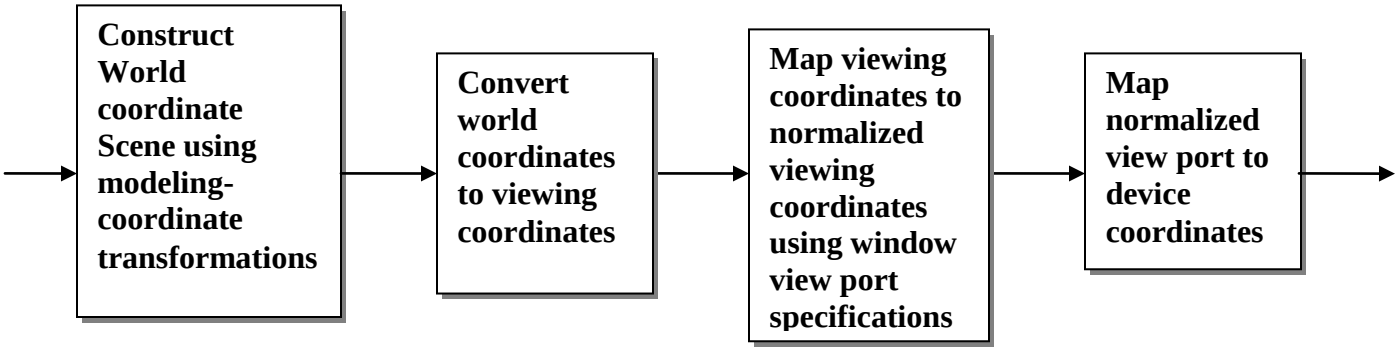
It may sometimes be desirable to select a part of object or drawing for display in order to obtain sufficient detail of the object on display

The part of interest is enclosed n a rectangular boundary called a window

For displaying one has to convert world coordinates into screen coordinates
This transformation is called viewing transformation

In general view transformation consists of operations such as scaling, translation , rotation etc.

Window to View Port Transformation (2-D Viewing Transformation Pipeline)



A point in position(x_w,y_w) in window is mapped into position (x_v, y_v) in the view port

To maintain same relative placement in view port as in window we require,

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

and,

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

solving equations for view port position (x_v , y_v)

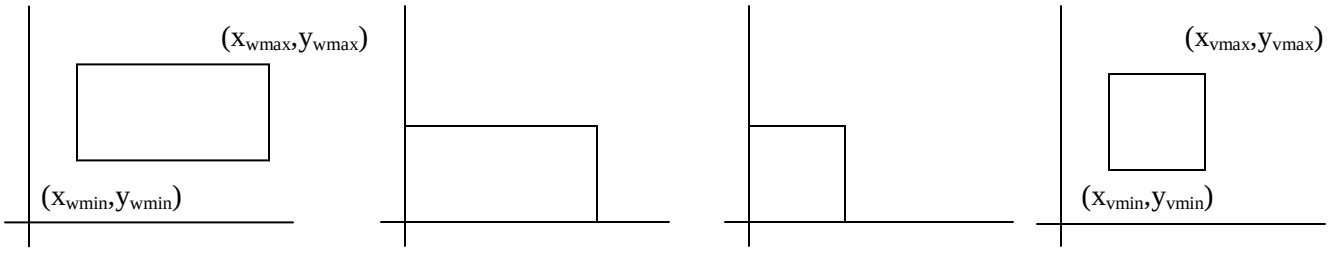
$$x_v = x_{vmin} + (x_w - x_{wmin}) \frac{(x_{vmax} - x_{vmin})}{(x_{wmax} - x_{wmin})} = x_{vmin} + (x_w - x_{wmin}) . s_x$$

$$y_v = y_{vmin} + (y_w - y_{wmin}) \frac{(y_{vmax} - y_{vmin})}{(y_{wmax} - y_{wmin})} = y_{vmin} + (y_w - y_{wmin}) . s_y$$

where, s_x and s_y are the scaling factors.

Sequence of Transformations

i. Perform scaling transformation using fixed point position(x_{wmin} , y_{wmin}) that scales the window area to the size of the view port
ii. Translate the scaled window area to the position of the view port



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

The mapping of a part of world coordinate scene to device coordinate is referred to as **viewing transformation.**

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 $(\mathbf{x_{wmin}, y_{wmin}})$and $(\mathbf{x_{wmax}, y_{wmax}})$ where as the view port coordinates be $(\mathbf{x_{vmin}, y_{vmin}})$and $(\mathbf{x_{vmax}, y_{vmax}})$

Therefore the viewing transformation is as follows:

1. We have to translate the window to the origin by

   $\mathbf{T_x = -x_{wmin}}$ and $\mathbf{T_y = -y_{wmin}}$

2. Then scale the window such that its size is matched to the view port using

   $\mathbf{S_x = (x_{vmax} - x_{vmin})/(x_{wmax} - x_{wmin})}$ $\mathbf{S_y = (y_{vmax} - y_{vmin})/(y_{wmax} - y_{wmin})}$

3. Again translate it by $\mathbf{T_x = x_{vmin}}$ and $\mathbf{T_y = y_{vmin}}$

All these steps can be represented by the following composite transformation:

$$\mathbf{CM = T_V * S_{wv} * T_w}$$

Where CM = Composite Transformation (here, Viewing Transformation)

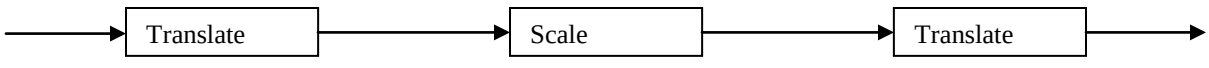
T_w = Translate window to origin

T_V = Translate view port to original location $(\mathbf{x_{vmin}, y_{vmin}})$ $(\mathbf{x_{vmax}, y_{vmax}})$

S_{wv} = Scaling of window to the size of view port

$$T_w = \begin{bmatrix} 1 & 0 & -x_{wmin} \\ 0 & 1 & -y_{wmin} \\ 0 & 0 & 1 \end{bmatrix} \quad S_{wv} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad T_v = \begin{bmatrix} 1 & 0 & x_{vmin} \\ 0 & 0 & y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

The Viewing Transformation can be shown as follow:



3D Clipping

In **2D Clipping** a **window is considered** as a clipping boundary **but** in **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**

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

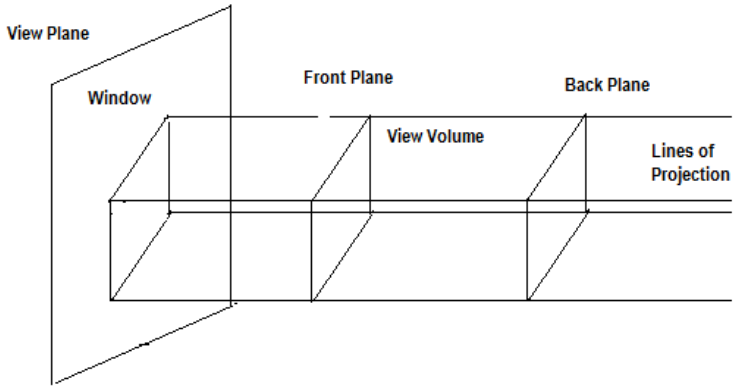
The view volume has **6 sides**: Left, Right , Bottom, Top, Near and Far

Cohen Sutherland 's region code approach **can be extended** for 3D clipping as well

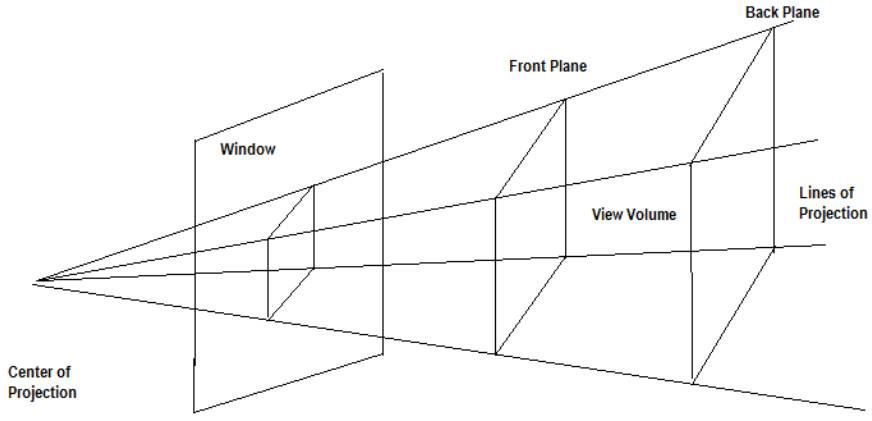
In 2D, a point is checked if it is inside the visible window/region or not but in 3D clipping a point is compared against a plane

View Volume in case of

i. Parallel Projection



ii. Perspective Projection



The front and back planes are positioned relative to the view reference point in the direction of the view plane normal

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 $x < x_{vmin}$ Bit 2 is set to 1 if $x > x_{vmax}$

Bit 3 is set to 1 if $y < y_{vmin}$ Bit 4 is set to 1 if $y > y_{vmax}$

Bit 5 is set to 1 if $z < z_{vmin}$ Bit 6 is set to 1 if $z > z_{vmax}$

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

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:

$x= x_1 + (x_2 – x_1) u$ $y= y_1 + (y_2 – y_1) u$ $z= z_1 + (z_2 – z_1) u$

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

Matrix Representation of 3D Transformations

2D transformations can be represented by 3 x 3 matrices using homogenous coordinates, so

3D transformations can be represented by 4 x 4 matrices, providing we use homogeneous coordinate representations of points in 2 space as well.

Thus instead of representing a point as (x , y ,z) , we represent if as (x , y , z, H), where two these quadruples represent the same point it one is a non zero multiple of the other the quadruple (0,0,0,0) is not allowed.

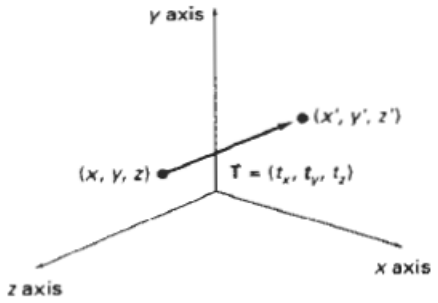
A standard representation of a point (x , y ,z ,H) with H not zero is given by (x/H , y/H, z/H, 1).

Transforming the point to this form is called homogenizing.

<u>Translation:</u>
A point is translated from position P=(x,y,z) to position P' = (x',y',z') with the matrix operation

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

or P' = T.P



Parameters t_x , t_y ,t_z specify translation distances for the coordinate directions x ,y and z.

This matrix representation is equivalent to three equations:

$$x' = x + t_x \qquad y' = y + t_y \qquad z' = z + t_z$$

<u>Scaling:</u>
Scaling changes size of an object and repositions the object relative to the coordinate origin.

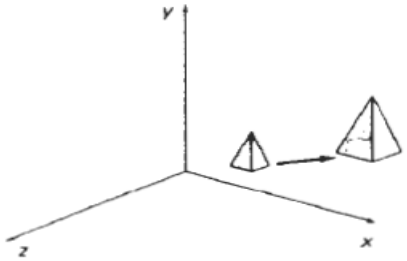
If transformation parameters are not all equal then figure gets distorted

So we can preserve the original shape of an object with uniform scaling($s_x = s_y = s_z$)

Matrix expression for scaling transformation of a position P = (x,y,z) relative to the coordinate origin can be written as :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

or P' = S . P

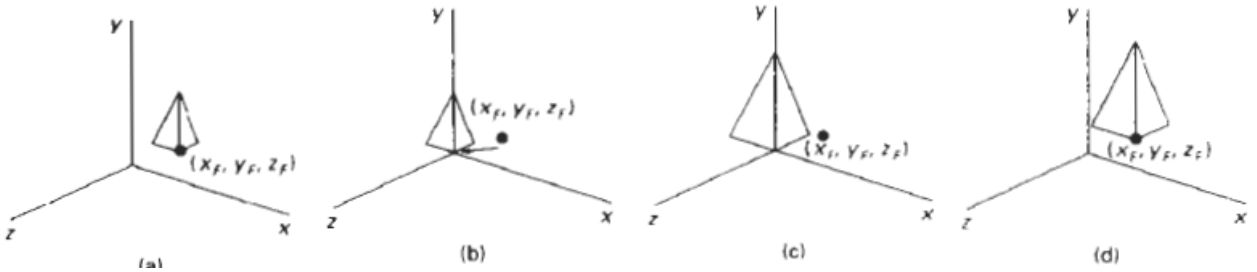


Scaling with respect to a selected fixed point

(x_f, y_f, z_f) can be represented with:

i. **Translate fixed point to the origin**
ii. **Scale object relative to the coordinate origin**
iii. **Translate fixed point back to its original position**

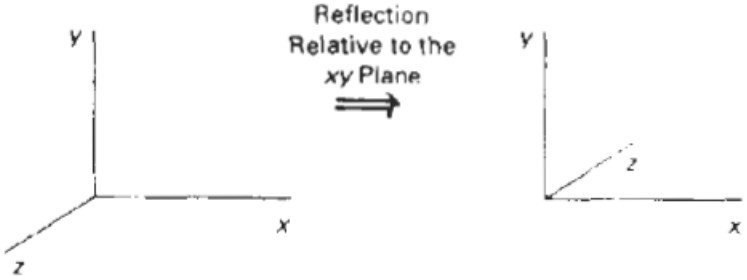
or $T(x_f, y_f, z_f).\ S(s_x, s_y, s_z).T(-x_f, -y_f, -z_f)$



Reflection:

Reflections with respect to a plane are equivalent to 180* rotations in four dimensional space.

$$RFz = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



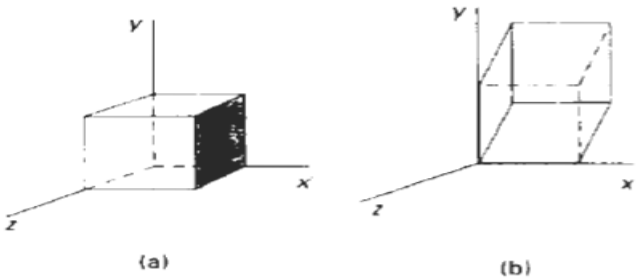
This transformation changes the sign of the z coordinates, leaving the x and y coordinate values unchanged.

Transformation matrices for inverting x and y values are defined similarly, as reflections relative to the yz plane and xz plane.

Shearing:

Shearing transformations are used to modify object shapes.
E.g. shears relative to the z axis:

$$SHz = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



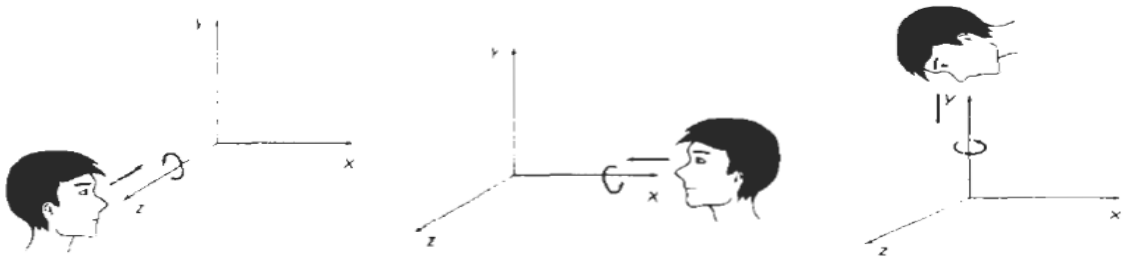
where parameters a and b assume any real values.

It alters the x and y coordinate values by an amount that is proportional to the z value while leaving the z coordinate unchanged.

Shearing matrices for x and y axis can be obtained similarly.

Rotation:

Designate the axis of rotation about which the object is to be rotated and the amount of angular rotation .



Axes that are parallel to the coordinate axes are easy to handle.

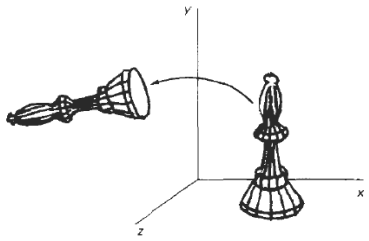
Coordinate axes Rotations:

2D z-axis rotation equations are easily extended to 3D:

$$x' = x\cos\theta - y\sin\theta \qquad y' = x\sin\theta + y\cos\theta \qquad z' = z \quad \ldots\ldots\ldots\ldots(i)$$

3D z-axis rotation equations are expressed in homogenous coordinate form as

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



or,

$$P' = Rz(\theta) . P$$

Cyclic permutation of the coordinate parameters x , y and z are used to get transformation equations for rotations about the other two coordinates

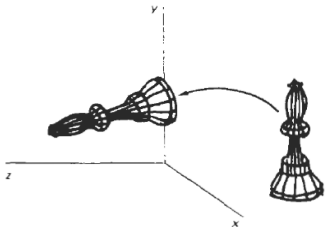
$$x \rightarrow y \rightarrow z \rightarrow \qquad \text{so,}$$

substituting permutations in (i) for an x axis rotation we get,

$$y' = y\cos\theta - z\sin\theta \qquad z' = y\sin\theta + z\cos\theta \qquad x' = x$$

3D x-axis rotation equations in homogenous coordinate form as

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



or,

$$P' = Rx(\theta) . P$$

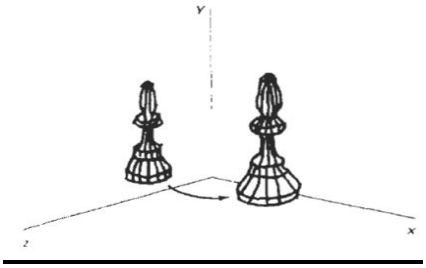
substituting permutations in (i) for a y axis rotation we get,

$$z' = z\cos\theta - x\sin\theta \qquad x' = z\sin\theta + x\cos\theta \qquad y' = y$$

3D y-axis rotation equations are expressed in homogenous coordinate form as

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

or, P' = Ry (θ) . P



Rotation about an axis parallel to one of the coordinate axes :

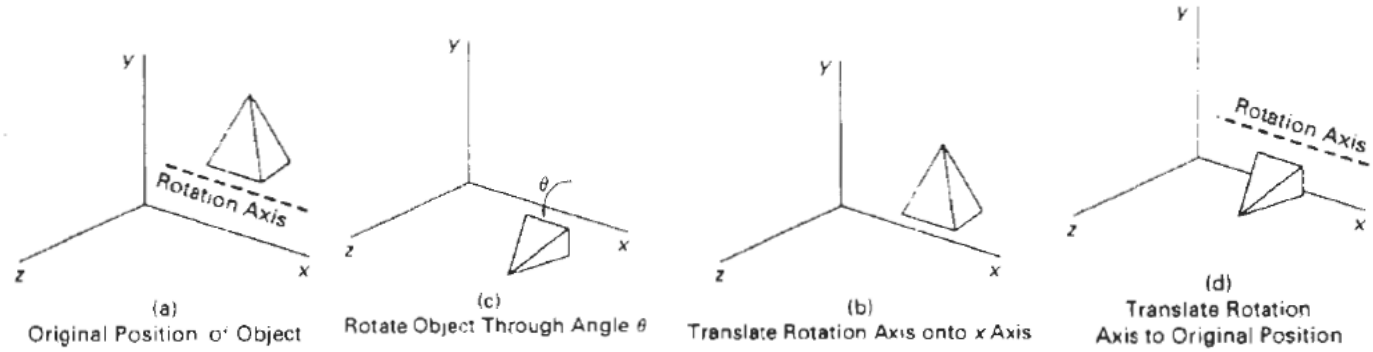
Steps:

i. Translate object so that rotation axis coincides with the parallel coordinate axis.

ii. Perform specified rotation about that axis

iii. Translate object back to it's original position.

ie. $P' = T^{-1} . R_x(\theta) . T . P$



Rotation about an arbitrary axis :

An arbitrary axis in space passing thru point (x_0, y_0, z_0) with direction cosines (c_x, c_y, c_z)

Steps required for rotation about this axis by some angle 0 are:

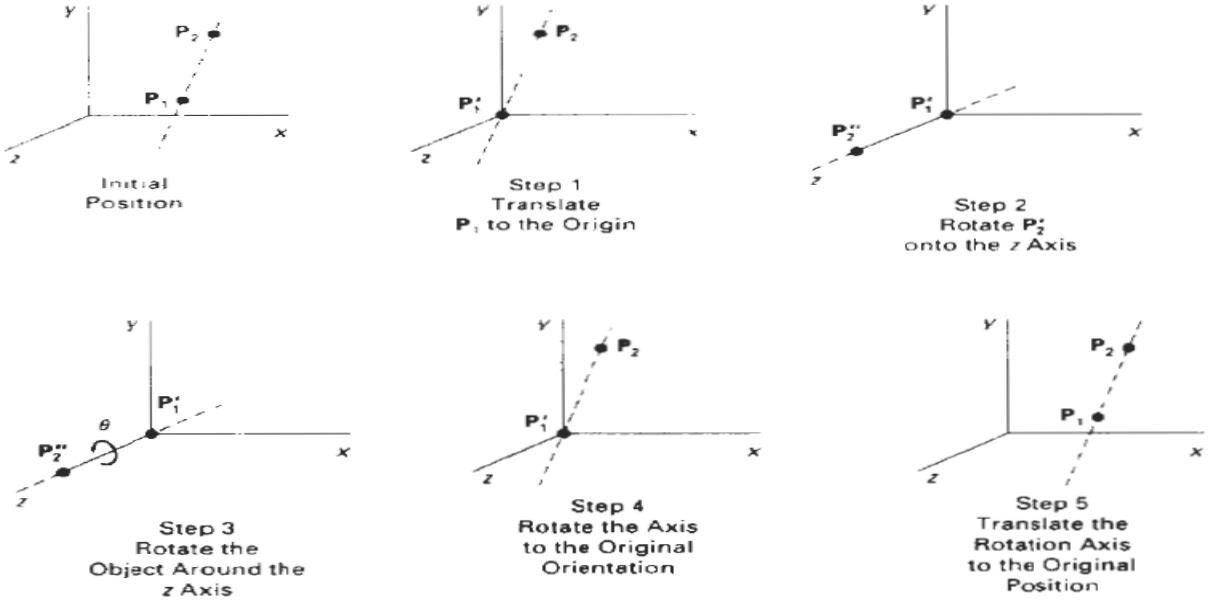
i. Translate so that point(x_0, y_0, z_0) is at the origin of the coordinate system

ii. Perform rotations to make axis of rotation coincident with the z axis

iii.Rotate about the z axis by the angle θ.

iv.Perform the inverse of the combined rotation transformation

v. Perform the inverse of the translation.



Making an arbitrary axis passing thru origin coincident with one of the coordinate axes requires two successive rotations about the other two coordinate axes.

To make arbitrary rotation axis coincident with the rotation angle, about the x axis used to place the arbitrary axis in the xz plane, first project the unit vector along the axis onto the yz plane.

y and z components of the projected vector are c_y and c_z (the direction cosines), so from fig.

$$d = c_y^2 + c_z^2$$

so,

$$\cos = c_z/d \qquad \sin = c_y/d$$

so, transformation matrix for rotation about x axis is:

$$Rx(\ ) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos & -\sin & 0 \\ 0 & \sin & \cos & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_z/d & -c_y/d & 0 \\ 0 & c_y/d & c_z/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

After rotation about the x axis into the xz plane, the z component of the unit vector is d, and x component is c_x (the direction cosines)

Rotation angle about the y axis required to make the arbitrary axis coincident with the z axis is

$$\cos = d \qquad \sin = c_x$$

so, transformation matrix for rotation about y axis is:

$$Ry(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} d & 0 & -c_x & 0 \\ 0 & 1 & 0 & 0 \\ c_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

required translation matrix is

$$T = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

finally,

rotation about the arbitrary axis is given by z axis rotation matrix,

$$R_z(\ ) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

so the transformation matrix for rotation about an arbitrary axis then can be expressed as the composition of these seven individual transformations:

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\theta) \cdot R_y^{-1}(\theta) \cdot R_z(\theta) \cdot R_y(\theta) \cdot R_x(\theta) \cdot T$$

If the values of direction cosines are not known (c_x, c_y, c_z) then they can be obtained knowing a second point on the axis (x_1, y_1, z_1) by normalizing the vector from the first to second point.
Vector along the axis from (x_0, y_0, z_0) to (x_1, y_1, z_1) is

$$[V] = [(x_1 - x_0)\ (y_1 - y_0)\ (z_1 - z_0)]$$

Normalized ,it yields the direction cosines,

$$[c_x\ c_y\ c_z] = \frac{[(x_1 - x_0)\ (y_1 - y_0)\ (z_1 - z_0)]}{[(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2]^{1/2}}$$

<u>Reflection thru an arbitrary plane :</u>
General reflection matrices cause reflection thru x = 0 y = 0 z = 0 coordinate planes respectively.

Often it is necessary to reflect an object thru a plane other than one of these.Which is obtained with the help of a series of transformations (composition).

i. **translate a known point P that lies in the reflection plane to the origin of the coordinate system**
ii. **rotate the normal vector to the reflection plane at the origin until it is coincident with the z axis this makes the reflection plane the z = 0 coordinate plane**

iii. **after applying the above transformation to the object reflect the object thru the z = 0 coordinate plane.**

i.e.

$$RFz = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

iv. **Perform the inverse transformation to those given above to achieve the desired result.**

So the general transformation matrix is:

$$M(\theta) = T^{-1} \cdot R_x^{-1}(\theta) \cdot R_y^{-1}(\theta) \cdot Rflct_z(\theta) \cdot R_y(\theta) \cdot R_x(\theta) \cdot T$$

NOTE:

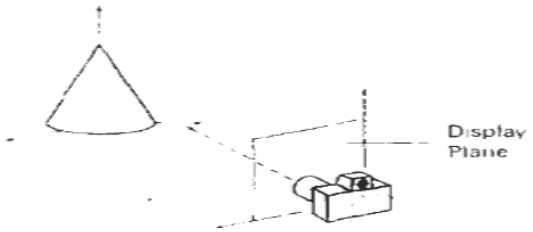
- **See class notes for rotation about arbitrary axis and rotation about arbitrary plane**

3D viewing

In 2D we specify a window on 2D world and a view port on 2D view surface
objects in world are clipped against window and are then transformed into view port for display.

Added complexity caused by

i. **added dimension**
ii. **display devise are only 2D**



Solution is accomplished by introducing projections that transform 3D objects onto 2D plane

In 3D we specify view volume(only those objects within view volume will appear in display on output device others are clipped from display) in world , projection onto projection plane and view port on view surface

So objects in 3D world are clipped against 3D view volume and are then projected
the contents of projection of view volume onto projection plane called window are then transformed (mapped onto) view port for display.

<u>Projections</u>
Transform points in coordinate system of dimension 'n' into points in a coordinate system of dimension less than 'n'.

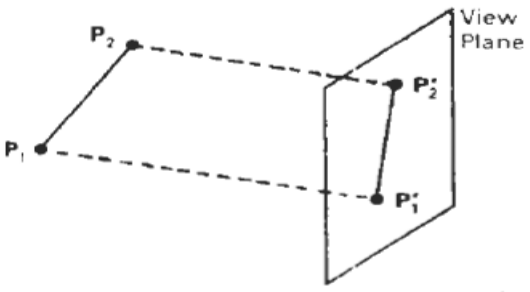
Projection of 3D object is defined by straight projection rays(projectors) emanating from center of projection, passing thru each point of object and intersecting a projection plane to form projection.

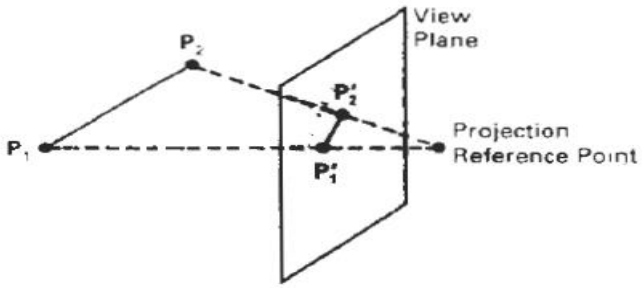
These are planar geometric projection as the projection is onto a plane rather than some curved surface and uses straight rather than curved projectors.

<u>2 types of projections</u>
i. distinction is in relation of center of projection to projection plane

ii. if the distance from one to other
is finite then projection is perspective

iii. if the distance is infinite then projection is parallel.





Perspective

A perspective projection whose center is a point at infinity becomes a parallel projection

Visual effect of perspective projection is similar to that of photographic system and human visual system.

Size of perspective projection of an object varies inversely with distance of that object from the center of projection

Although objects tend to look realistic, is not particularly useful for recording exact shape and measurements of objects.

Perspective projection of any set of parallel lines that are not parallel to projection plane converge to vanishing point

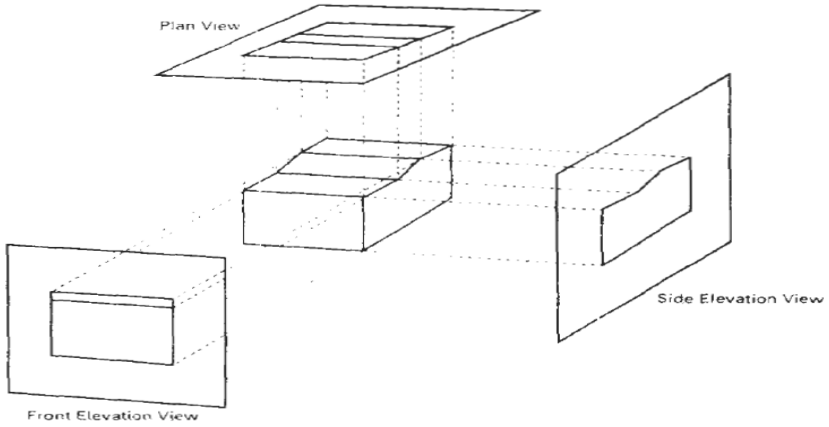
In 3D parallel lines meet only at infinity

If the set of lines parallel to one of three principal axes then vanishing point is called axis vanishing point

eg. If projection plane cuts only z axis and normal to it , only z axis has principle vanishing point as lines parallel to either y or x axes are also parallel to projection plane and has no vanishing points
In fig lines parallel to x,y do not converge only lines parallel to z axis converge

Parallel

Coordinate positions are transformed to the view plane along parallel lines



Preserves relative proportions of objects so that accurate views of various sides of an object are obtained but doesn't give realistic representation of the 3D object.

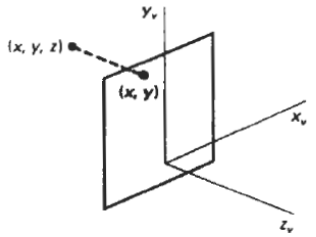
Can be used for exact measurements so parallel lines remain parallel.

a. Orthographic parallel projection

When projection is perpendicular to view plane we have orthographic parallel projection.

Used to produce the front, side and top views of an object. Front, side and rear orthographic projections of an object are called elevations

Top orthographic projection Is called a plan view.



Used in Engineering and architectural drawings.

Views that display more than one face of an object are called axonometric orthographic projections. Most commonly used axonometric projection is the isometric projection.

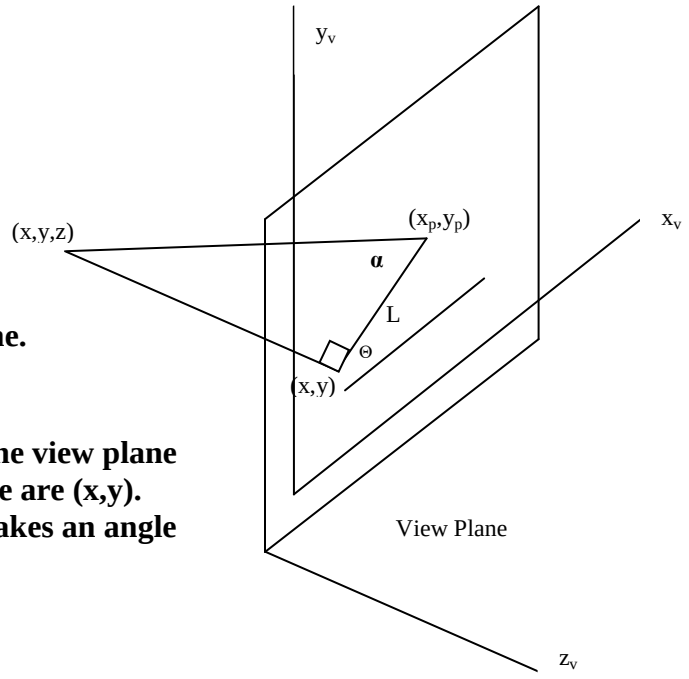
Transformation equations

If view plane is placed at position z_{vp} along z_v axis then any point (x,y,z) is transformed to projection as

$$x_p = x, \quad y_p = y$$

z value is preserved for depth information needed (visible surface detection).

b. Oblique parallel projection



Obtained by projecting points along parallel lines that are not perpendicular to projection plane.

Often specified with two angles Θ and α
Point (x ,y ,z) is projected to position(x_p , y_p) on the view plane
Orthographic projection coordinated on the plane are (x,y).
Oblique projection line from (x,y,z) to (x_p , y_p) makes an angle with then line on the projection plane that joins (x_p , y_p) and (x , y)

This line of length L is at an angle Θ with the horizontal direction in the projection plane.
Expressing projection coordinates in terms of x,y,L and Θ as

$$x_p = x + L\cos\Theta$$
$$y_p = y + L\sin\Theta$$

L depends on the angle α and z coordinate of point to be projected

$$\tan\alpha = z/L$$

thus,

$$L = z/\tan\alpha$$

$$= zL_1 \qquad L_1 \text{ is the inverse of } \tan\alpha$$

so the oblique projection equations are

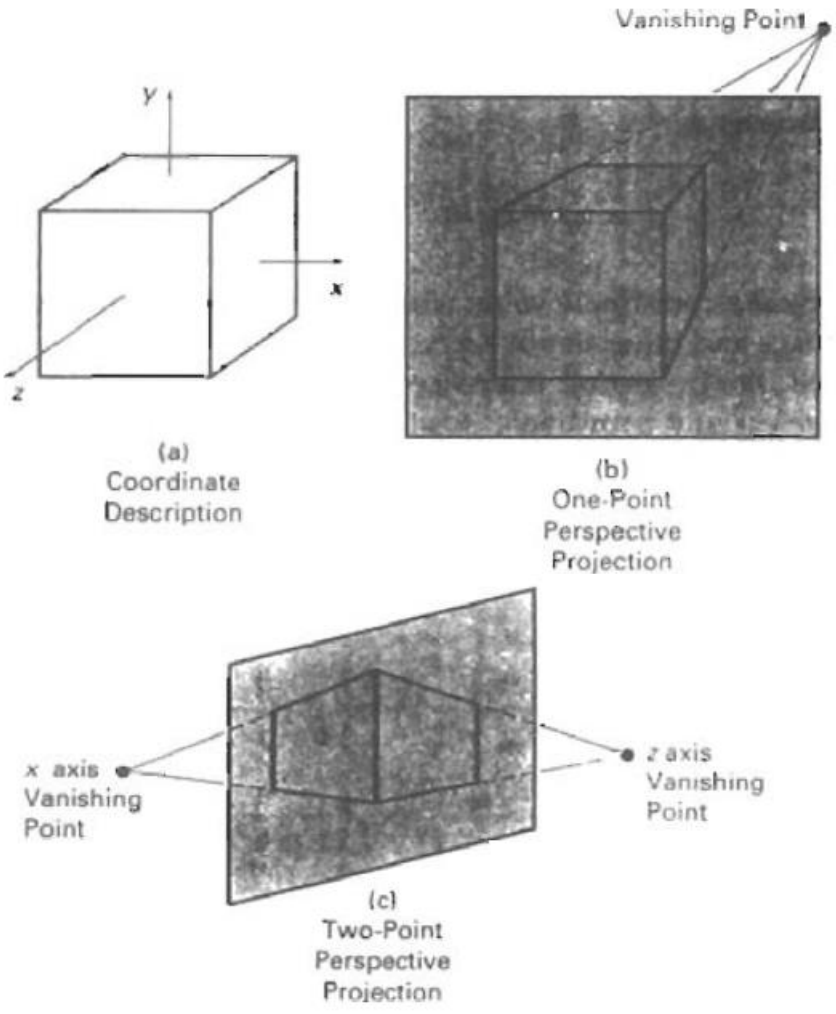
$$x_p = x + z(L_1\cos\Theta)$$

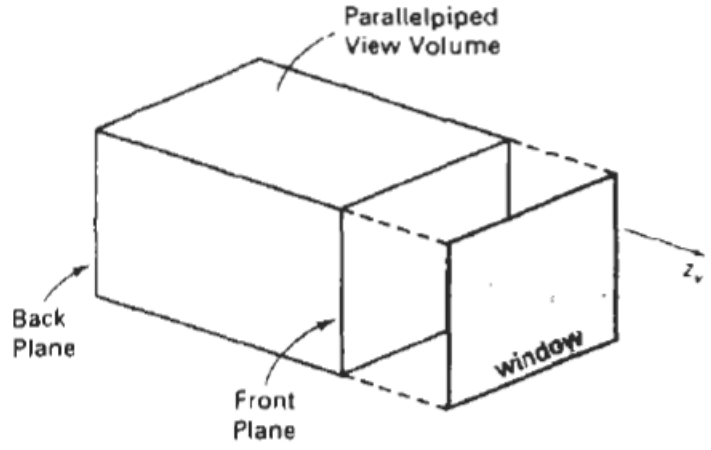
$$y_p = y + z(L_1\sin\Theta)$$

the transformation matrix for producing any parallel projection onto the $x_v\ y_v$ plane can be written as

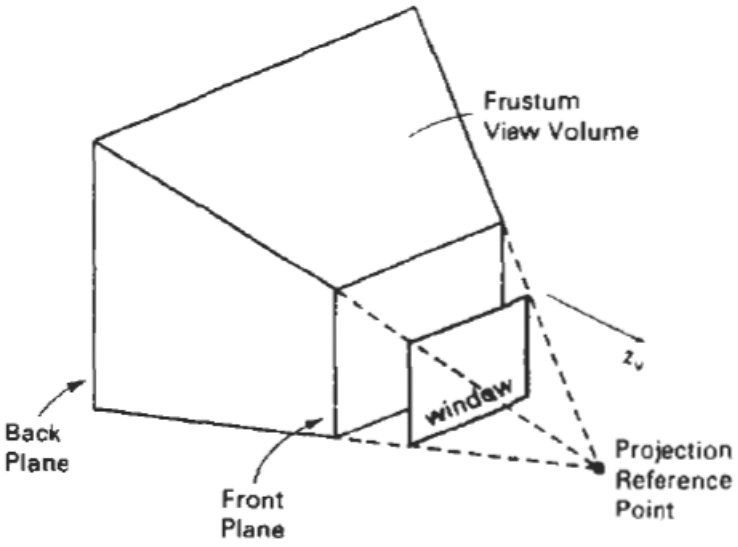
$$M_{parallel} = \begin{pmatrix} 1 & 0 & L_1\cos\Theta & 0 \\ 0 & 1 & L_1\sin\Theta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthographic projection is obtained when $L_1 = 0$ (occurs at projection angle α of 90*)
Oblique projection is obtained with non zero values for L_1.





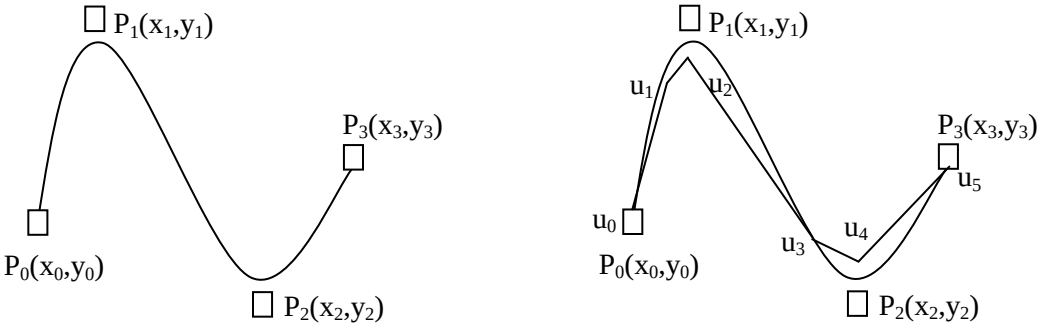
Parallel Projection



Perspective Projection

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

Suppose P_0, P_1, P_2, P_3 are four control points

Number of segments in a line segment : nSeg

i = 0 to nSeg

u = i/nSeg [0,1] 0<= u <= 1

u0,u1 ……..u3

$x(u) = \sum_{j=0}^{n} x_j \, BEZ_{j,n}(u)$ n : number of control points

$x(u) = x_0 \, BEZ_{0,3}(u) + x_1 \, BEZ_{1,3}(u) + x_2 \, BEZ_{2,3}(u) + x_3 \, BEZ_{3,3}(u)$

similarly

$y(u) = \sum_{j=0}^{n} y_j \, BEZ_{j,n}(u)$ n : number of control points

$y(u) = y_0 \, BEZ_{0,3}(u) + y_1 \, BEZ_{1,3}(u) + y_2 \, BEZ_{2,3}(u) + y_3 \, BEZ_{3,3}(u)$

The Bezier blending function BEZj,n (u) is defined as,

$$BEZ_{j,n}(u) = \frac{n!}{j!\,(n-j)!} u^j (1-u)^{n-j}$$

$$BEZ_{j,n}(u) = C_{(n,j)} \, u^j (1-u)^{n-j}$$

Where $C_{(n,j)}$ is the Binomial Coefficient

$$C_{(n,j)} = \frac{n!}{j!\,(n-j)!}$$

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

Now

$Q(u) = P_0\, BEZ_{0,3}(u) + P_1\, BEZ_{1,3}(u) + P_2\, BEZ_{2,3}(u) + P_3\, BEZ_{3,3}(u)$

Four blending functions must be found based on Bernstein Polynomials

$$BEZ_{0,3}(u) = \frac{3!}{0!\ 3!} u^0 (1-u)^3 = (1-u)^3 \qquad BEZ_{1,3}(u) = \frac{3!}{1!\ 2!} u^1 (1-u)^2 = 3u(1-u)^2$$

$$BEZ_{2,3}(u) = \frac{3!}{2!\ 1!} u^2 (1-u) = 3u^2(1-u) \qquad BEZ_{3,3}(u) = \frac{3!}{3!\ 0!} u^3 (1-u)^0 = u^3$$

Normalizing properties apply to blending function s that means thy 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 $Q(u) = P_3$

in Matrix Form

$$Q(u) = \begin{bmatrix} (1-u)^3 & 3u(1-u)^2 & 3u^2(1-u) & u^3 \end{bmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

or

$$Q(u) = \begin{bmatrix} (1-3u+3u^2-u^3) & (3u-6u^2+3u^3) & (3u^2-3u^3) & u^3 \end{bmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

or

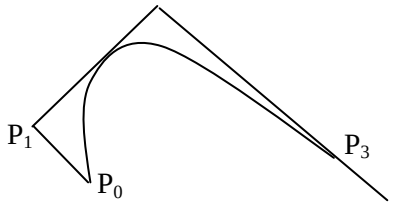
$$Q(u) = \begin{bmatrix} u^3 & u^2 & u^1 & 1 \end{bmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$$

Properties of a Bezier Curve

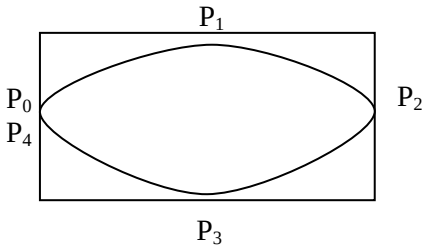
1. Bezier curve lies in the convex hull of the control points which ensure that the curve smoothly follows the control

P_2

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
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

Two sets of orthogonal Bezier curves can be used to design an object surface be specifying by an input mesh of control points. The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions:

Bezier surfaces are defined by simple generalization of the curve formulation. Here, tensor product approach is used with two directions of parameterization 'u' and 'v'.

Any point on the surface can be located to given values of parametric pair by

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} P_{j,k} BEZ_{j,m}(u) BEZ_{k,n}(v) \qquad 0 \le u,v \le 1$$

As in the case of Bezier curves the $P_{j,k}$ define the control vertices and the $BEZ_{j,m}(u)$ and $B_{k,n`}(v)$ are the Bernstein blending functions in the u and v directions

The Bézier functions specify the weighting of a particular knot. They are the Bernstein coefficients. The definition of the Bézier functions is

$$BEZ_{j,m}(u) = C(m,j)\, u^{j} (1-u)^{m-j}$$
$$BEZ_{k,n}(v) = C(n,k)\, v^{k} (1-v)^{n-k}$$

where C(m,j) represents the binomial coefficients.

$$C(m,j) = \frac{m!}{j!\,(m-j)!}$$

where C(n,k) represents the binomial coefficients.

$$C(n,k) = \frac{n!}{k!\,(n-k)!}$$

When t = 0, the function is one for j = 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.

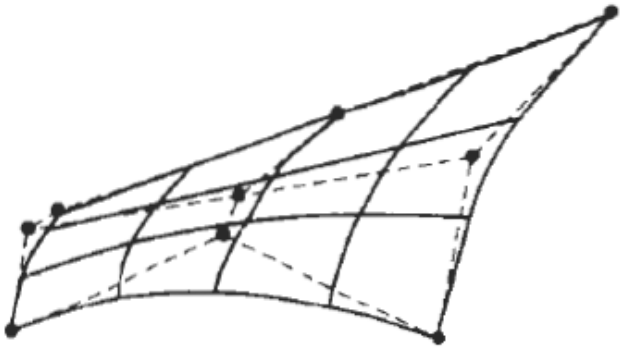
Bézier surfaces are useful for interactive design and were first applied to car body design.

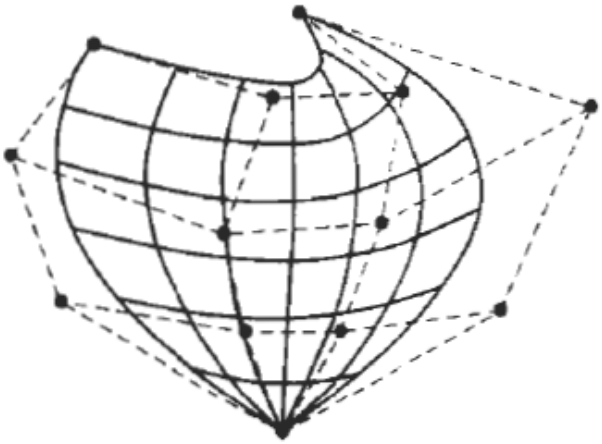
The degree of blending functions does not have to be the same in two parametric directions it could be cubic in 'u' and quadratic in 'v'

The properties of Bezier surfaces are controlled by the blending functions

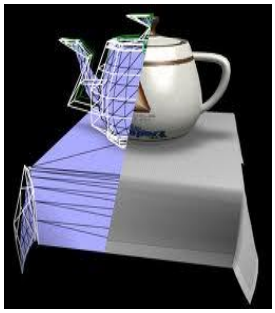
- The surface takes the general shape of the control points
- The surface is contained within the convex hull of the control points
- The corners of the surface and the corner control vertices are coincident

Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points.
The parametric vector function for the Bezier surface is formed as the Cartesian product of Bezier blending functions with $P_{j,k}$ specifying the location of the (m + 1) by (n + 1) control points.





Bezier surfaces constructed for m = 3 n = 3 and m = 4 n = 4
Dashed lines connect the control points



FRACTAL-GEOMETRY METHODS

Natural objects, such as mountains and clouds, don't have smooth surface or regular shapes instead they have fragmented features, and Euclidean methods do not realistically model these objects.

Natural objects can be realistically described with fractal-geometry methods, where procedures rather than equations are used to model objects.

In computer graphics, fractal methods are used to generate displays of natural objects and visualizations of various mathematical and physical systems.

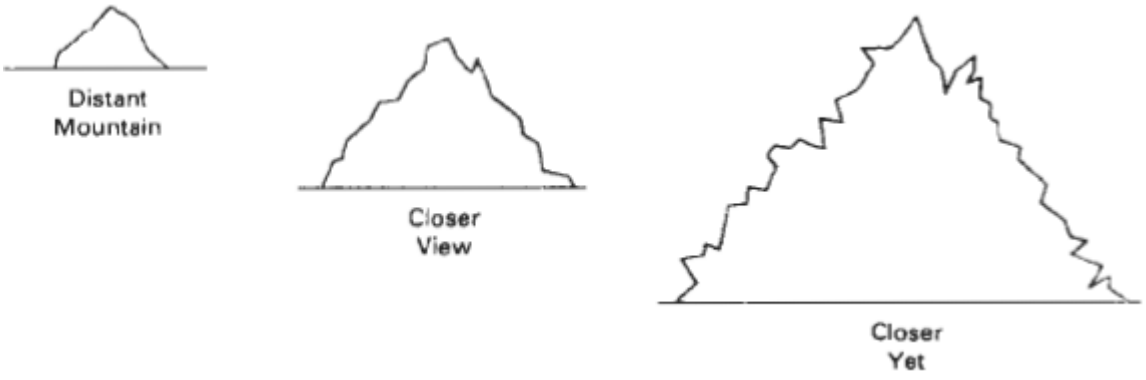
A fractal object has two basic characteristics:

i. Infinite detail at every point
ii. Certain self-similarity between the object parts and the overall features of the object

We describe a fractal object with a procedure that specifies a repeated operation for producing the detail in the object subparts. Natural objects are represented with procedures that theoretically repeat an infinite number of times. Graphics displays of natural objects are, of course, generated with a finite number of steps.

If we zoom in on a continuous Euclidean shape, no matter how complicated, we can eventually get the zoomed-in view to smooth out.

But if we zoom in on a fractal object, we continue to see as much detail in the magnification as we did in the original view.



Zooming in on a graphics display of a fractal object is obtained by selecting a smaller window and **repeating the fractal procedures** to generate the detail in the new window.

A consequence of the infinite detail of a fractal object is that it has no definite size. As we consider more and more detail, the size of an object tends to infinity, but the coordinate extents of the object remain bound within a finite region of space.

The amount of variation in the object detail with a number called the ***fractal dimension.***

In graphics applications, fractal representations are used to model terrain, clouds, water, trees and other plants, feathers, fur, and various surface textures, and just to make pretty patterns.

Fractal-Generation Procedures

A fractal object is generated by **repeatedly applying a specified transformation function** to points within a region of space.

If $\mathbf{P_0}$ = (x0, y0, z0) is a selected initial point, each iteration of a transformation function F generates successive levels of detail with the calculations

In general, the transformation function can be applied to a specified point set, or we could apply the transformation function to an initial **set** of primitives, such as straight lines, curves, color areas, surfaces, and solid objects.

Also, we can use either deterministic or random generation procedures at each iteration. The transformation function may be defined in terms of geometric transformations (scaling, translation, rotation), or it can be set up with nonlinear coordinate transformations and decision parameters.

Although fractal objects, by definition, contain infinite detail, we apply the transformation function a finite number of times. Therefore, the objects we display actually have finite dimensions.

A procedural representation approaches a "true" fractal as the number of transformations is increased to produce more and more detail.

The **amount of detail** included in the final graphical display of an object **depends on the number of iterations performed** and the resolution of the display system

We cannot display detail variations that are smaller than the size of a pixel.

To **see more** of the object detail, we **zoom in on selected sections** and **repeat the transformation function** iterations.

Classification of Fractals

i. Self-similar fractals

Self-similar fractals have parts that are scaled-down versions of the entire object.

Starting with an initial shape, the object subparts are constructed by apply a scaling parameter ‘s’ to the overall shape.

The same scaling factors can be used for all subparts, or different scaling factors can be used for different scaled-down parts of the object.

If random variations are applied to the scaled-down subparts, the fractal is said to be statistically self-similar. The parts then have the same statistical properties.

Statistically self-similar fractals are commonly used to model trees, shrubs, and other plants.

ii. Self-affine fractals

Self-affine fractals have parts that are formed with different scaling parameters, $\mathbf{s_x, s_y, s_z}$ in different coordinate directions.

Random variations can also be used to obtain statistically self-affine fractals.

Terrain, water, and clouds are typically modeled with statistically self-affine fractal construction methods.

iii. Invariant fractal

Invariant fractal sets are formed with nonlinear transformations.

This class of fractals includes self-squaring fractals, such as the Mandelbrot set, which are formed with squaring functions in complex space; and self-inverse fractals, formed with inversion procedures.

Fractal Dimension

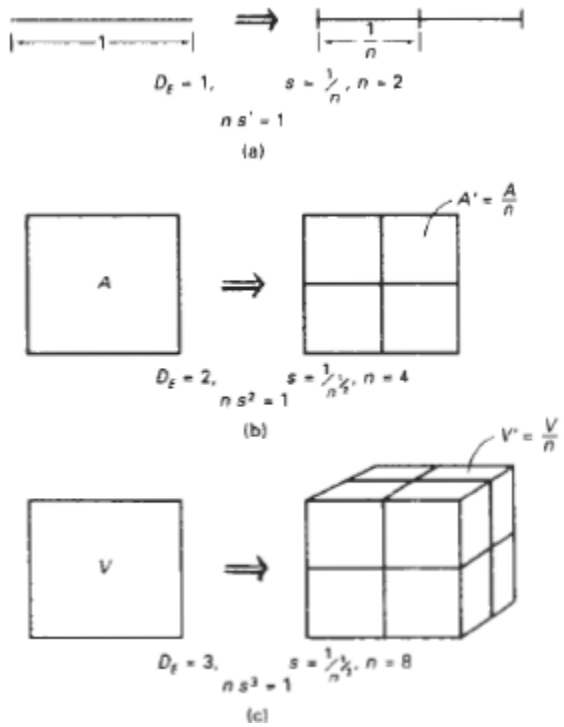
The detail variation in a fractal object can be described with a number D, called the **fractal dimension**, which is a measure of the roughness, or fragmentation, of the object.

More jagged-looking objects have larger fractal dimensions.

Iterative procedures can be set up to generate fractal objects using a given value for the fractal dimension **D.**

An expression for the fractal dimension of a self-similar fractal, constructed with a single scalar factor **s**, is obtained by analogy with the subdivision of a Euclidean object.

Suppose an object is composed of clay or elastic. If it is deformed into a line then its topological dimension D_t is 1, if it is deformed into a plane or a disk then the topological Dimension is 2 and if it is deformed into a ball or a cube then its topological dimension is 3



The 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 can be shown

If take a line segment having length L and divide it into n pieces each piece having length '*l*'

The scaling factor s = 1/n

If it is broken into two pieces, $s^1 = 1/2$, the unit line segment is divided into two equal-length subparts.

Similarly, the square is divided into four equal-area subparts, $s^2 = 1/4$

The cube is divided into eight equal-volume subparts $s^3 = 1/8$

For each of these objects, the relationship between the number of subparts and the scaling factor is $n \cdot s^D = 1$. In analogy with Euclidean objects, the fractal dimension D for self-similar objects can be obtained from

$$n \cdot s^D = 1.$$

Solving this expression for D, the fractal similarity dimension, we have

$$D = (\log n) / (\log (1/s))$$

The fractal dimension gives the measure of the roughness or fragmentation of objects and is always greater than the corresponding topological dimension

For a self-similar fractal constructed with different scaling factors for the different parts, the fractal similarity dimension is obtained from the implicit relationship where $\mathbf{s_k}$ is the scaling factor for subpart number k.

Geometric Construction of Deterministic Self-Similar Fractals

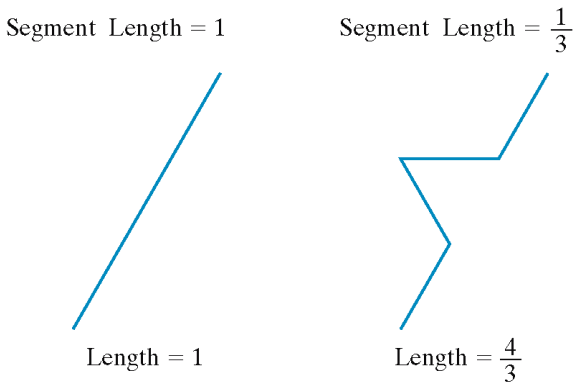
To geometrically construct a deterministic (nonrandom) self-similar fractal, we start with a given geometric shape, called the ***initiator.*** Subparts of the initiator are then replaced with a pattern, called the generator.

Koch Curve

1. Begin with a line segment

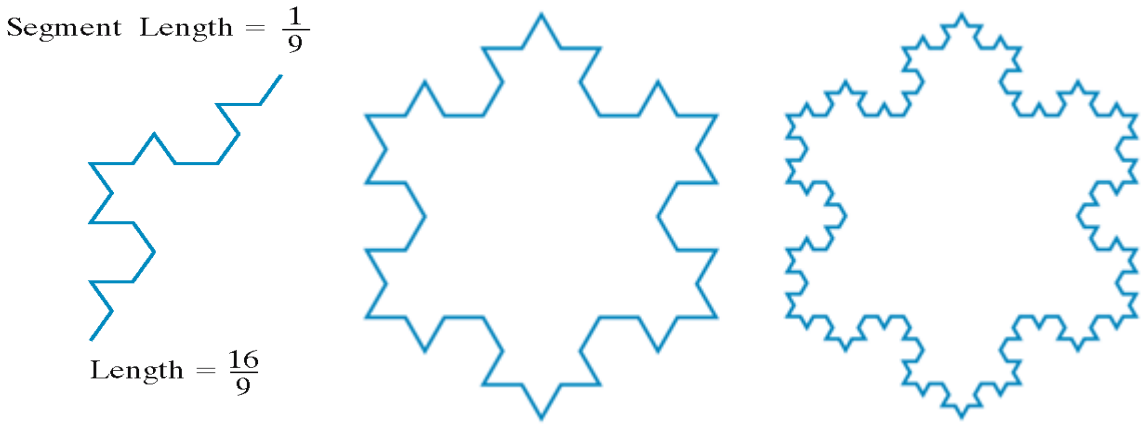
2. Divide it into thirds i.e. scaling factor = 1/3 and replace the center third by the two adjacent sides of an equilateral triangle

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



5. Repeat the process for each of the four segments

6. The cure 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

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

1. Sub-divide a square into 4 quadrants and draw the curve which connects the center points of each

2. Further subdivide each of the quadrants and connect the centers of each of these finer divisions before moving to the next major quadrant

3. The third approximation subdivides again it again connects the centers of the finest level before stepping to the next level of detail

The above process is indefinitely continued depending upon the degree of roughness of the curve generated

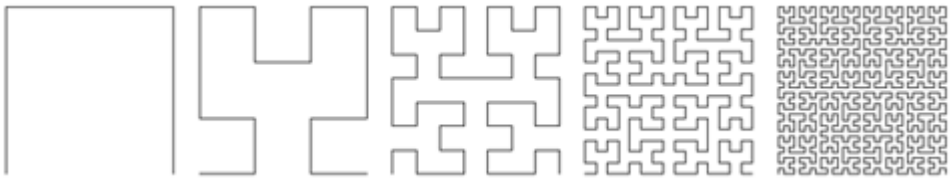
- The curve never crosses itself
- There is no limit to the subdivision
- The curve fills the square
- With each subdivision the length increases by a factor of 4 and since there is no limit to subdivision there is no limit to the length
- The curve constructed is topologically equivalent to the line $D_t = 1$ but it is so twisted and folded that it exactly fills up a square

- 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 the Topological Dimension is 1



Geometric Construction of Statistically Self-Similar Fractals

One way to introduce some randomness into the geometric construction of a self-similar fractal is to choose a generator randomly at each step from a set of predefined shapes. Another way to generate random self-similar objects is to compute coordinate displacements randomly.

A random snowflake pattern can be created by selecting a random, midpoint displacement distance at each step.

A modified "snowflake" pattern using random midpoint displacement.

Affine Fractal-Construction Methods

Highly realistic representations for terrain and other natural objects can be obtained using affine fractal methods that model object features as ***fractional Brownian motion.***

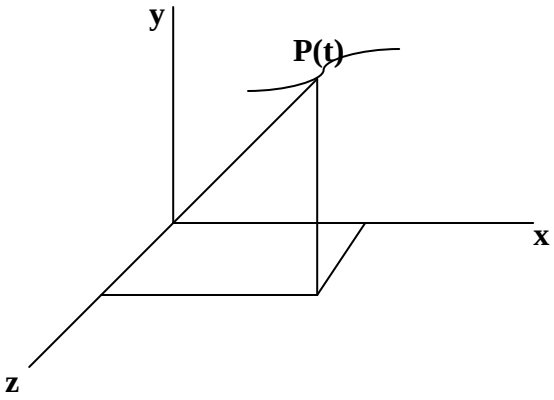
This is an extension of standard Brownian motion, a form of "random walk**",** that **describes** the erratic, zigzag movement of particles in a gas or otter fluid.

Starting from a given position, we generate a straight-line segment in a random direction and with a random length. We then move to the endpoint of the first line segment

Parametric Cubic Curve

A parametric cubic curve is defined as $P(t) = \sum_{i=0}^{3} a_i t^i$ **0<= t <=1 ------ (i)**

Where, P(*t*) is a point on the curve



Expanding equation (i) yields

$P(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ **----------(ii)**

This equation is separated into three components of P(t)

$x(t) = a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x}$

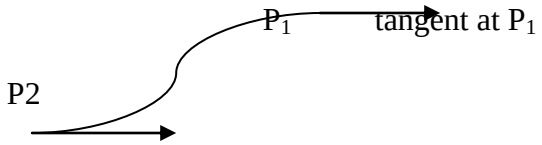
$y(t) = a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_{0y}$

$z(t) = a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z}$ **----------(iii)**

To be able to solve (iii) the twelve unknown coefficients a_{ij} (algebraic coefficients) must be specified

From the known end point coordinates of each segment, six of the twelve needed equations are obtained. The other six are found by using tangent vectors at the two ends of each segment

The direction of the tangent vectors establishes the slopes(direction cosines) of the curve at the end points



This procedure for defining a cubic curve using end points and tangent vector is one form of *hermite* interpolation

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 relation ship between two end point vectors and the algebraic coefficients are found

$P(0) = a_0$ $P(1) = a_3 + a_2 + a_1 + a_0$

To find the tangent vectors equation ii must be differentiated with respect to t

$P'(t) = 3a_3t^2 + 2a_2t + a_1$

The tangent vectors at the two end points are found by substituting t = 0 and t = 1 in this equation

$P'(0) = a_1$ $P'(1) = 3a_3 + 2a_2 + a_1$

The algebraic coefficients 'a_i' in equation (ii) can now be written explicitly in terms of boundary conditions – endpoints and tangent vectors are

$a_0 = P(0)$ $a_1 = P'(0)$

$a_2 = -3P(0) + 3P(1) - 2P'(0) - P'(1)$ $a3 = 2P(0) - 2P(1) + P'(0) + P'(1)$

substituting these values of 'a_i' in equation (ii) and rearranging the terms yields

$P(t) = (2t^3 - 3t^2 + 1)P(0) + (-2t^3 + 3t^2)P(1) + (t^3 - 2t^2 + t)P'(0) + (t^3 - t^2)P'(1)$

The values of P(0), P(1), P'(0), P'(1) are called *geometric coefficients* and represent the known vector quantities in the above equation

The polynomial coefficients of these vector quantities are commonly known as *blending functions*

By varying parameter t in these blending function from 0 to 1 several points on curve segments can be found

Perspective Projection

Method for generating a view of a three-dimensional scene by **projecting points to the display plane along converging paths**.

This causes **objects farther from the viewing position to be displayed smaller** than objects of the same size that are nearer to the viewing position.

In a perspective projection, parallel lines in a scene that are not parallel to the display plane are **projected into converging lines.**

Scenes displayed using perspective projections **appear more realistic**, since this is the way that eyes and camera lens form images.

In the perspective projection view, parallel lines appear to converge to a distant point in the background, and distant objects appear smaller than objects closer to the viewing position.

To obtain a perspective projection of a three-dimensional object, transform points along projection lines that meet at the **projection reference point**.

Suppose the perspective reference point is set at position z_{prp} along the z_v axis, and the view plane is placed at z_{vp} .

Equations describing coordinate positions along this perspective projection line in parametric form as

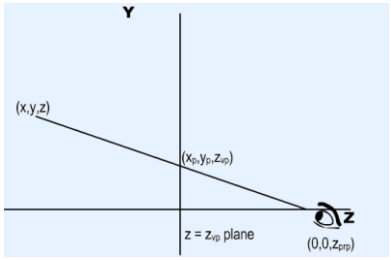
$$x' = x - xu$$

$$y' = y - yu$$

$$z' = z - (z - z_{prp})u$$

Parameter 'u' takes values from 0 to 1, and coordinate position (x', y', z') represents any point along the projection line.

When u = 0, we are at position P = (x,y, z)



At the other end of the line, $u = 1$ and we have the projection reference point coordinates $(0,0, z_{prp})$ On the view plane, $z' = z_{vp}$, and we can solve the z' equation for parameter u at this position along the projection line:

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

Substituting this value of u into the equations for x' and y', we obtain the perspective transformation equations

$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{d_p}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = y\left(\frac{d_p}{z_{prp} - z}\right)$$

where, $d_p = z_{prp} - z_{vp}$ is the distance of the view plane from the projection reference point.

Using a three-dimensional homogeneous-coordinate representation, we can write the perspective projection transformation in matrix form as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -z_{vp}/d_p & z_{vp}(z_{prp}/d_p) \\ 0 & 0 & -1/d_p & z_{prp}/d_p \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In this representation, the homogeneous factor is

$$h = \frac{z_{prp} - z}{d_p}$$

and the projection coordinates on the view plane are calculated from the homogeneous coordinates as

$$x_p = x_h/h, \qquad y_p = y_h/h$$

where the original z-coordinate value would be retained in projection coordinates for visible-surface and other depth processing.

In general, the projection reference point does not have to be along the z_v axis.

There are a number of special cases for the perspective transformation equations.

If the view plane is taken to be the uv plane, then $z_{vp} = 0$ and the projection coordinates are

$$x_p = x\left(\frac{z_{prp}}{z_{prp} - z}\right) = x\left(\frac{1}{1 - z/z_{prp}}\right)$$

$$y_p = y\left(\frac{z_{prp}}{z_{prp} - z}\right) = y\left(\frac{1}{1 - z/z_{prp}}\right)$$

And, in **some** graphics packages, the projection reference point is always taken to be at the viewing-coordinate origin. In this case, $z_{prp} = 0$ and the projection coordinates on the viewing plane are

$$x_p = x\left(\frac{z_{vp}}{z}\right) = x\left(\frac{1}{z/z_{vp}}\right)$$

$$y_p = y\left(\frac{z_{vp}}{z}\right) = y\left(\frac{1}{z/z_{vp}}\right)$$

When a three-dimensional object is projected onto a view plane using perspective transformation equations, any set of parallel lines in the object that are not parallel to the plane are projected into converging lines.

Parallel Lines that are parallel to the view plane will be projected as parallel lines.

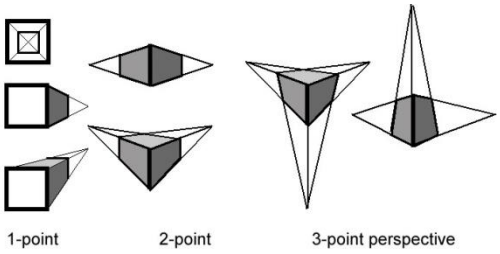
The point at which a set of projected parallel lines appears to converge is called a vanishing point.

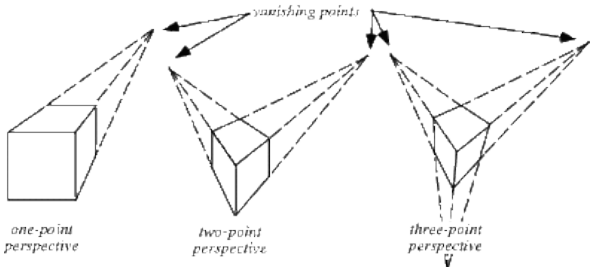
Each such set of projected parallel lines will have a separate vanishing point; and in general, a scene can have any number of vanishing points, depending on how many sets of parallel lines there are in the scene.

The vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as a principal vanishing point.

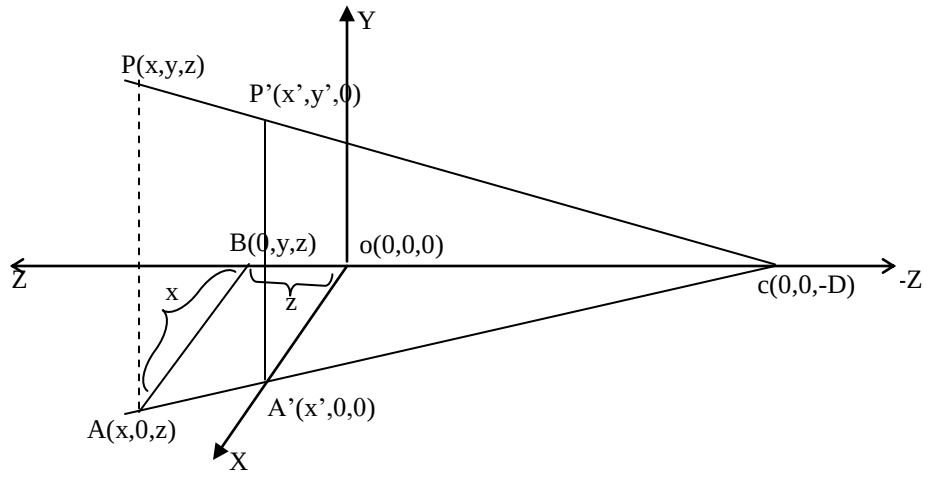
The number of principal vanishing points (one, two, or three) are controlled with the orientation of the projection plane, and perspective projections are accordingly classified as one-point, two-point, or **three-point** projections.

The number of principal vanishing points in a projection is determined by the number of principal axes intersecting the view plane.





Perspective Projection (Standard)



Here center of Projection is c (0,0,-D) along the direction of Z axis so the reference point is taken of world coordinate space W_c and the normal vector N is aligned with the y axis.

So now the view plane vp is the xy plane and center of projection is c (0,0,-D) now from similar triangles ABC and A'OC

$$\frac{x}{x'} = \frac{z+D}{D} = \frac{AC}{A'C}$$

$$\text{or } \frac{xD}{z+D} = x' \quad \text{or} \quad x' = \frac{Dx}{z+D}$$

similarly from triangles APC and A'P'C

$$\frac{y}{y'} = \frac{z+D}{D} = \frac{AC}{A'C}$$

$$\text{or } y' = \frac{yD}{z+D}$$

and z' =0

now in homogenous coordinates

$$\text{Perk} = \begin{pmatrix} X' \\ Y' \\ Z' \\ 1 \end{pmatrix} = \frac{1}{z+D} \begin{pmatrix} Dx \\ Dy \\ 0 \\ z+D \end{pmatrix} = \frac{1}{z+D} \begin{pmatrix} D & 0 & 0 & 0 \\ 0 & D & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & D \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

A unit cube is projected into xy plane . Draw the projected image using standard perspective transformation where center of projection is (0,0,-20)

Here,

Center of projection = (0,0,-20) i.e. d = 20

Persp =

$$\begin{pmatrix} 20 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 20 \end{pmatrix}$$

The cube represented in Homogenous coordinate is

V(A,B,C,D,E,F,G,H) =

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

V' = Persp * V

$$\begin{pmatrix} 20 & 0 & 0 & 0 \\ 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 20 \end{pmatrix} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

= 1/z+D

$$\begin{pmatrix} 0 & 20 & 20 & 0 & 0 & 0 & 20 & 20 \\ 0 & 0 & 20 & 20 & 20 & 0 & 0 & 20 \\ 0 & 0 & 0 & 0 & 20 & 20 & 20 & 20 \\ 20 & 20 & 20 & 20 & 21 & 21 & 21 & 21 \end{pmatrix}$$

Hence,

1/(z+D) = 1/ (0 +20) == 1/ 20

A' =(0,0,0) B' =(1,0,0) C' =(1,1,0) D' =(0,1,0)

1/(z+D) = 1/ (1 +20) = 1/ 21

E' =(0,20/21,0) F' =(0,0,0) G' =(20/21,0,0) H' =(20/21, 20/21,0)

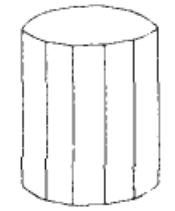
3D Object Representation (Polygon Surfaces: Planar)

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.

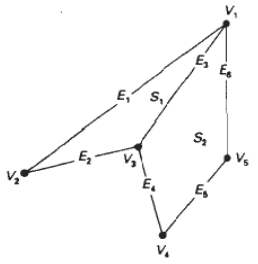


Polygon Tables

Polygon data tables can be organized into two groups: geometrical and attribute tables
Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of polygon surfaces
Attribute information for an object includes parameters specifying the degree of transparency of object and its surface reflectivity and texture characteristics.

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

$$Ax + By + Cz + D = 0 \quad \text{.....(i)}$$

Where, (x,y,z) is any point on the plane- coefficients ABCD are constants describing the spatial properties of the 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,

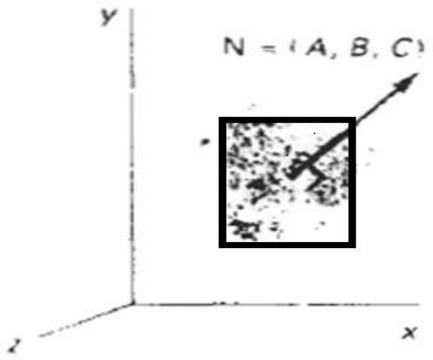
$$\frac{A}{D} x_k + \frac{B}{D} y_k + \frac{C}{D} z_k = -1 \quad \text{.....(ii)} \qquad k = 1,2,3$$

Using Cramers rule,

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \qquad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Expanding Determinants,

$$A = y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2)$$
$$B = z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2)$$
$$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2)$$
$$D = -x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_3 z_1 - y_1 z_3) - x_3 (y_1 z_2 - y_2 z_1)$$



The vector N, normal to the surface of a plane described by equation Ax + By + Cz + D =0 has Cartesian Components (A,B,C)

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

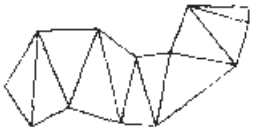
$$Ax + By + Cz + D$$

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

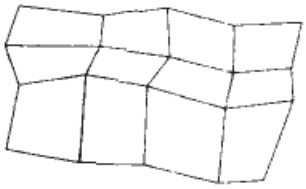
<u>Polygon Meshes</u>

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 functions the quadrilateral mesh that generates a mesh of (n-1)(m-1) quadrilaterals,
given the coordinates for an n by m array of vertices



When polygons are specified with more than 3 vertices, it is possible that the vertices may not all lie in one plane.

This can be due to numerical error or errors in selecting coordinate positions for the vertices

Remedies:

i. **Simply divide the polygons into triangles**
ii. **Approximate the plane parameters ABC and project in same plane (i.e. approximate A to yz plane, B to xz plane, C to xy plane etc)**

High quality graphics systems typically model objects with polygon meshes and set up a database of geometric and attribute information to facilitate processing of the polygon facets.

Curved Surfaces and Lines

Display of 3D curved lines and surfaces can be generated from an input set of mathematical functions (spheres, ellipsoid etc) defining the objects or from a set of user specified data points.

Spline representations are examples of generating curves and surfaces. These are commonly used to design new object shapes , to digitize drawings and to describe animation paths.

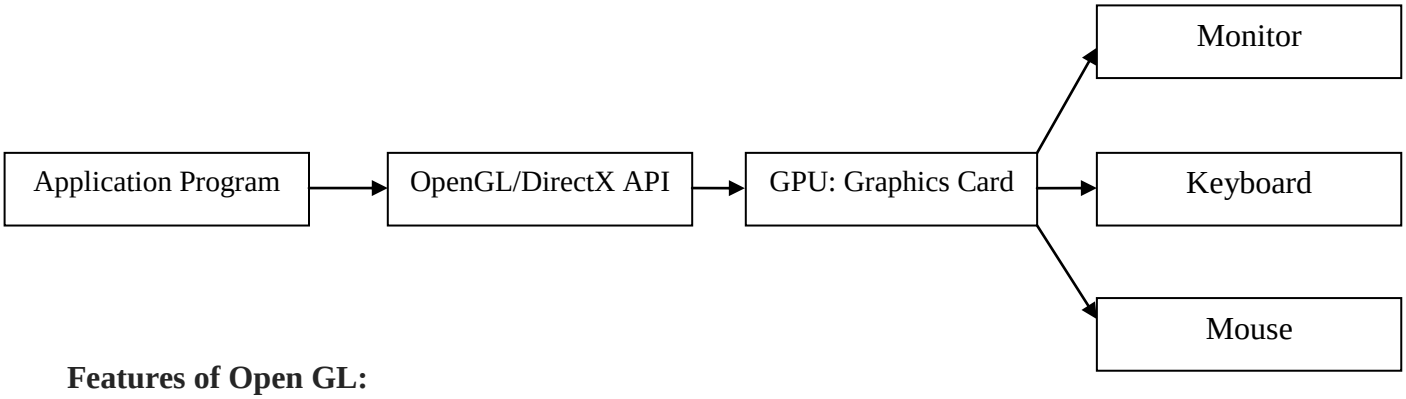
Curve fitting methods are also used to display graphs of data values by fitting specified curve functions to the discrete data set, using regression techniques such as the least square methods.

OPENGL

OpenGL is a **software interface** to graphics hardware that consists of **250 distinct commands** (200 in core OpenGL and 50 in OpenGL Utility Library) to produce interactive 3D applications It was designed as **hardware independent interface** to be implemented on different hardware platforms for building models that are built up from a small set of geometric primitives – points, lines, polygons

It is 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.



Features of Open GL:

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 on 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 surface 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

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 Open GL architecture Review board an independent consortium guides the Open GL specification Open GL is the only true open vendor-neutral, multiplatform graphics standard with a broad industry support

Stability: Updating to Open GL 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.

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: Open GL 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 b e 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.

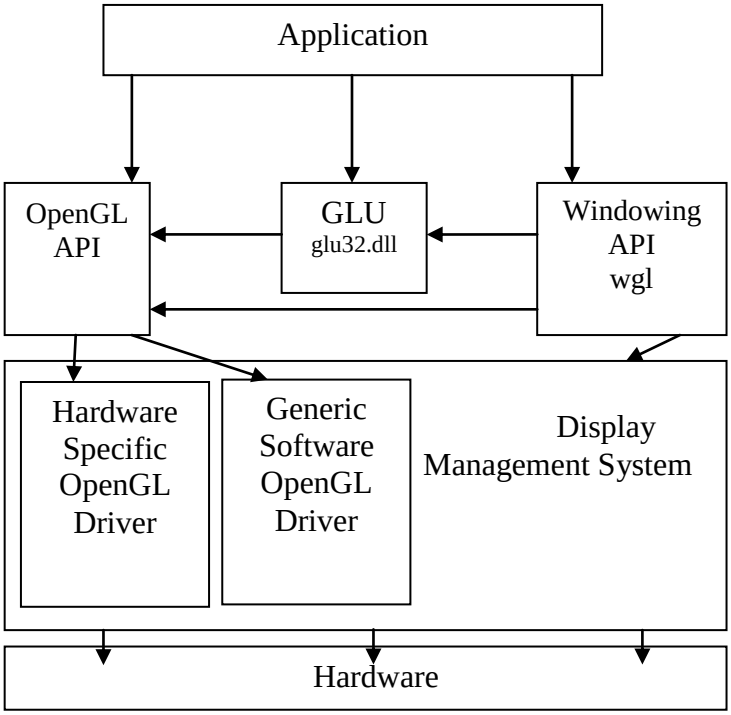
OpenGL has been through a number of revisions which have predominantly been incremental additions where extensions to the core API have gradually been incorporated into the main body of the API. For example OpenGL 1.1 added the glBindTexture extension to the core API.

OpenGL 2.0 incorporates the significant 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.

General OpenGL Architecture



Block Diagram of Typical Application / OpenGL / OS interactions.

OpenGL is the definitive cross-platform 3D library.

- The **Application** module is developed by the programmer
- **Windowing API** functions specifically developed to support OpenGL are used to set up, shut down, and handles **event** signals from an OpenGL drawing area called a Rendering Context (RC) in a window or component . These functions may be defined by the operating system, third party widget set, or by a cross platform OpenGL library.
    - OS Defined Libraries:
        - Windows: WGL functions
        - XWindows (Linux, most UNIXs): GLX functions
        - Mac OS X: AGL, CGL, NSOpenGL classes
    - Cross Platform OpenGL Libraries
        - SDL
        - GLUT (pronounced gloot)
    - Cross Platform Widget Sets with OpenGL support
        - wxWidgets
        - fltk
        - tcl/tk
        - Qt

- **OpenGL API** functions defined in OpenGL.h and provided by the active OpenGL driver (specified by the RC) are used to draw.
    - There are some optional helper functions defined in glu.h which simplify some of the more complex OpenGL tasks.
    - Some of the Special OpenGL libraries also provide special drawing functions for certain shapes or text.
- **OpenGL Drivers** implement the details of OpenGL functions either by passing data directly to 3D hardware, performing computations on the main CPU or some combination of the two.
    - Whether the programmer will render with dedicated hardware or with a software implementation is determined by what OpenGL features the programmer requests with the Windowing API and what the computer's hardware supports.

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.

Object-oriented programming was developed, in part, to aid with modularising the components of Model-View-Architecture programs. Most user interface APIs are written in an Object Oriented language.

Structuring programs to handle these three aspects of an interactive program. A 3D graphics:

- ***Model*** consists of descriptions of the geometry, positioning, apperance and lighting in your scene. Ideally these could be saved to and loaded from a file.
- ***View*** consists of the OpenGL rendering calls that set up your rendering area, interpret the document, and send things to be drawn.

Controller **is usually a set functions you write that respond to** event **signals sent to your program via the Windowing API.**

OpenGL Command Syntax

Uses the prefix `gl` and initial capital letters for each word making up the command name

e.g. `glClearColor(---------); glVertex3f(---------);`

OpenGL defined constants begin with GL and use all capital letters and underscores to separate words e.g. GL_COLOR_BUFFER_BIT

```
#include <necessary header files>
main(){
      InitializeWindow();
      Rendering operations();
      UpdateWindowAndCheckForEvents();
}
```

Data types

Data type	Corresponding C language	OpenGL Type Definition
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 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

GLUT provides several routines for opening windows detecting input and creating complicated 3D objects like spehere , 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 any 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);
glutInitWindowPosition(int x, int y)
```

- specifies screen location for the upper loft 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))
```

- The first and 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

```
glutMainLoop()
```

- This is an infinite loop
- All the windows that have been created are now shown infinitely

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

Examples

```
#include <windows.h>
#include <gl/glut.h>
#include <gl/glu.h>
void display(){
}
void main(int argc, char **argv){
 glutInit(&argc,argv);
 glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
 glutInitWindowPosition(100,100);
 glutInitWindowSize(100,100);
 glutCreateWindow("saroj");
 glutDisplayFunc(display);
 glutMainLoop();
}
//CLEARING THE BACKGROUND
void display(){
 glClearColor(1.0,0.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
}
```

```
// 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();

//ADDED TO DRAW A POLYGON
void display(){
  glClearColor(1.0,0.0,1.0,1.0);
  glClear(GL_COLOR_BUFFER_BIT);
  glShadeModel(GL_FLAT);
  glColor3f(0.0,1.0,0.0);
// TO SPECIFY FILL COLOR
  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();
 glFlush();
}
```

More Examples

To create a window

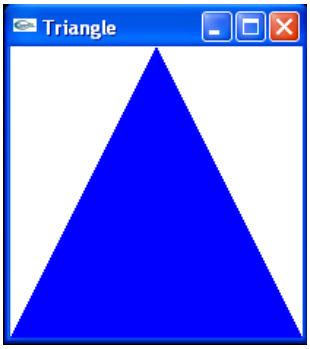
```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
void display(){

}
void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(111,111);
  glutInitWindowSize(200,100);
  glutCreateWindow("dsaroj");
  glutDisplayFunc(display);
  glutMainLoop();}
```

To clear the background to a desired color, add these code to the display function

```
void display(){
 glClearColor(1.0,0.0,1.0,1.0);
 glClear(GL_COLOR_BUFFER_BIT);
 glFlush();
}
```

Drawing a Triangle

```
static 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);  //fill color
  glBegin(GL_POLYGON);
    glVertex3f(0.0,0.0,0.0);
    glVertex3f(50.0,0.0,0.0);
    glVertex3f(25.0,50.0,0.0);
  glEnd();
glFlush();
}
```




```
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.

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();
}
void mouse(int button, int state, int x, int y){
   switch(button){
        case GLUT_LEFT_BUTTON:
          if (state == GLUT_DOWN)
            glutIdleFunc(trans);
         break;
        case GLUT_RIGHT_BUTTON:
          if (state == GLUT_DOWN)
            glutIdleFunc(NULL);
         break;
}
void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowPosition(111,111);
  glutInitWindowSize(200,100);
  glutCreateWindow("saroj");
  glutDisplayFunc(display);
  glutMouseFunc(mouse);
  glutMainLoop();}
```

Getting input from a Keyboard

```
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();
}
void key(unsigned char key, int x, int y){
    switch(key){
       case 'a':
         glutIdleFunc(trans);
        break;
       case 'f':
         glutIdleFunc(NULL);
        break;
      }
}
```

To rotate a triangle

```
glRotatef(type angle, type x, type y, type z);
```

multiplies current matrix by a matrix that rotates an object in counter clockwise direction

```
glTranslatef(type x, type y, type z);
```

multiplies current matrix by a matrix that translates an object by th e give x, y , z values

```
glScalef(type x, type y, type z);
```

multiplies current matrix by a matrix that stretches or shrinks an object

Smooth Shading (Intensity Interpolation Scheme)

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
#include <stdlib.h>
#include <math.h>
```

Draw triangle with three vertices with different colors

```
void display(){
glClearColor(0.0,0.0,0.0,0.0);
//glShadeModel(GL_FLAT);       //CIS
glShadeModel(GL_SMOOTH);  //Gouroud Shading
glClear(GL_COLOR_BUFFER_BIT);
glutWireSphere(1.0,20,16);
glBegin(GL_TRIANGLES);
     glColor3f(0.0,1.0, 0.0); //green
     glVertex2f(25.0,  5.0);

     glColor3f(0.0,0.0, 1.0); //blue
     glVertex2f(5.0,  25.0);

     glColor3f(1.0,0.0, 0.0); //red
     glVertex2f(5.0,  5.0);
 glEnd();
 glFlush();
}
```

Define Coordinate System and create view port

```
void reshape(int w,int h){
 glViewport(0,0,(GLsizei)w,(GLsizei)h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
  if(w<= h)
    gluOrtho2D(0.0,250.0,0.0,250.0);
  else
    gluOrtho2D(0.0,30.0,0.0,30.0*(GLfloat)h/(GLfloat)w);
 glMatrixMode(GL_MODELVIEW);

}
```

```
int main(int argc, char* argv[]){
   glutInit(&argc,argv);
   glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
   glutInitWindowPosition(100,100);
   glutInitWindowSize(600,600);
   glutCreateWindow("Intensity Interpolation Scheme");
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMainLoop();
 return 0;
}
```

//Animation using Double Buffer

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glu.h>
#include <gl/glut.h>
static GLfloat spins= 0.0;

//Create a rectangle
void display(){
  glClearColor(0.0,0.0,0.0,0.0);
  glShadeModel(GL_FLAT);
  glClear(GL_COLOR_BUFFER_BIT);
  glPushMatrix();
       glRotatef(spins,0.0,0.0,1.0);
       glColor3f(1.9,1.0,0.8);
       glRectf(-25.0,-25.0,25.0,25.0);
  glPopMatrix();
  glutSwapBuffers();
}
```

//Function for changing the value of spins

```
void spin(){
   spins = spins + 2;
    if( spins > 360.0 )
        spins = spins - 360.0;
    glutPostRedisplay();
}
```

//Trap mouse click event

```
void mouse(int button,int state, int x, int y){
    switch(button){
      case GLUT_LEFT_BUTTON:
         if(state == GLUT_DOWN)
      glutIdleFunc(spin);     //NULL to Pause
      break;
      }
}
```

//Specify coordinate system

```
void reshape(int w, int h){
 glViewport(0,0,(GLsizei)w,(GLsizei)h);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-50.0,50.0,-50.0,50.0,-1.0,1.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
}
void main(int argc, char* argv[ ]){
   glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);
   glutCreateWindow("Animation");
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutMouseFunc(mouse);
   glutMainLoop();
}
```

Font Generation

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
GLubyte rasters[24] = {
0xc0,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00,
0xff,0x00, 0xff,0x00, 0xc0,0x00, 0xc0,0x00, 0xc0,0x00,
0xff,0xc0, 0xff,0xc0,
};
void init(){
 glPixelStorei(GL_UNPACK_ALIGNMENT,1);
 glClearColor(0.0,0.0,0.0,0.0);
}

void display(void){
 glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1,1,1);
glFlush();
}
void show(){
glRasterPos2i(20,20);
glBitmap(10,12,0,0,11,0,rasters);
glBitmap(10,12,0,0,11,0,rasters);
glBitmap(10,12,0,0,11,0,rasters);
glFlush();
}
void reshape(int w, int h){
  glViewport(0,0,(GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0, w,0 , h, -1.0,1.0);
  glMatrixMode(GL_MODELVIEW);
}

void keyboard(unsigned char key, int x, int y){
    switch(key){
      case 'f':
        show();
        case 'a':
              exit(0);
```

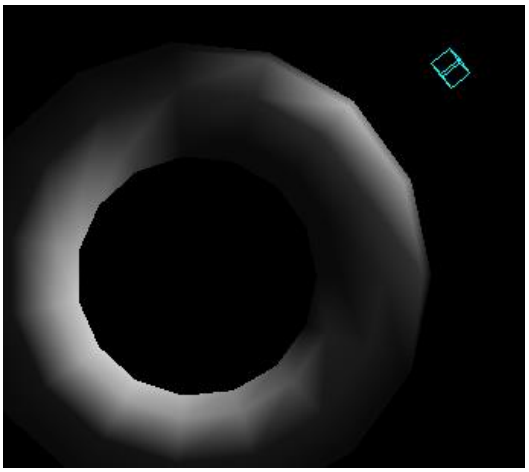
```
}
 }

void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
}
```

Creating a lighting effect

```
#include <windows.h>
#include <gl/gl.h>
#include <gl/glut.h>
#include <gl/glu.h>
static int spin = 0;

void init(){
 glClearColor(0.0,0.0,0.0,0.0);
 glShadeModel(GL_SMOOTH);
 glEnable(GL_LIGHTING);
 glEnable(GL_LIGHT0);
 glEnable(GL_DEPTH_TEST);
}

void display(){
GLfloat position[]={0.0,0.0,1.5,1.0};
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
 glPushMatrix();
      glTranslatef(0.0,0.0,-5.0);
      glPushMatrix();
           glRotated ((GLdouble)spin,1.0,0.0,1.0);
           glLightfv(GL_LIGHT0,GL_POSITION,position);
           glTranslated(0.0,0.0,1.5);
           glDisable(GL_LIGHTING);
```

```
            glColor3f(0.0,1.0,1.0);
            glutWireCube(0.1);
           glEnable(GL_LIGHTING);
      glPopMatrix();
            glutSolidTorus(0.275,0.85,8,15);
 glPopMatrix();
 glFlush();
}

void reshape(int w, int h){
  glViewport(0,0,(GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0,(GLfloat) w / (GLfloat)h, 1.0,20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

}
void mouse(int button, int state, int x, int y){
    switch(button){
      case GLUT_LEFT_BUTTON:
         if (button == GLUT_DOWN){
      spin = (spin + 30) % 360;
      glutPostRedisplay();
         }
         break;
         default:
            break;
    }
 }

void main(int argc, char **argv){
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(100,100);
  glutCreateWindow(argv[0]);
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
```

```
  glutMainLoop();
}
```



`glLightfv(GL_LIGHT0,GL_POSITION,position):`

- Set light-source parameters.

```
void glLightfv(
  GLenum light,
  GLenum pname,
  const GLfloat *params
);
```

light

The identifier of a light

GL_LIGHTi where $0 \leq i < GL\_MAX\_LIGHTS$

At least eight lights are supported.

pname

A single-valued light-source parameter for *light*

param

The value to which parameter *pname* of light source *light* will be set.

PHIGS

PHIGS is an international **ISO standard of functional interface** between an application program and a configuration of graphical input and output devices.

This interface contains basic functions for dynamic interactive 2D and 3D graphics on wide variety of graphics equipment

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 graphics called **structure networks**

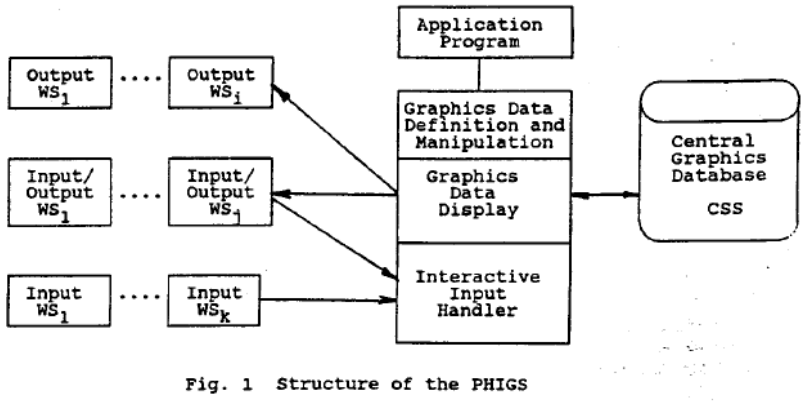


Fig. 1 Structure of the PHIGS

The two abstract concepts (input and output) are the building blocks of an **abstract 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**

Each workstation has a type falling into one of five categories

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

For every type of workstation present in a PHIGS implementation**, there exists a generic workstation description table** which describes the standard capabilities and characteristics of the workstation.

When the workstation is opened, a **new specification workstation description table is created** for that workstation description table obtained from the device itself. And possibly from other implementation dependent source.

The content of the specific workstation description table may change at any time while the workstation is open.

The application program can inquire which generic capabilities are available before the workstation is open.

The specific capabilities may be inquired while the workstation is open by first inquiring the workstation type of an open workstation to obtain the workstation type of the specific workstation description table, and then using this workstation type as a parameter to the inquiry functions which query the workstation description table. This information may be used by the application program to adapt its behavior accordingly

The application program references a workstation by means of a workstation identifier. Connection to a particular workstation is established by the function OPEN WORKSTATION which associates the workstation identifier with a generic workstation type and a connection identifier.

The current state of each open workstation . state values of the WSSL maybe set up by the “set functions” and may be inquired by “inquire functions”.

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

Graphical Output

Picture generated by PHIGS are build up of basic pieces called output primitives. Output primitives are generated from structure elements by structural traversal

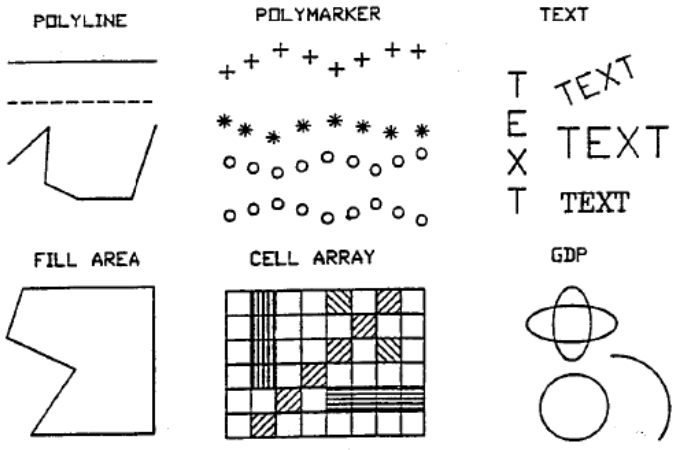


Fig. 2 Examples of output primitives

POLYLINE set of connected lines defined by point sequences POLYLINE3

POLYMARKER set of symbols of one type centered at a given position POLYMARKER 3

TEXT character string at a given position on an arbitrary plane in modeling coordinate space

ANNOTATION TEXT RELATIVE character string at specified position in an x y plane parallel to view plane ANNOTATION TEXT RELATIVE 3

FILL AREA single polygonal area which may be hollow or filled with uniform color, pattern or hatch style

FILL AREA SET a set of polygonal areas which may be filled similar as FILL AREA

CELL ARRAY two dimensional array of cells with individual colo9ur4s

GENERALIZED DRAWING PRIMITIVE special geometrical output capabilities of a workstation such as the drawing of spline curves, circular arcs, elliptic arcs

for individual specifications of aspects, there is a separate attribute for each aspect. These attributes are workstation independent

bundled aspects are selected by a bundle index into a bundle table each entry of which contains non geometric aspects of a primitive. The non geometric aspects are workstation dependent in

that each workstation has its own set of bundle tables (stored in the workstation state list) the values in a particular bundle may be different for different workstations

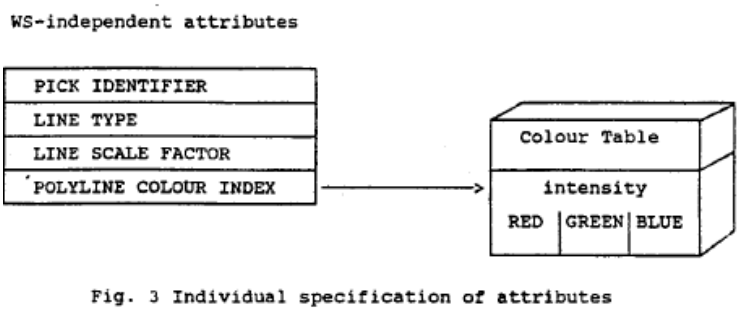


Fig. 3 Individual specification of attributes

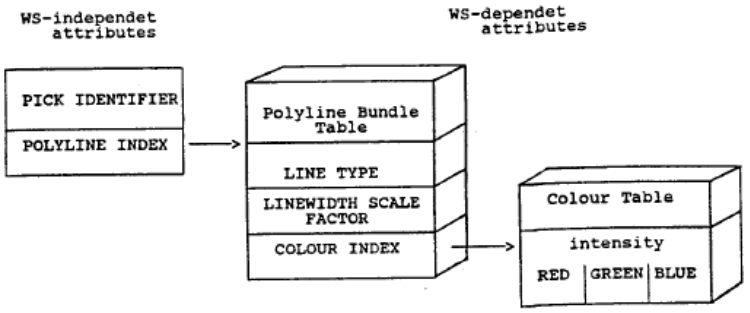


Fig. 4 Bundled specification of attributes

Output primitive attributes

Attributes of the first type control the geometric aspects of primitives. These are aspects that affect the shape or size for the entire primitive (e.g. `CHARACTER HEIGHT` for `TEXT`)

Hence they are sometimes referred to as geometric attributes. Geometric attributes are workstation independent and if they represent coordinate data they are expressed in modeling coordinates

Attributes of the second type control the non geometric aspects of primitives these are aspects that affect a primitive's appearance(for example. Line type for `POLYLINE`, or color index for all primitives except `CELL ARRAY`) or the shape or size of the component parts of the primitive (for example , marker size scale factor for `POLYMARKER`)

Non geometric aspects do not represent coordinate data

The non geometric aspects of primitive may be specified either via a bundle or individually the geometric aspects only individually

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

Each structure is identified by unique name which is specified by the application

element	attribut	element	attribut
POLYLINE	POLYLINE INDEX LINETYPE LINEWIDTH SCALE FACTOR POLYLINE COLOUR INDEX LINETYPE ASF LINEWIDTH SCALE FACTOR ASF POLYLINE COLOUR INDEX ASF	FILL AREA	INTERIOR INDEX INTERIOR STYLE INTERIOR STYLE INDEX INTERIOR COLOUR INDEX INTERIOR STYLE ASF INTERIOR STYLE INDEX ASF INTERIOR COLOUR INDEX ASF PATTERN SIZE PATTERN REFERENCE POINT PATTERN REFERENCE VECTORS
POLY-MARKER	POLYMARKER INDEX MARKER TYPE MARKER SIZE SCALE FACTOR POLYMARKER COLOUR INDEX MARKER TYPE ASF MARKER SIZE FACTOR ASF POLYMARKER COLOUR INDEX AS	FILL AREA SET	see FILL AREA and addition EDGE INDEX EDGE FLAG EDGETYPE EDGEWIDTH SCALE FACTOR EDGE COLOUR INDEX EDGE FLAG ASF EDGETYPE ASF EDGEWIDTH SCALE FACTOR ASF EDGE COLOUR INDEX ASF
TEXT	TEXT INDEX TEXT FONT TEXT PRECISION CHARACTER EXPANSION FACTOR CHARACTER SPACING TEXT COLOUR INDEX TEXT FONT ASF TEXT PRECISION ASF CHARACTER EXPANSION FACTOR CHARACTER SPACING ASF TEXT COLOUR INDEX ASF CHARACTER HEIGHT CHARACTER UP VECTOR TEXT PATH TEXT ALIGNMENT	CELL ARRAY	zero attributes
		GDP	Zero or more of attributes of POLYLINE or FILL AREA or FILL AREA SET
ANNO-TATION TEXT RELATIVE	nongeometric attributes see TEXT attributes ANNOTATION TEXT CHARACTER HEIGHT ANNOTATION TEXT CHARACTER UP VECTOR ANNOTATION TEXT PATH ANNOTATION TEXT ALIGNEMT ANNOTATION STYLE		

Structure networks are organized as directed acyclic graphs

So a structure may contain invocations of other structures containing in CSS. The invocation of a structure is achieved using the **execute structure element**. Such an invocation is known as a **structure reference**. Structure can not be referenced recursively

A structure can be created in one of the following ways:

- When a reference to the nonexistent structure is inserted into a structure in the CSS
- When the structure is opened for the first time (function `OPEN STRUCTURE`)
- When the structure is posted for display on a workstation (`POST STRUCTURE`)
- When the structure is referenced in any function changing the structure identifier
- When the not existing structure in CSS is retrieved from an archive (`RETRIEVE STRUCTURE`)
- When the not existing structure is emptied (`EMPTY STRUCTURE`)

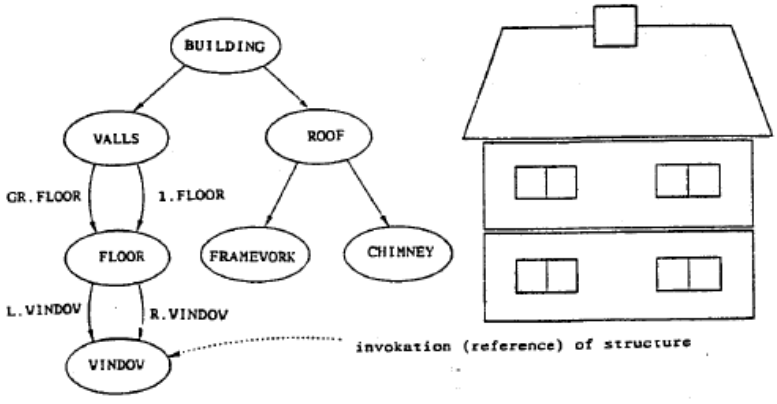


Fig. 5 Building and its structure network

Structure traversal and display

- A structure network is identified for display on a workstation by posting function POST STRUCTURE

A structure may be displayed only if it is a member of a posted structure network

To display a network the structure elements have to be extracted from the CSS and processed . that process is called traversal process

The traversal process interprets each structure element in the structure network sequentially starting at the first element of the top of the network

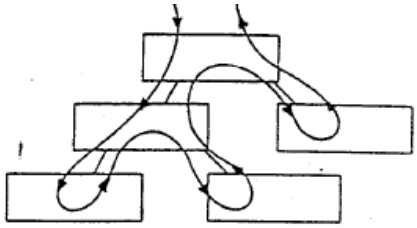


Fig. 6 Traversal process

A traversal state list is associated with each traversal process. Values in this state list ;may be accessed when the structure elements are interpreted

Each time a traversal is initiated the associated list is initialized

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. Navigate within the structure and inquire structure element content

A structure is identified for editing an element pointer is established which points at the lat element in the structure . 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

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

The workstation dependent stage then performs a transformation on the geometrical information contained in output primitives , attributes and logical input values

These 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

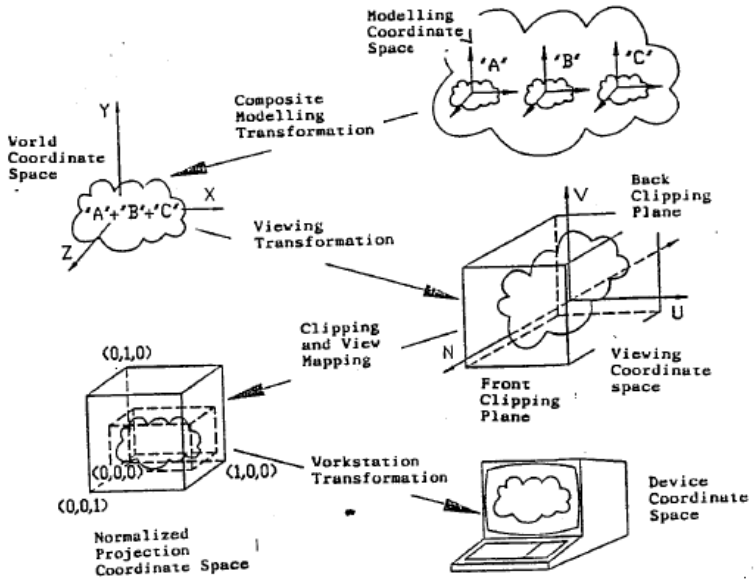


Fig. 7 Coordinate systems and transformations in PHIGS

Output primitives and attributes are mapped from world coordinate to view reference coordinate by the **view orientation transformation**, from view reference coordinates to normalized projection coordinates by the **view mapping transformation** and from normalized projection coordinates to device coordinates by the **workstation transformation**. Hidden lines removals and clippings are also done during the mapping process

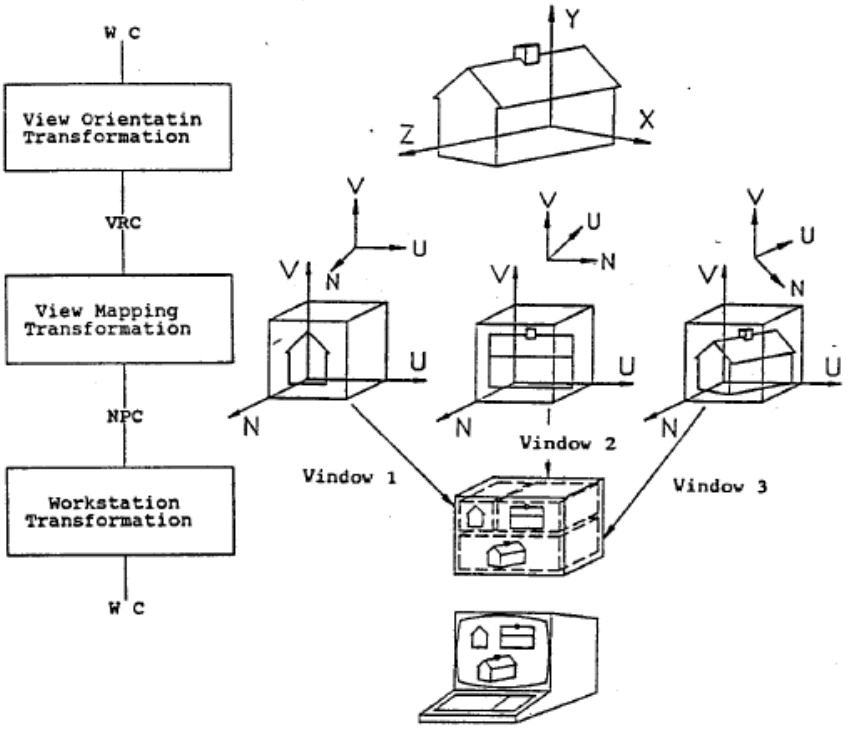


Fig.8 Viewing transformations in PHIGS

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

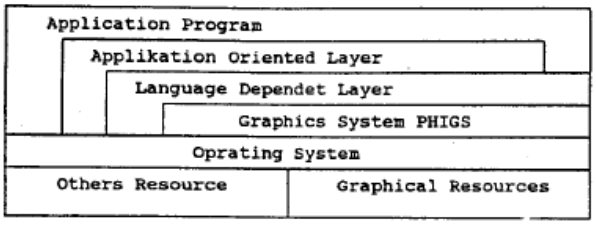


Fig. 9. Layer model of PHIGS

7. Graphics system PHIGS +

PHIGS does not support a shading of pictures and modeling a non uniform rational B Spline curves and surfaces (NURBS)

An extension of PHIGS for these functionalities is 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, crates 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

GKS (the Graphical Kernel System) is an ANSI and ISO standard.

GKS **standardizes two-dimensional graphics functionality** at a relatively 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.

It consists of **three basic parts**:

1. An **informal exposition** of the contents of the standard which includes such things as how text is positioned, how polygonal areas are to be filled, and so forth.

2. A **formalization of the expository material** by way of abstracting the ideas into discrete functional descriptions. These functional descriptions **contain such information as descriptions of input and output parameters, precise descriptions of the effect each function should have, references into the expository material, and a description of error conditions**. The functional descriptions in this section are language independent.

3. **Language bindings**. These bindings are an implementation of these abstract functions in a specific computer language such as Fortran or Ada or C.

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: which draws a sequence of connected line segments.
Polymarker: which marks a sequence of points with the same symbol.
Fill area: which displays a specified area.
Text: which draws a string of characters.
Cell array: which 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.

For example, the parameters of the **Text** primitive are the **string or characters to be drawn** and the **starting position of that string**. Thus:

TEXT(X, Y, 'ABC') will draw the characters ABC at the position (X, Y).

Although the parameters enable the form of the primitives to be specified, additional data are necessary to describe the actual appearance (or aspects) of the primitives.

For example, 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**.

Polylines

The GKS function for **drawing line segments** is called Polyline. The polyline function **takes an array of X-Y coordinates** and draws line segments connecting them. The attributes that control the appearance of a polyline are:
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.

The main line drawing primitive of GKS is the polyline which is generated by calling the function:
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.

Polymarkers

The GKS polymarker function **allows drawing marker symbols** centered at coordinate points specified.
The attributes that control the appearance of polymarkers are:
Marker, which 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, which controls **how large** each marker is (except for the dot marker).

Polymarker color index, which specifies **what color** the marker is.

GKS provides the primitive polymarker marks a set of points, instead of drawing lines through a set of points.

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(l9, XDK, YDK)

Text

The GKS text function **allows drawing a text string** at a specified coordinate position.
The attributes that control the appearance of text are:
Text font and **precision**, which specifies **what text font** should be used for the characters and **how precisely** their representation should adhere to the settings of the other text attributes.

Character expansion factor, which **controls the height-to-width ratio** of each plotted character.

Character spacing, which specifies **how much additional white space** should be inserted between characters in a string.

Text color index, which specifies **what color** the text string should be.

Character height, which specifies **how large** the characters should be.

Character up vector, which specifies at **what angle** the text should be drawn.

Text path, which specifies in what direction the text should be written (right, left, up, or down).

Text alignment, which 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.

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)

Fill Area

The GKS fill area function **allows specifying a polygonal shape of an area to be filled** with various interior styles.
The attributes that control the appearance of fill areas are:
Fill area interior style, which specifies **how the polygonal area should be filled**: with solid colors or various hatch patterns, or with nothing, that is, a line is drawn to connect the points of the polygon, so to get only a border.
Fill area style index. If the fill area style is hatch, this index specifies **which hatch pattern** is to be used: horizontal lines; vertical lines; left slant lines; right slant lines; horizontal and vertical lines; or left slant and right slant lines.
Fill area color index, which specifies **the color of the fill patterns** or solid areas.

At the same time, there are now many devices which have the concept of an area which may be filled in some way. These vary from intelligent pen plotters which can cross-hatch an area to raster displays which can completely fill an area with a single colour or in some cases fill an area by repeating a pattern.

GKS provides a fill area function to satisfy the application needs which can use the varying device capabilities. Defining an area is a fairly simple extension of defining a polyline. An array of points is specified which defines the boundary of the area. If the area is not closed (i.e. the first point is not the same as the last point), the boundary is the polyline defined by the points but extended to join the last point to the first point. A fill area may be generated by invoking the function:

FILL AREA(N, XPTS, YPTS)
where, as usual XPTS and YPTS are arrays giving the N points (XPTS(1), YPTS(1)) to (XPTS(N), YPTS(N)).

Cell Array

The GKS cell array function **displays raster like images** in a device-independent manner.

The cell array function takes the two corner points of a rectangle specified, a number of divisions (M) in the X direction and a number of divisions (N) in the Y direction. It then partitions the rectangle into M x N subrectangles called cells. Assign each cell a color and create the final cell array by coloring each individual cell with its assigned color.