

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, the user controls the contents, structure, and appearance of images of the objects by using input devices, such as 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 points on the screen. Each point is called a pixel, short for "Picture Elements". Once bitmap graphics became affordable, an explosion of easy-to-use and inexpensive graphics-based applications soon followed. 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 concepts of a "desktop" now became a popular 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(e.g.. 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.

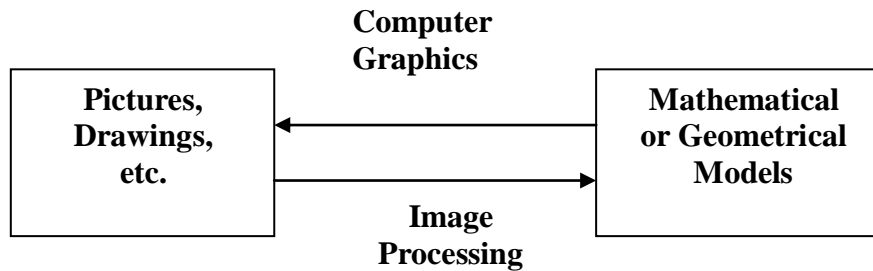
At the same time, it was becoming clear to computer, automobile, and aerospace manufacturers that CAD and computer-aided manufacturing(CAM) activities had enormous potential for automating drafting and other drawing-intensive activities. The General Motors CAD system for automobile design , and the Itek 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 whereas other used punch cards, a non-interactive system. Among the reasons for this were these:

- The high cost of the graphics hardware-at a time when automobiles cost a few thousand dollars, computers cost several millions of dollars, and the first 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.

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

The Difference between Computer Graphics and Image Processing:



- **Computer Graphics:** Synthesize pictures from mathematical or geometrical models.
- **Image Processing:** analyze pictures to derive descriptions (often in mathematical or geometrical forms) of objects appeared in the pictures.

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 desktop windows 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 all are 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 the 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 whereas 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 the huge amount of data sets. It would , otherwise , be almost impossible to analyze those data numerically.

6. Simulation and modeling

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 without 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 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 Kings and The Beauty of Beast, and other scientific movies like Jurassic Park, The lost world etc are the best example of the application of computer graphics in the field of entertainment. Instead of drawing all 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 of production while maintaining the quality. Computer and video games such FIFA, Doom ,Pools are few to name where graphics is used extensively.

8. Art and commerce

Here computer graphics is used to produce pictures that express 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 a 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.

Main Subjects in Computer Graphics Research

- **Mathematical and geometrical modeling**
- **Rendering algorithms**
- **Animation techniques**
- **Input and output technologies**
- **Graphics architecture**

Graphics Software

General classifications

- **General programming packages** provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN. Eg. GL (Graphics Library) system on Silicon Graphics equipment
 - include basic functions for generating picture components (straight lines, polygons, circles, and other figures), setting colors and intensity values, selecting views, and applying transformations

- **Special purpose applications packages** are, in contrast designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work. The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. Eg. the artist's painting programs and various business, medical, and *Computer-Aided Design (CAD) systems*.

Computer-Aided Design (CAD) is the use of computer technology to aid in the design of a product, particularly the drafting of a part or the product—a part visual (drawing) and part symbol method of communications particular to a specific technical field. It is in origination, the use of computers to aid the art of drafting—the integral communications of technical drawings — which for a three dimensional object are typically represented by three projected views at right angles —drafting is the Industrial arts sub-discipline which underlies all involved technical endeavors. Current CAD software packages range from 2D vector base drafting systems to 3D solid and surface modellers. Modern CAD packages can also frequently allow rotations in three dimensions, allowing viewing of a designed object from any desired angle, even from the inside looking out. Some CAD software is capable of dynamic mathematic modeling, in which case it may be marketed as CADD — Computer Aided Design and Drafting.

Software standards

The primary goal of standardized graphics software is *portability*. When packages are designed with standard graphics functions, software can be moved easily from one hardware system to another and used in different implementations and applications. Without standards, programs designed for one hardware system often cannot be transferred to another system without extensive rewriting of the programs.

- The International Standards Organization (ISO) and American Nation Standards Institute (ANSI) adopted **General Kernel System (GKS)** as the first graphics software standard.
- **PHIGS (Programmer's Hierarchical Interactive Graphics Standard)**, which is an extension of GKS, increased capabilities for object modeling, color specifications, surface rendering, and picture manipulations. **PHIGS+** was developed to provide three-dimensional surface-shading capabilities not available in PHIGS.

Although PHIGS presents a specification for basic graphics functions, it does not provide a standard methodology for a graphics interface to output devices. Nor does it specify methods for storing and transmitting pictures. Separate standards have been developed for these areas. Standardization for device interface methods is given in the **Computer Graphics Interface (CGI)** system. And **Computer Graphics Metafile (CGM)** system specifies standards for archiving and transporting pictures.

PHIGS (Programmer's Hierarchical Interactive Graphics System) is an API standard for rendering 3D computer graphics, at one time considered to be the 3D graphics standard for the 1990s. Instead a combination of features and power led to the rise of OpenGL, which remains the de facto 3D standard to this day. PHIGS is no longer used.

PHIGS was available as a standalone implementation (examples: Digital Equipment Corporation's DEC PHIGS, IBM's graPHIGS, Sun's SunPHIGS) and also used with the X Window system, supported via PEX, the "PHIGS Extension to X". PEX consisted of an extension to X, adding commands that would be forwarded from the X server to the PEX system for rendering. Workstations were placed in windows typically, but could also be forwarded to take over the whole screen, or to various printer-output devices.

PHIGS originally lacked the capability to render illuminated scenes, and was superseded by **PHIGS+**. PHIGS+ works in essentially the same manner, but added methods for lighting a 3D scene, and was often referred to as PHIGS PLUS (where the PLUS was a slightly tongue-in-cheek acronym for "Plus Lumière Und Shading"). PHIGS+ also introduced more advanced graphics primitives.

OpenGL (Open Graphics Library) is a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls, which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992[1] and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL).

OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

The Rise of OpenGL and the Decline of PHIGS

[OpenGL](#), unlike PHIGS, is an immediate-mode rendering system with no "state"; once an object is sent to a view to be rendered it essentially disappears. Changes to the model have to be re-sent into the system and re-rendered, dramatically increasing programmer workload. For simple projects, PHIGS was considerably easier to use and work with.

However, OpenGL's "low-level" API allowed the programmer to make dramatic improvements in rendering performance by first examining the data on the [CPU](#)-side before trying to send it over the bus to the graphics engine. For instance, the programmer could "cull" the objects by examining which objects were actually visible in the scene,

and sending only those objects that would actually end up on the screen. This was kept private in PHIGS, making it much more difficult to tune performance.

Given the low performance systems of the era and the need for high-performance rendering, OpenGL was generally considered to be much more "powerful" for 3D programming. PHIGS fell into disuse. Version 6.0 of the PEX protocol was designed to support other 3D programming models as well, but did not regain popularity. PEX was mostly removed from [XFree86 4.2.x \(2002\)](#) and finally removed from the X Window System altogether in X11R6.7.0 (April [2004](#)) [[2](#)]

Major Graphic File Formats

<http://en.wikipedia.org/wiki/Graphics_file_format>

Image file formats provide a standardized method of organizing and storing image data. Image files are made up of either [pixel](#) or [vector \(geometric\)](#) data, which is [rasterized](#) to pixels in the display process, with a few exceptions in [vector graphic display](#). The pixels that make up an image are in the form of a grid of columns and rows. Each pixel in an image consists of numbers representing brightness and color.

Image file sizes, expressed in [bytes](#), increase with the number of pixels in the image, and the color depth of the pixels. The more rows and columns, the greater the [image resolution](#) and the greater the file size. Also, each pixel making up the image increases in size as color depth is increased. An 8-bit pixel (1 byte) can store 256 colors and a 24-bit pixel (3 bytes) can store 16 million colors. The latter is known as [truecolor](#).

Image compression is a method of using [algorithms](#) to decrease file size. High resolution cameras produce large image files. File sizes may range from hundreds of kilobytes to many megabytes depending on the resolution of the camera and the format used to save the images. High resolution [digital cameras](#) record 8 megapixels (MP) (1MP= 1000000 pixels/ 1 million) images, or more, in truecolor. Consider an image taken by an 8 MP camera. Since each of the pixels uses 3 bytes to record true color, the uncompressed image would occupy 24,000,000 bytes of memory. That is a lot of storage space for just one image, and cameras must store many images to be practical. Faced with large file sizes, both within the camera, and later on disc, image file formats have been developed to address the storage problem. An overview of the [major graphic file formats](#) is given below.

There are two types of **image file compression** algorithms: [lossy](#) and [lossless](#).

Lossless compression algorithms reduce file size with no loss in image quality, though they usually do not compress to as small a file as a lossy method does. When image quality is valued above file size, lossless algorithms are typically chosen.

Lossy compression algorithms take advantage of the inherent limitations of the human eye and discard information that cannot be seen. Most lossy compression algorithms allow for variable levels of quality (compression) and as these levels are increased, file size is reduced. At the highest compression levels, image deterioration becomes noticeable. This deterioration is known as compression artifacting.

- **JPEG (Joint Photographic Experts Group)** files are a lossy format (in most cases). The DOS filename extension is JPG, although other operating systems may use JPEG. Nearly all digital cameras have the option to save images in JPEG format. The JPEG format supports 8 bits per color – red, green, and blue, for 24-bit total – and produces relatively small file sizes. The compression when not too severe does not detract noticeably from the image. But JPEG files can suffer generational degradation when repeatedly edited and saved. Photographic images may be better stored in a lossless non-JPEG format if they will be re-edited in future, or if the presence of small "artifacts" (blemishes), due to the nature of the JPEG compression algorithm, is unacceptable. JPEG is also used as the image compression algorithm in many Adobe PDF files.
- **TIFF (Tagged Image File Format)** is a flexible image format that normally saves 8 or 16 bits per color – red, green and blue – for a total of 24 or 48 bits, and uses a filename extension of TIFF or TIF. TIFF's flexibility is both a feature and a curse, with no single reader capable of handling all the different varieties of TIFF files. TIFF can be lossy or lossless. Some types of TIFF files offer relatively good lossless compression for bi-level (black and white, no grey) images. Some high-end digital cameras have the option to save images in the TIFF format, using the LZW compression algorithm for lossless storage. The TIFF image format is not widely supported by web browsers. TIFF is still widely accepted as a photograph file standard in the printing industry. TIFF is capable of handling device-specific color spaces, such as the CMYK defined by a particular set of printing press inks.
- **RAW** refers to a family of raw image formats that are options available on some digital cameras. These formats usually use a lossless or nearly-lossless compression, and produce file sizes much smaller than the TIFF formats of full-size processed images from the same cameras. The raw formats are not standardized or documented, and differ among camera manufacturers. Many graphic programs and image editors may not accept some or all of them, and some older ones have been effectively orphaned already. Adobe's Digital Negative specification is an attempt at standardizing a raw image format to be used by cameras, or for archival storage of image data converted from proprietary raw image formats.
- **PNG (Portable Network Graphics)** file format is regarded, and was made, as the free and open-source successor to the GIF file format. The PNG file format supports true color (16 million colors) whereas the GIF file format only allows

256 colors. PNG excels when the image has large areas of uniform color. The lossless PNG format is best suited for editing pictures, and the lossy formats like JPG are best for final distribution of photographic-type images because of smaller file size. Many older browsers do not yet support the PNG file format, however with the release of Internet Explorer 7 all popular modern browsers fully support PNG. The Adam7-interlacing allows an early preview even when only a small percentage of the data of the image has been transmitted.

- **GIF (Graphics Interchange Format)** is limited to an 8-bit palette, or 256 colors. This makes the GIF format suitable for storing graphics with relatively few colors such as simple diagrams, shapes, logos and cartoon style images. The GIF format supports animation and is still widely used to provide image animation effects. It also uses a lossless compression that is more effective when large areas have a single color, and ineffective for detailed images or dithered images.
- **BMP file format (Windows bitmap)** is used internally in the Microsoft Windows operating system to handle graphics images. These files are typically not compressed, resulting in large files. The main advantage of BMP files is their wide acceptance, simplicity, and use in Windows programs.

Which Graphics File Format Is Best To Use When?

Here are some general guidelines:

- If the images are for the Web or online, use JPEG, PNG, or GIF.
- If the images are for print, use TIFF.
- If you want to keep a version that remains editable, choose your software's [native file format](#). (PSD for Photoshop, PSP for Paint Shop Pro, CPT for Corel Photo-Paint, etc.)

Common	Graphic	File	Formats
<http://www.reasoft.com/articles/formats-1.shtml>			

Following are descriptions of some commonly used file formats:

GIF: The Graphics Interchange Format was originally developed by CompuServe in 1987. It is most commonly used for bitmap images composed of line drawings or blocks of a few distinct colors. The GIF format supports 8 bits of color information or less. In addition, the GIF89a file format supports transparency, allowing you to make a color in your image transparent. This feature makes GIF a particularly popular format for Web images.

JPEG: Like GIF, the Joint Photographic Experts Group format is one of the most popular formats for Web graphics. It supports 24 bits of color information, and is most commonly used for photographs and similar continuous-tone bitmap images. The JPEG file format stores all of the color information in an RGB image. JPEG was designed so

that changes made to the original image during conversion to JPEG would not be visible to the human eye. Most imaging applications let you control the amount of lossy compression performed on an image, so you can trade off image quality for smaller file size and vice versa. Be aware that the chances of degrading your image when converting it to JPEG increase proportionally with the amount of compression you use. Unlike GIF, JPEG does not support transparency.

BMP: The Bitmap file format is used for bitmap graphics on the Windows platform only. Unlike other file formats the BMP format stores image data from bottom to top and pixels in blue/green/red order. Compression of BMP files is not supported, so they are usually very large.

TIFF: The Tag Interchange File Format is a tag-based format that was developed and maintained by Aldus (now Adobe). TIF, which is used for bitmap images, is compatible with a wide range of software applications and can be used across platforms such as Macintosh, Windows, and UNIX. The TIFF format is complex, so TIFF files are generally larger than GIF or JPEG files. TIFF supports lossless LZW (Lempel-Ziv Welch) compression; however, compressed TIFFs take longer to open.

PNG: The Portable Network Graphics format will likely be the successor to the GIF file format. PNG is expected to become a mainstream format for Web images and could replace GIF entirely. It is platform independent and should be used for single images only (not animation). Compared with GIF, PNG offers greater color support and better compression, gamma correction for brightness control across platforms, better support for transparency, and a better method for displaying progressive images.

Hardware Concepts

Input Devices

1. Tablet:

A tablet is digitizer. In general a digitizer is a device which is used to scan over an object, and to 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 (pencil-shaped device) or puck (link mouse with cross hairs for sighting positions) held in the user's hand. A tablet is flat surface, and its size of the tablet varies from about 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 mainly three types of tablets.

a. Electrical tablet:

A grid of wires on $\frac{1}{4}$ to $\frac{1}{2}$ inch centers is embedded in the tablet surface and 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 is taken as "near" otherwise 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 time per second.

b. 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 does not require a dedicated working area for the microphones can be placed on any surface to form the "tablet" work area. This facilitates digitizing drawing on thick books. Because in an electrical tablet this is not convenient for the stylus can not get closer to the tablet surface.

c. 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 the conducting layer. The strength of he 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-rays 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.

2. Touch panel

The touch panel allows the users to point at the screen directly with a finger to move the cursor around the screen, or to select the icons. Following are the mostly used touch panels.

a. Optical touch panel

It uses a series of infra-red 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 finger's position. The cursor is then moved to this position, or the icon at this position is selected. It two parallel beams are broken, the finger is presumed to be

centered between them; if one is broken, the finger is presumed to be on the beam. There is a low-resolution panel, which offers 10 to 50 positions in each direction.

b. Sonic 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.

c. Electrical touch panel:

It consists of slightly separated two transparent plates 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 touched position. The resolution of this touch panel is similar to that of sonic touch panel.

3. 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} for the first scanning line. For second line, Y decreases by one and X again changes from 0 to X_{\max} , and so on. When the activated light pen "sees" a burst of light at certain position as the electron beam hits the phosphor coating at that position, it generates a electric pulse, which is used to save the video controller's X and Y registers and interrupt the computer. By reading the saved values, the graphics package can determine the coordinates of the position seen by the light pen. Because of the following drawbacks the light pens are not popular now a days.

- Light pen obscures the screen image as it is pointed to the required spot
- Prolong use of it can cause arm fatigue
- It can not 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
- It gives sometimes false reading due to background lighting in a room

4. Keyboard

A keyboard creates a code such as ASCII uniquely corresponding to a pressed key. It usually consists of alphanumeric keys, function keys, cursor-control keys, and separate numeric pad. It is used to move the cursor, to select the menu item, pre-defined functions. In computer graphics keyboard is mainly used for entering screen coordinates and text, to invoke certain functions. Now-a-days ergonomically designed keyboard (Ergonomic keyboard) with removable palm rests is available. The slope of each half of the keyboard can be adjusted separately.

5. Mouse

A mouse is a small hand-held device used to position the cursor on the screen. Mice are relative devices, that is, they can be picked up, moved in space, and then put down gain without any change in the reported position. For this, the computer maintains the current mouse position, which is incremented or decremented by the mouse movements. Following are the mice, which are mostly used in computer graphics.

a. Mechanical mouse

When a roller in the base of this mechanical mouse is moved, a pair of orthogonally arranged toothed wheels, each placed in between a LED and a photo detector, interrupts the light path. the number of interrupts so generated are used to report the mouse movements to the computer.

b. Optical mouse

The optical mouse is used on a special pad having a grid of alternating light and dark lines. A LED on the bottom of the mouse directs a beam of light down onto the pad, from which it is reflected and sensed by the detectors on the bottom of the mouse. As the mouse is moved, the reflected light beam is broken each time a dark line is crossed. The number of pulses so generated, which is equal to the number of lines crossed, are used to report mouse movements to the computer.

Display devices

The display devices used in graphics system is video monitor. The most common video monitor is based on CRT technology.

Cathode Ray Tube (CRT)

- CRT are the most common display devices on computer today. A CRT is an evacuated glass tube, with a heating element on one end and a phosphor-coated screen on the other end.
- When a current flows through this heating element (filament) the conductivity of metal is reduced due to high temperature. These cause electrons to pile up on the filament.
- These electrons are attracted to a strong positive charge from the outer surface of the focusing anode cylinder.
- Due to the weaker negative charge inside the cylinder, the electrons head towards the anode forced into a beam and accelerated by the inner cylinder walls in just the way that water is speeds up when its flow though a small diameter pipe.
- The forwarding fast electron beam is called Cathode Ray. A cathode ray tube is shown in figure below.

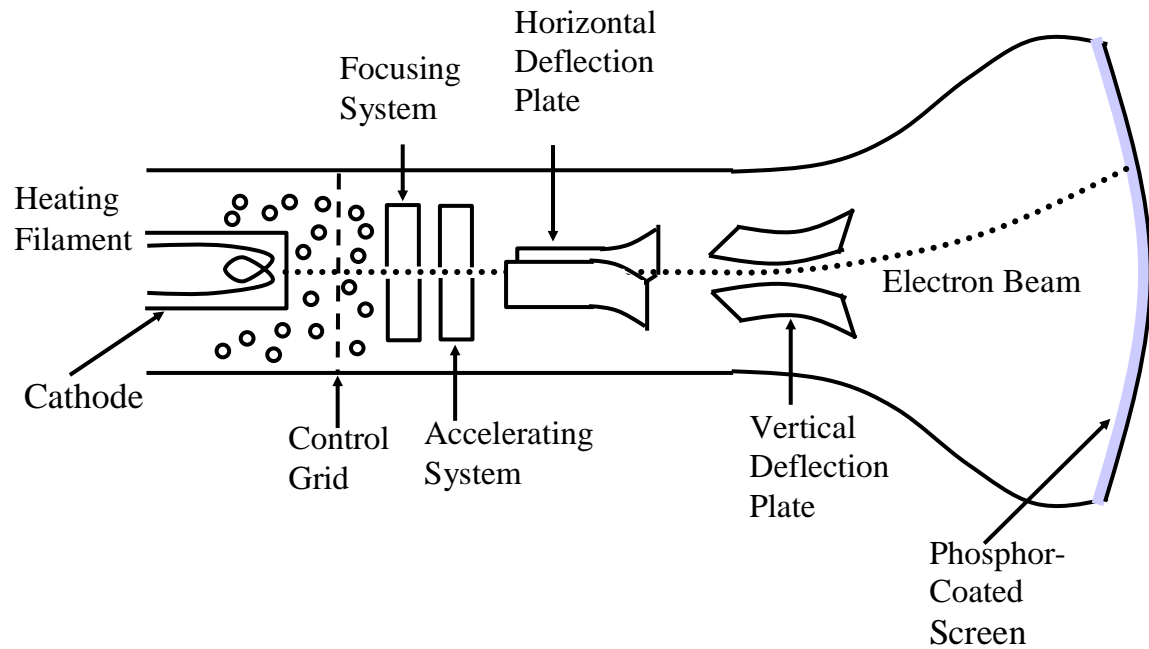


Figure :Cathode Ray Tube

- There are two sets of weakly charged deflection plates with oppositely charged, one positive and another negative. The first set displaces the beam up and down and the second displaces the beam left and right.
- The electrons are sent flying out of the neck of bottle (tube) until they smash into the phosphor coating on the other end.
- When electrons strike on phosphor coating, the phosphor then emits a small spot of light at each position contacted by electron beam. The glowing positions are used to represent the picture in the screen.
- The amount of light emitted by the phosphor coating depends on the no of electrons striking the screen. The brightness of the display is controlled by varying the voltage on the control grid.

Persistence:

- How long a phosphor continues to emit light after the electron beam is removed
- Persistence of phosphor is defined as **the time** it takes for emitted light to decay to **1/10 (10%)** of its original intensity. Range of persistence of different phosphors can react many seconds.
- Phosphors for graphical display have persistence of 10 to 60 microseconds. Phosphors with low persistence are useful for animation whereas high persistence phosphor is useful for highly complex, static pictures.

Refresh Rate:

- Light emitted by phosphor fades very rapidly, so to keep the drawn picture glowing constantly, it is required to redraw the picture repeatedly and quickly directing the electron beam back over the same point. The no of times/sec the image is redrawn to give a feeling of non-flickering pictures is called refresh-rate.
- If Refresh rate decreases, flicker develops.
- For refresh displays, it depends on picture complexity
- Refresh rate above which flickering stops and steady it may be called as critical fusion frequency(CFF).

Resolution:

Maximum number of points displayed horizontally and vertically without overlap on a display screen is called resolution. In other ways , resolution is referred as the no of points per inch(dpi/pixel per inch).

Raster-Scan Display

- Raster Scan Display is based on television technology. In raster-scan the electron beam is swept across the screen, one row at a time from top to bottom. No of scan line per second is called horizontal scan rate.
- As electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots. Picture definition is stored in a memory called frame buffer or refresh buffer. Frame buffer holds all the intensity value for screen points.

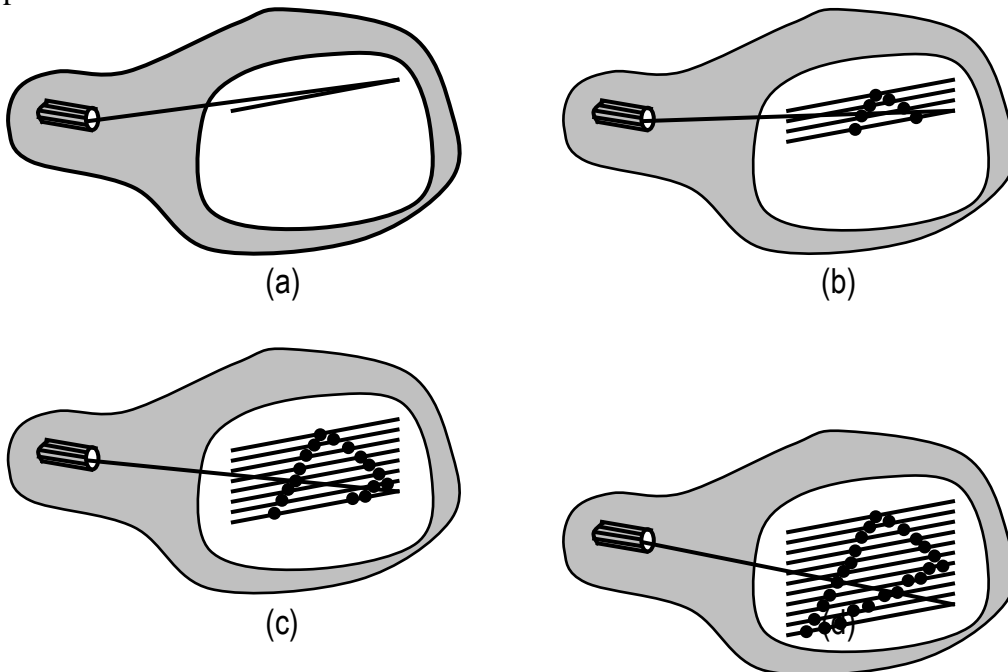


Figure: A raster-scan system displays an object as a set of points across each screen scan line

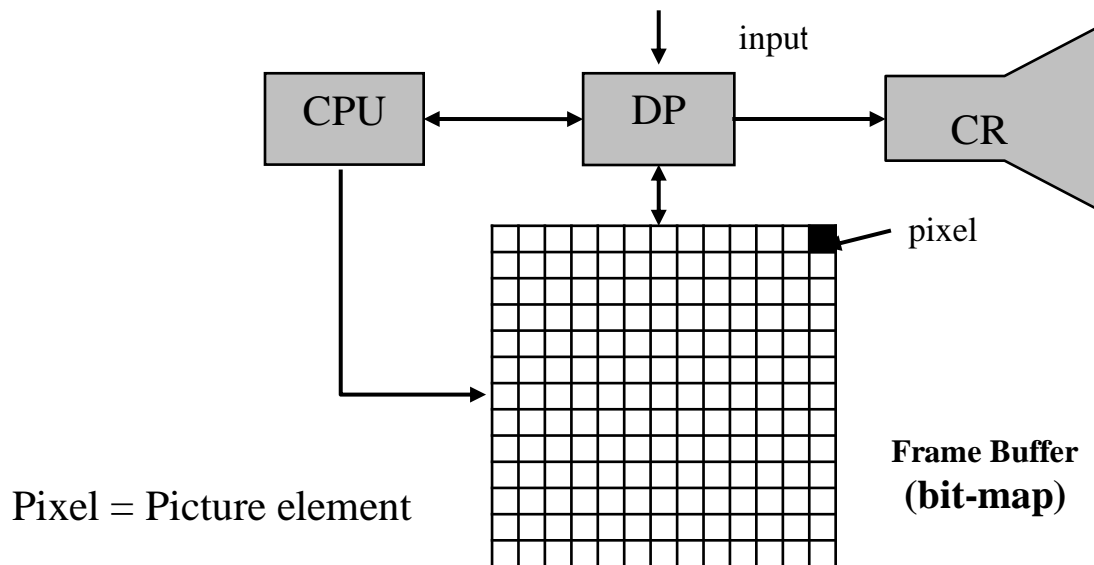


Figure: Raster Scan display system

- The stored intensity value is retrieved from frame buffer and painted on the scan line at a time. Home television are common examples using raster display
- Intensity range for pixel position depends on capability of raster system. For B/W system each point on screen are either on or off, so only one bit per pixel is needed to control the pixel intensity. To display color with varying intensity level, additional bits are needed. Up to 24 to 32 bit per pixel are included in high quality systems, which require more space of storage for the frame buffer, depending upon the resolution of the system.
- A system with 24 bit pixel and screen resolution 1024×1024 require 3 megabyte of storage in frame buffer.
 $1024 * 1024 \text{ pixel} = 1024 * 1024 * 24 \text{ bits} = 3 \text{ MB}$
- The frame butter in B/W system stores a pixel with one bit per pixel so it is termed as bitmap. The frame buffer in multi bit per pixel storage, is called pixmap.
- Refreshing on Raster-Scan display is carried out at the rate of 60 or higher frames per second. 60 frames per second is also termed as 60 cycle per second usually used unit Hertz (HZ)
- Returning of electron beam from right end to deft end after refreshing each scan line is **horizontal retrace** . At the end of each frame, the electron beam returns to the top left corner to begin next frame called **vertical retrace**.

Interlaced: Display in two pass with interlacing.

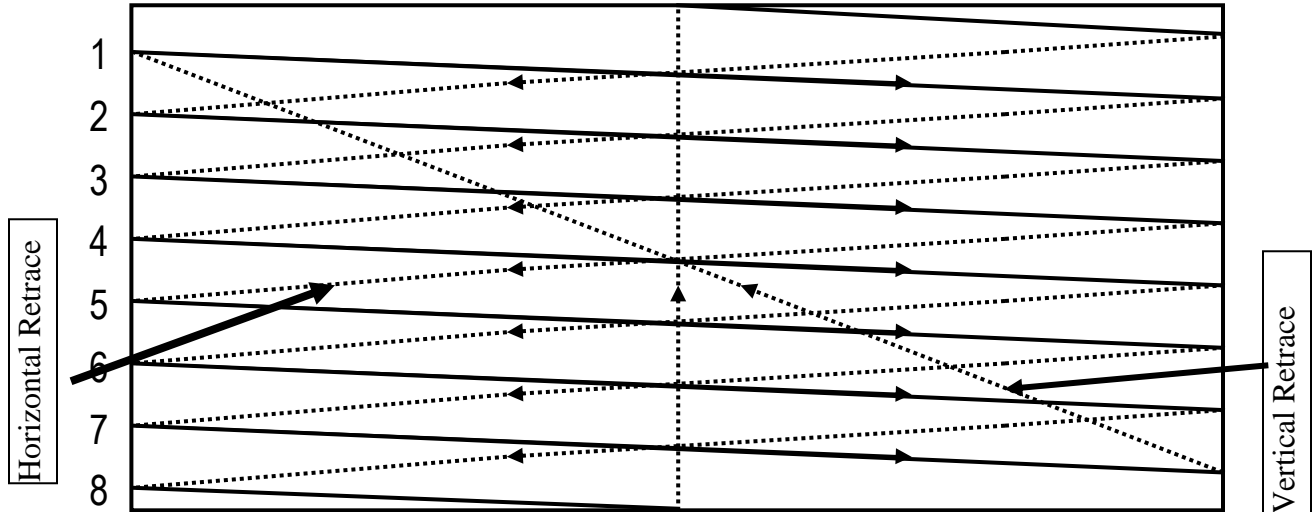


Figure: Horizontal retrace and Vertical retrace

Question: Consider a RGB raster system is to be designed using 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?

Solution: Size of screen = 8 inch \times 10 inch.

Pixel per inch(Resolution) = 100.

Then, Total no of pixels = $8 \times 100 \times 10 \times 100$ pixels

Bit per pixel storage = 6

Therefore Total storage required in frame buffer = $(800 \times 1000 \times 6)$ bits
= $(800 \times 1000 \times 6) / 8$ Bytes
= 600000 Bytes.

Frame Buffer Architecture of Raster Display

1. Indexed-color frame buffer.

In indexed –color frame buffer,

- Each pixel uses one byte in frame buffer.
- Each byte is an index into a color map.
- Each pixel may be one of 2^{24} colors, but only 256 color can be displayed at a time.
- There is a look-up table which has as many entries as there are pixel values.
- The table entry value is used to control the intensity or color of the CRT.

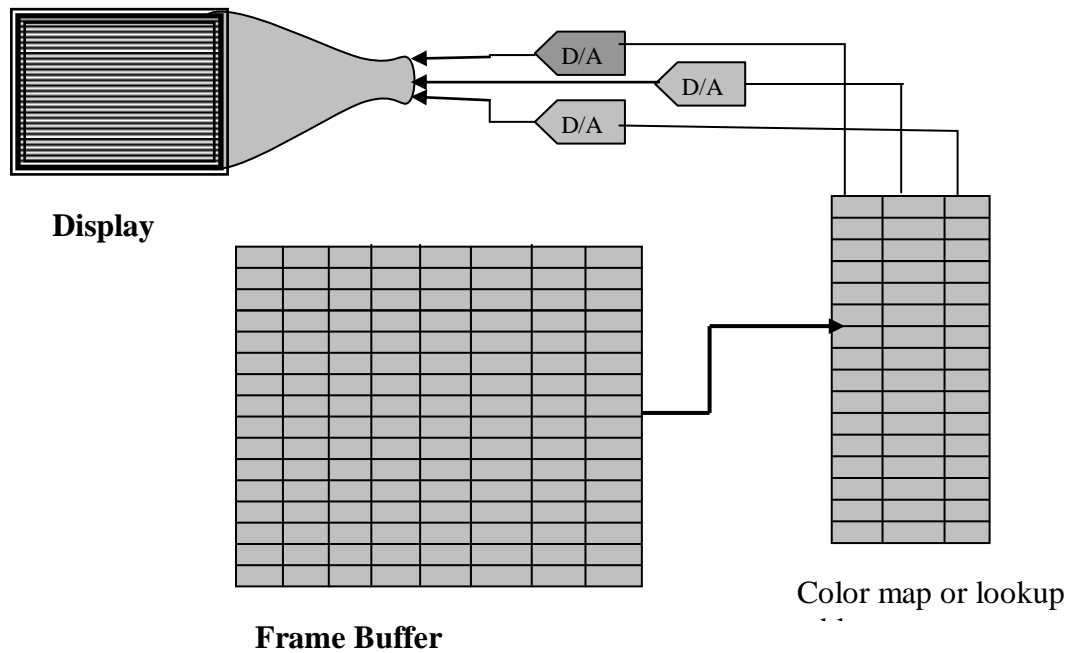


Figure: Indexed color frame buffer

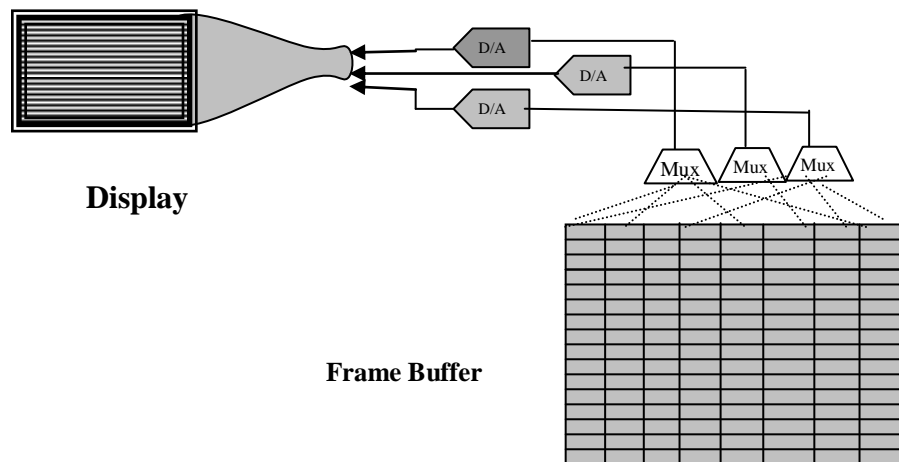


Figure : True Color Frame Buffer

2. **True-color frame buffer:** (24 bit or above): In true color frame buffer,
 - Each pixel requires at least 3-bytes, one for each primary color (R,G,B)
 - Sometimes combined with a look-up table per primary.

- Each pixel can be one of 2^{24} colors.

3 High-color frame buffer

- Popular PC/SVGA standard
- Pixels are packed in a short i.e. each primary color use 5 bit.
- Each pixel can be one of 2^{15} colors

Red	Green	Blue
-----	-------	------

Random scan display: (Vector display)

In random scan system, the CRT has the electron beam that is directed only to the parts of the screen where the picture is to be drawn. It draws a picture one line at a time, so it is also called **vector display** (or stroke writing or calligraphic display). The component lines of a picture are drawn and refreshed by random scan system in any specified order.

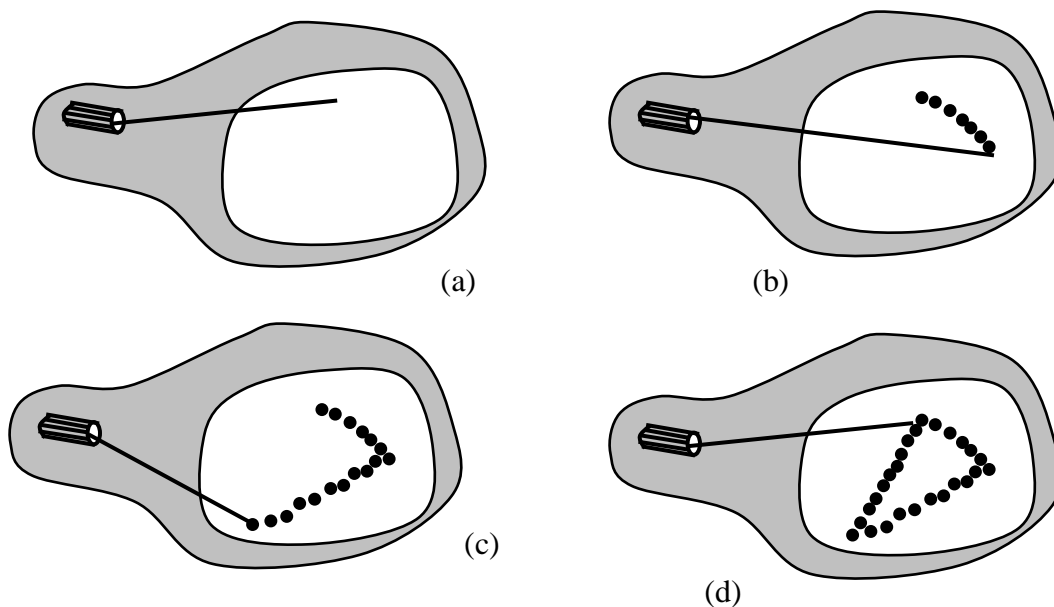


Figure: Random Scan Display

- The refresh rate of vector display depends upon the no of lines to be displayed for any image. Picture definition is stored as a set of line drawing instructions in an area of memory called the refresh display file (Display list or display file)
- To display a picture, the system cycles through the set of commands (line drawing) in the display file. After all commands have been processed, the system cycles back to the first line command in the list.
- Random scan systems are designed for drawing all component lines 30 to 60 times per second. Such systems are designed for line-drawing applications and can not display realistic shaded scenes. Since CRT beam directly follows the line path, the vector display system produce smooth line.

Color CRT

In color CRT, the phosphor on the face of CRT screen are laid into different fashion. Depending on the technology of CRT there are two methods for displaying the color pictures into the screen.

1. Beam penetration method

2. Shadow mask method

Beam Penetration method:

This method is commonly used for random scan display or vector display. In random scan display CRT, the two layers of phosphor usually red and green are coated on CRT screen. Display color depends upon how far electrons beam penetrate the phosphor layers.

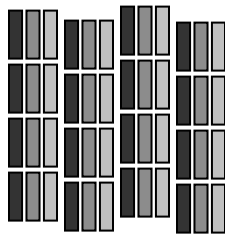
Slow electron excite only red layer so that we can see red color displayed on the screen pixel where the beam strikes. Fast electron beam excite green layer penetrating the red layer and we can see the green color displayed at the corresponding position. Intermediate is combination of red and green so two additional colors are possible – orange and yellow.

So only four colors are possible so no good quality picture in this type of display method.

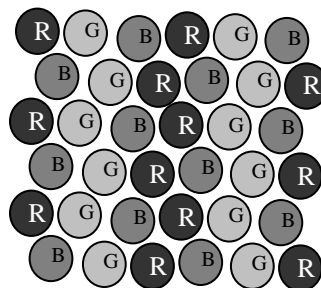
Shadow Mask Method:

Shadow mask method is used for raster scan system so they can produce wide range of colors. In shadow mask color CRT, the phosphor on the face of the screen are laid out in a precise geometric pattern. There are two primary variations.

1. The stripe pattern of inline tube
2. The delta pattern of delta tube



Stripe pattern



Delta Pattern

- In color CRT, the neck of tube, there are three electron guns, one for each red, green and blue colors. In phosphor coating there may be either strips one for each primary color, for a single pixel or there may be three dots one for each pixel in delta fashion.
- Special metal plate called a shadow mask is placed just behind the phosphor coating to cover front face.
- The mask is aligned so that it simultaneously allow each electron beam to see only the phosphor of its assigned color and block the phosphor of other two color.

Depending on the pattern of coating of phosphor, two types of raster scan color CRT are commonly used using shadow mask method.

1. Delta-Delta CRT:

- In delta-delta CRT, three electron beams one for each R,G,B colors are deflected and focused as a group onto shadow mask, which contains a series of holes aligned with the phosphor dots.
-

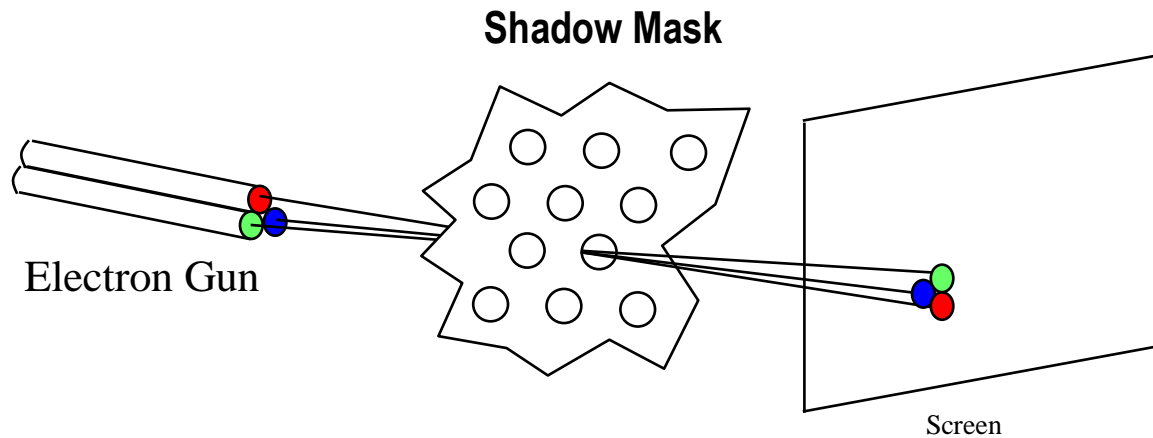


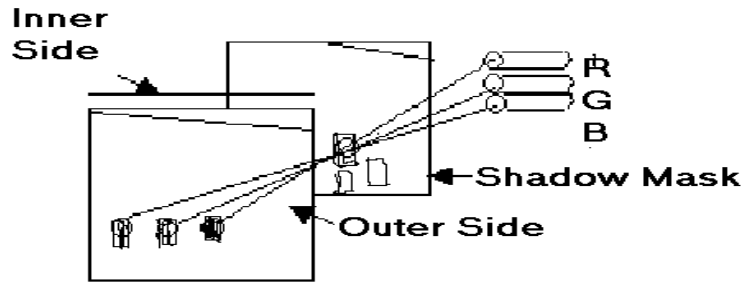
Figure: Shadow mask in Delta-Delta CRT

- Inner side of viewing has several groups of closely spaced red ,green and blue phosphor dot called triad in delta fashion.
- Thin metal plate adjusted with many holes near to inner surface called shadow mask which is mounted in such a way that each hole aligned with respective triad.
- Triad are so small that is perceived as a mixture of colors. When three beams pass through a hole in shadow mask, they activate the dot triangle to illuminate an small spot colored on the screen.
- The color variation in shadow mask CRT can be obtained by varying the intensity level of the three electron guns.

The main draw back of this CRT is due to difficulty for the alignment of shadow mask hole and respective triads.

A precision inline CRT:

This CRT uses strips pattern instead of delta pattern. Three strips one for each R, G, B color are used for a single pixel along a scan line so called inline. This eliminates the drawbacks of delta-delta CRT at the cost of slight reduction of image sharpness at the edge of the tube.



- Normally 1000 scan lines are displayed in this method. Three beams simultaneously expose three inline phosphor dots along scan line.

Architecture of Raster Scan System:

The raster graphics systems typically consists of several processing units. CPU is the main processing unit of computer systems. Besides CPU, graphics system consists of a special purpose processor called video controller or display processor. The display processor controls the operation of the display device.

The organization of raster system is as shown below

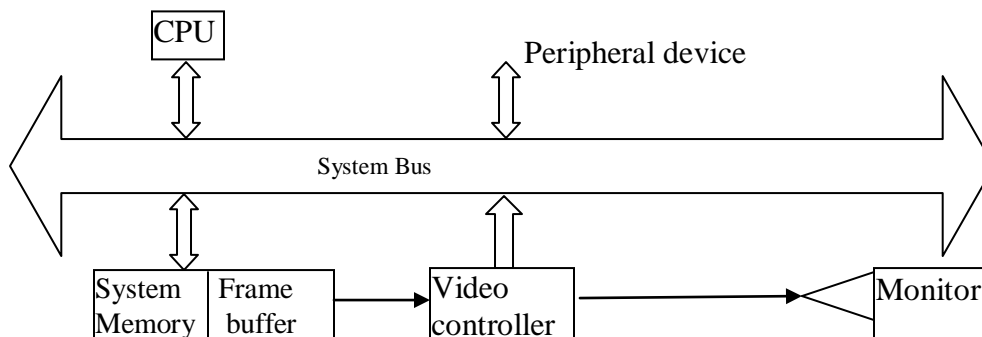


Figure: A simple Raster System.

- A fixed area of system memory is reserved for the frame buffer. The video controller has the direct access to the frame buffer for refreshing the screen.
- The video controller cycles through the frame buffer, one scan line at a time, typically at 60 times per second or higher. The contents of frame buffer are used to control the CRT beam's intensity or color.

The video controller:

The video controller is organized as in figure below. The raster-scan generator produces deflection signals that generate the raster scan and also controls the X and Y address registers, which in turn defines memory location to be accessed next. Assume that the frame buffer is addressed in X from 0 to X_{\max} and in Y from 0 to Y_{\max} then, at the start of each refresh cycle, X address register is set to 0 and Y register is set to 0

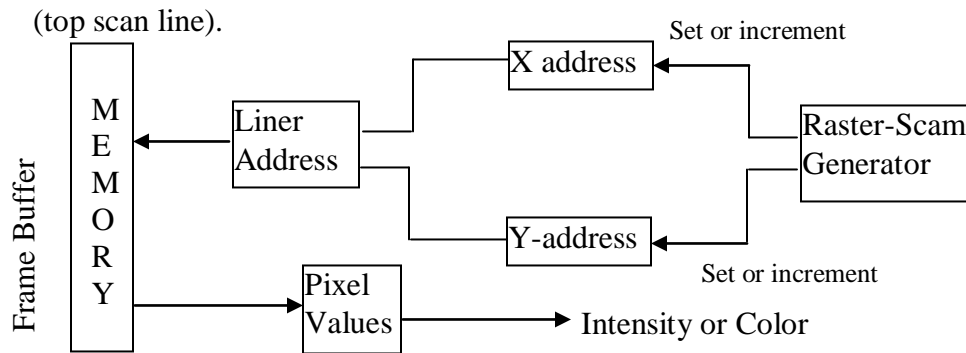


Figure: Organization of Video Controller.

As first scan line is generated, the X address is incremented up to X_{\max} . Each pixel value is fetched and used to control the intensity of CRT beam. After first scan line X address is reset to 0 and Y address is incremented by 1. The process is continued until the last scan line ($Y=Y_{\max}$) is generated.

Raster-Scan Display Processor:

The raster scan with a peripheral display processor is a common architecture that avoids the disadvantage of simple raster scan system. It includes a separate graphics processor to perform graphics functions such as scan conversion and raster operation and a separate frame buffer for image refresh.

The display processor has its own separate memory called display processor memory.

- System memory holds data and those programs that execute on the CPU, and the application program, graphics packages and OS.
- The display processor memory holds data plus the program that perform scan conversion and raster operations.
- The frame buffer stores displayable image created by scan conversion and raster operations.

The organization is given below in figure

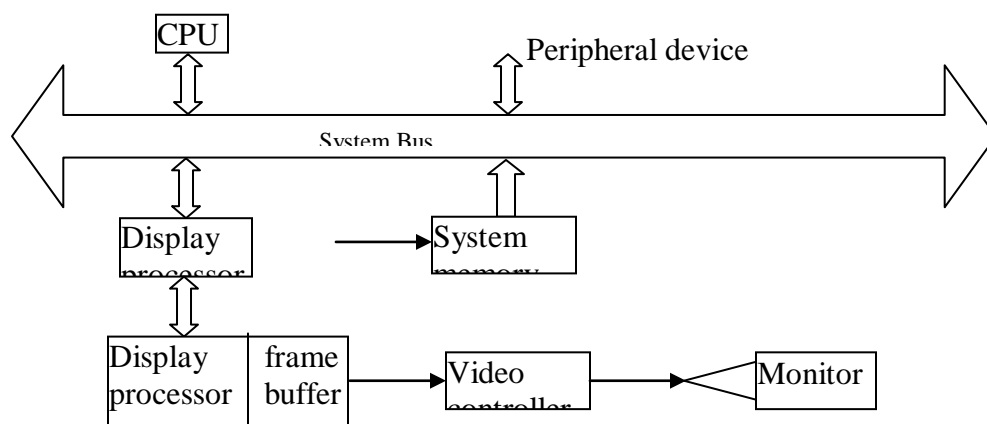


Figure: Architecture Raster scan system with display processor

2. Vector Display System.

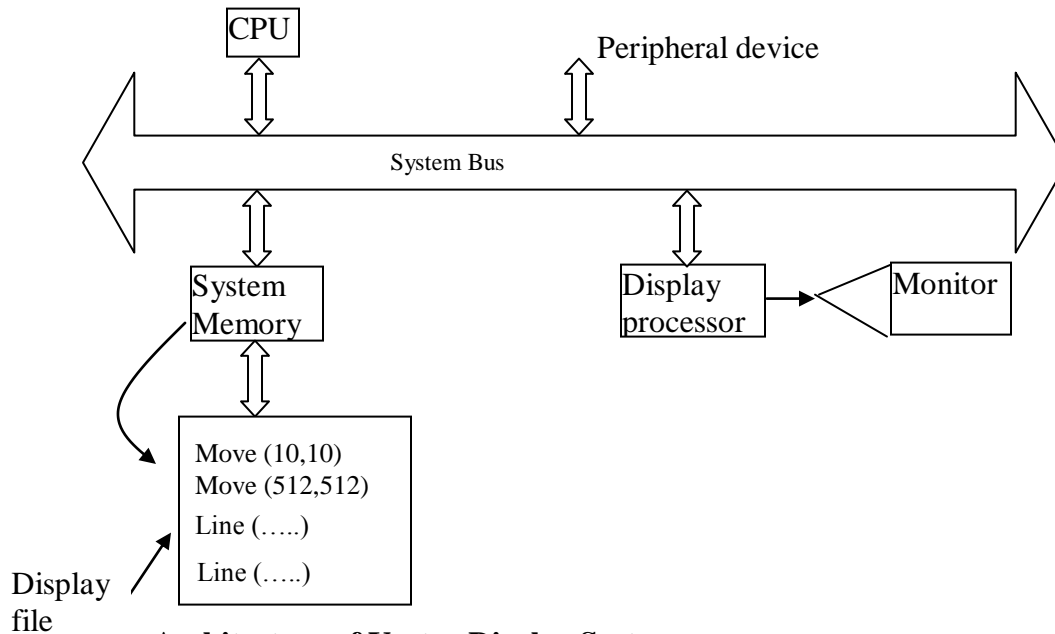


Figure : Architecture of Vector Display System

- Vector display system consists of several units along with peripheral devices. The display processor is also called as graphics controller.
- Graphics package creates a display list and stores in systems memory (consists of points and line drawing commands) called display list or display file.
- Refresh time around 30 cycle per second.
- Vector display technology is used in monochromatic or beam penetration color CRT.
- Graphics are drawn on a vector display system by directing the electron beam along component line.

Advantages:

- Can produce output with high resolutions.
- Better for animation than raster system since only end point information is needed.

Disadvantages:

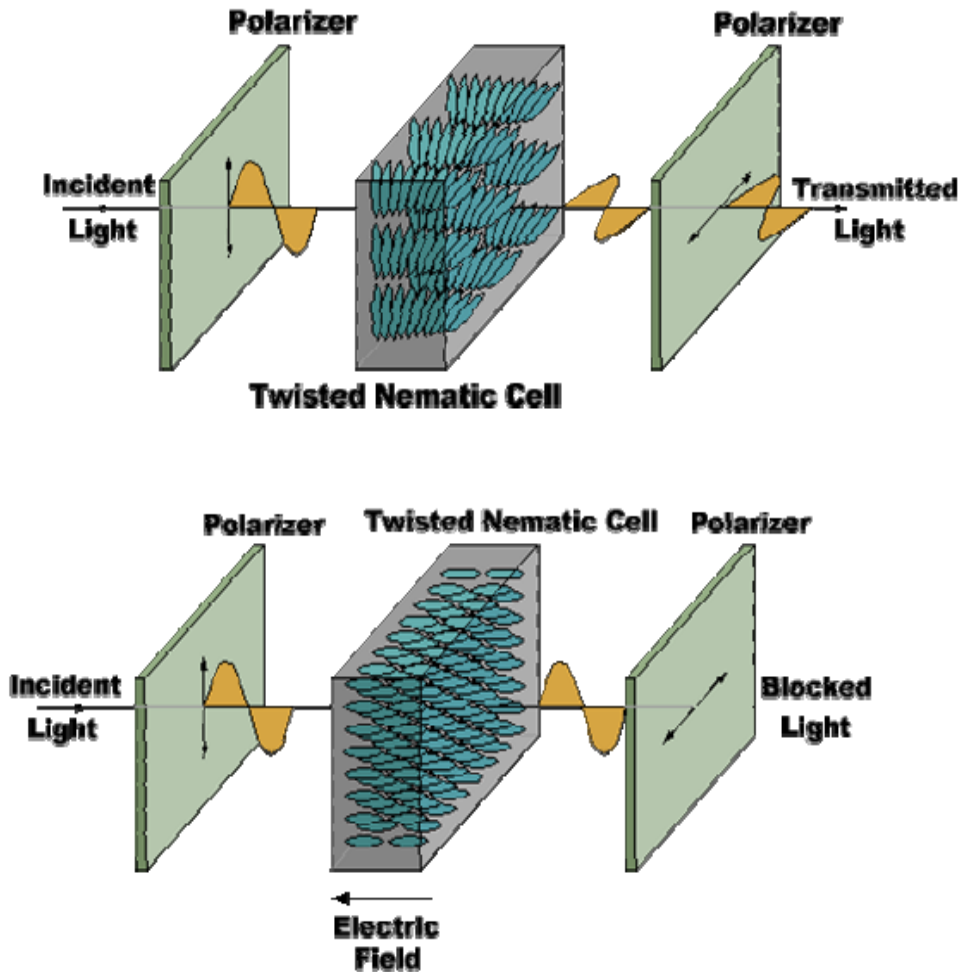
- Cannot fill area with pattern and manipulate bits.
- Refreshing image depends upon its complexity.

Flat Panel Display

- The term flat panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to CRT
- Thinner CRT, can able to hang on a wall or able to wear on wrist
- We can write in some flat panel displays
- Can be used in small TV monitor, calculators, pocket video games, laptop computers, armrest viewing of movies on airlines, as advertisement hoarding board etc.
- We can separate flat panel displays into two categories
 - Emissive Display
 - Non Emissive Display (LCD)
- Emissive Display
 - Device that convert electrical energy into light
 - Plasma panels, thin film electroluminescent display and LED are examples of emissive display.
 - Plasma panels, also called gas charge display, are constructed that usually includes neon.
 - A series of vertical conducting ribbons is placed on one glass panel, and a set of horizontal ribbons is built into other glass panel
 - Firing voltages applied to a part of horizontal and vertical conductors causes the gas at the intersection of the two conductors to break down into glowing plasma of electrons and ions
 - Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh pixel positions (at the intersection of conductors) 60 times per second.
- AC methods are used provide faster application of the firing voltages, and thus brighter displays
- Separation between pixels is provided by the electric field of the conductors.
- One disadvantage of plasma panels has been that they were strictly monochromatic devices, but systems have been developed that are now capable of displaying color or grayscale.
- Non-Emissive Display
 - Non emissive displays (or non emitters) use optical effects to convert sunlight or light from some other source into graphics patterns
 - The most important example of non emissive flat panel display is a Liquid Crystal Display
- **Liquid Crystal Display**
 - The most popular alternative to the CRT is the Liquid Crystal Display (LCD)

- LCDs are organic molecules that, in the absence of external forces, tend to align themselves in crystalline structures.
- When an external force is applied they will rearrange themselves as if they were a liquid.
- Some liquid crystals respond to heat (i.e. *mood rings*), others respond to electromagnetic forces.
- When LCDs are used as optical (light) modulators they are actually changing polarization rather than transparency (at least this is true for the most popular type of LCD called *Super-twisted Nematic Liquid crystals*)
- In their unexcited or crystalline state the LCDs rotate the polarization of light by 90 degrees.
- In the presence of an electric field, LCDs behave like a liquid and align the small electrostatic charges of the molecules with the impinging E field.
- Two glass plates, each containing a light polarizer at right angles to the other plate, sandwich the liquid crystal material
- Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate
The intersection of two conductors defines a pixel position.
- Picture definitions are stored in refresh buffer, and the screen is refreshed at the rate of 60 frames per second
- Back lighting is also commonly applied using solid state electronic devices, so that the system is no completely dependent on out light sources.
- Colors can be displayed by using different materials or dyes and by placing a triad of color pixels at each screen location.
- Another method of constructing LCDs is to place a transistor at each pixel location, using thin film transistor technology.
- The transistors are used to control the voltage at pixel locations and to prevent gradually leaking out of liquid crystal cells
- The LCD's transition between crystalline and liquid states is a slow process.
- This has both good and bad side effects. LCDs, like phosphors, remain "on" for some time after the E field is applied.
- Thus the image is *persistent* like a CRT's, but this lasts just until the crystals can realign themselves, thus they must be constantly refreshed, again, like a CRT.

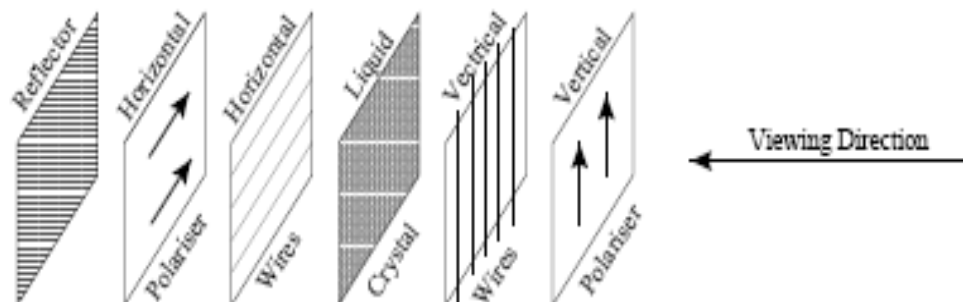
LCD Off and LCD On state



- LCD operation
 - The LCD's themselves have extremely low power requirements.
 - A very small electric field is required to excite the crystals into their liquid state.
 - Most of the energy used by an LCD display system is due to the back lighting.
 - LCD's slowly transition back to their crystalline state when the E field is removed.
 - In scanned displays, with a large number of pixels, the percentage of the time that LCDs are excited is very small.
 - Thus the crystals spend most of their time in intermediate states, being neither "On" or "Off".
 - This behavior is indicative of *passive displays*.
 - Notice that these displays are not very sharp and are prone to ghosting.

- Another way to building LCD displays uses an *active matrix*.
- The main difference is that the electric field is retained by a capacitor so that the crystal remains in a constant state.
- Transistor switches are used to transfer charge into the capacitors during the scanning process.
- The capacitors can hold the charge for significantly longer than the refresh period yielding a crisp display with no shadows.
- Active displays, require a working capacitor and transistor for each LCD or pixel element, and thus, they are more expensive to produce.
- LCDs are commonly used in small systems, such as calculators, and portable, laptop computers. These non emissive devices produce a picture by passing polarized light from the surroundings or from the internal light source through liquid-crystal material that can be aligned to either block or transmit the light.
- Liquid crystals can change their transmission of polarised light by applying an electrical potential.
- Liquid crystal refers to the fact that these compounds have a crystalline arrangement of molecules, yet they flow like a liquid. Flat panel displays commonly use nematic (threadlike) liquid crystal compounds that tend to keep the long axes of the rod-shaped molecules aligned.
- LCDs must be refreshed when the older wire technology is used.
- Unlike CRTs, liquid crystals have a rather longer persistence of several hundred milliseconds.
- Most modern LCDs use active matrix panels, where Thin Film Transistor (TFT) technology is used to create tiny transistors at each pixel location.
- TFT displays have traditionally been expensive, because each pixel actually requires three transistor switches.
- LCD displays have a native resolution.
- Displaying different resolutions requires anti-aliasing.
- Alternative display devices have been invented e.g. plasma panels, and projector devices similar issues apply.

LCDs



LED display Technology

- LED stands for Light emitting diode and is the one of the latest technology used in display devices.

- Here organic molecules are placed between two parallel plates. When a voltage is applied b/w these plates, the organic molecule takes a shape and is displayed.
- LCD chips and pixels do not produce their own light. In order to produce a visible image on a screen, the LCD's pixels have to be "backlit". Cold cathode fluorescent lamp (CCFL) is used as backlighting in LCD display.
- LCD display whose backlighting has been replaced by organic light emitting diode (OLED) is known as LED display.
- There are two main ways that LED backlighting is applied to LCD display.
- Edge Lighting
- Full Array Lighting

In Edge lighting, LEDs are placed around the edge of the screen, then the light is dispersed across the screen.

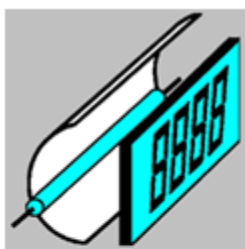
- The main advantages of this method is that the display can be made very thin.
- Whereas in case of full array, LEDs are placed behind the entire surface of the screen.
- The main advantages of this method is that single LED or specific group of LEDs can be switched on or off independently within certain area.

Thus providing a more control of brightness and darkness over each area.

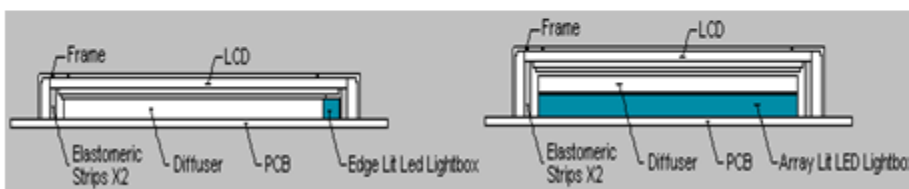
LED/LCD vs Standard LCD

- Lower power consumption.
- No Mercury used as in some other LCD backlight systems.
- More balanced color saturation.
- In LED/LCD TVs using the Full Array backlight method, there is little or no light leakage in dark scenes.

Different types of LCD backlighting



CCFL backlighting



LED backlighting

Video Cards and CRT Monitors

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

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

- Video Display StandardsVideo Electronics Standards Association (VESA), which consists of video card and monitor manufacturers, develops video stands to define the resolution, number of colors, and other display properties.
 - Monochrome Display Adapter (MDA)
 - Hercules Graphics Card
 - Color Graphics Adapter (CGA)
 - Enhanced Graphics Adapter (EGA)
 - Video Graphics Adapter (VGA)
 - Super VGA (SVGA) and Other Standards Beyond VGA
 -

Line Drawing Algorithms.

The slope-intercept equation of a straight line is:

$$y = mx + b \quad \text{where } m = \text{slope of line and } b = \text{y-intercept.}$$

for any two given points (x_1, y_1) and (x_2, y_2)

$$\text{slope (m)} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\therefore b = y - \frac{y_2 - y_1}{x_2 - x_1} x \quad \text{from above equation i.e. } y = mx + b$$

At any point (x_k, y_k)

$$y_k = mx_k + b \quad \dots\dots\dots 1$$

At (x_{k+1}, y_{k+1}) ,

$$y_{k+1} = mx_{k+1} + b \quad \dots\dots\dots 2.$$

subtracting 1 from 2 we get-

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

Here $(y_{k+1} - y_k)$ is increment in y as corresponding increment in x.

$$\therefore \Delta y = m \Delta x$$

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

For incremental algorithm in line drawing ,

- Increment x by 1
- Computer corresponding y and display pixel at position $(x_i, \text{round}(y_i))$

Problem: Floating point multiplication & addition

- The round function.

DDA line Algorithm:

The digital differential analyzer (DDA) is a scan conversion line drawing algorithm based on calculating either Δx or Δy from the equation,

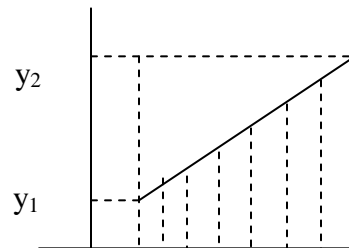
$$\Delta y = m \Delta x.$$

We sample the line at unit intervals in one co-ordinate and determine the corresponding integer values nearest to the line path for the other co-ordinates.

Consider a line with positive slope.

If $m \leq 1$, we sample x co-ordinate. So $\Delta x = 1$ and compute each successive y value as:

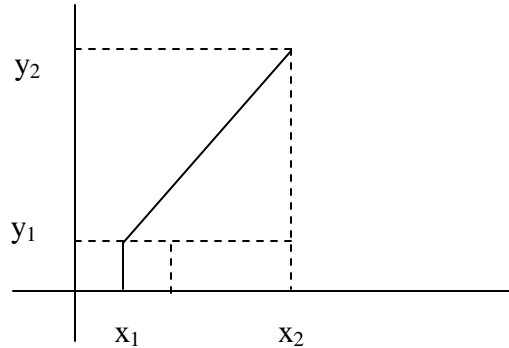
$$y_{k+1} = y_k + m \quad \because m = \frac{\Delta y}{\Delta x}, \Delta x = 1$$



Here k takes value from starting point and increase by 1 until x_1 and point. m x_2
any real value between 0 and 1. $m \leq 1$

For line with positive slope greater than 1,
we sample $\Delta y = 1$ and calculate
corresponding x values as

$$x_{k+1} = x_k + \frac{1}{m} \quad \because m = \frac{\Delta y}{\Delta x}, \Delta y = 1$$



$m > 1$

The above equations are under the assumption that the lines are processed from left to right. i.e. left end point is starting. If the processing is from right to left, we can sample $\Delta y = -1$ for line $|m| < 1$

$$\therefore y_{k+1} = y_k - m,$$

If $|m| > 1$, $\Delta y = -1$ and calculate

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

The complete C function for DDA algorithm is.

```
void lineDDA (int x1, int y1, int x2, int y2)
{
    int dx, dy, steps, k;
    float incrx; incry; x,y;
    dx=x2-x1;
    dy=y2-y1;
    if (abs(dx)>abs(dy))
        steps=abs(dx);
    else
        steps=abs(dy);
    incrx=dx/steps;
    incry=dy/steps;
    x=x1;    /* first point to plot */
    y=y1;
    putpixel(round(x), round(y),1); //1 is used for color
    for (k=1;k<=steps;k++)
    {
        x = x + incrx;
        y = y + incry;
        putpixel(round(x), round(y),1);
    }
}
```

```

    }
}

```

The DDA algorithm is faster method for calculating pixel position but it has problems:

- m is stored in floating point number.
- round of error
- error accumulates when we proceed line.
- so line will move away from actual line path for long line

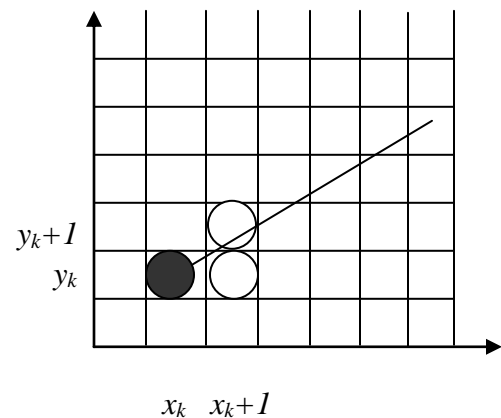
Bresenham's Line algorithm:

An accurate and efficient line generating algorithm, developed by Bresenham that scan converts lines only using integer calculation to find the next (x,y) position to plot. It avoids incremental error accumulation.

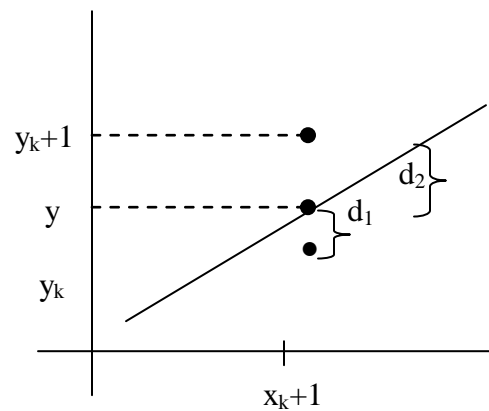
Line with positive slope less than 1 ($0 < m < 1$)

Pixel position along the line path are determined by sampling at unit x intervals. Starting from left end point, we step to each successive column and plot the pixel closest to line path.

Assume that (x_k, y_k) is pixel at k^{th} step then next point to plot may be either $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$



At sampling position $x_k + 1$, we label vertical pixel separation from line path as d_1 & d_2 as in figure . The y -coordinate on the mathematical line path at pixel column $x_k + 1$ is $y = m(x_k + 1) + b$



Then $d_1 = y - y_k$

$$= m(x_k + 1) + b - y_k$$

$$d_2 = (y_k + 1) - y$$

$$= (y_k + 1) - m(x_k + 1) - b$$

$$\text{Now } d_1 - d_2 = 2m(x_k + 1) - (y_k + 1) - y_k + 2b$$

$$= 2m(x_k + 1) - 2y_k + 2b - 1$$

A decision parameter p_k for the k^{th} step in the line algorithm can be obtained by substituting $m = \frac{\Delta y}{\Delta x}$ in above eqⁿ and defining

$$\begin{aligned} p_k &= \Delta x(d_1 - d_2) \\ &= \Delta x[2 \frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1] \\ &= 2\Delta y.x_k - 2\Delta x.y_k + 2\Delta y + \Delta x(2b - 1) \\ &= 2\Delta y.x_k - 2\Delta x.y_k + c \end{aligned}$$

Where the constant $c = 2\Delta y - \Delta x(2b - 1)$ which is independent.

If decision parameter p_k is negative i.e. $d_1 < d_2$, pixel at y_k is closer to the line path than pixel at $y_k + 1$. In this case we plot lower pixel. ($x_k + 1, y_k$). *other wise plot upper pixel ($x_k + 1, y_k + 1$).*

Co-ordinate change along the line occur in unit steps in either x, or y direction. Therefore we can obtain the values of successive decision parameters using incremental integer calculations.

At step $k+1$, p_{k+1} is evaluated as.

$$\begin{aligned} p_{k+1} &= 2\Delta y.x_{k+1} - 2\Delta x.y_{k+1} + c \\ \therefore p_{k+1} - p_k &= 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k) \end{aligned}$$

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

$$\therefore p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

The term $y_{k+1} - y_k$ is either 0 or 1 depending upon the sign of p_k .

The first decision parameter p_0 is evaluated as.

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

and successively we can calculate decision parameter as

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

so if p_k is negative $y_{k+1} = y_k$ so $p_{k+1} = p_k + 2\Delta y$

otherwise $y_{k+1} = y_k + 1$, then $p_{k+1} = p_k + 2\Delta y - 2\Delta x$

Algorithm:

1. Input the two line endpoint and store the left endpoint at (x_o, y_o)
2. Load (x_o, y_o) in to frame buffer, i.e. Plot the first point.
3. Calculate constants $2\Delta x, 2\Delta y$ calculating $\Delta x, \Delta y$ and obtain first decision parameter value as

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

4. At each x_k along the line, starting at $k=0$, perform the following test,

if $p_k < 0$, next point is $(x_k + 1, y_k)$

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

otherwise

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

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

5. Repeat step 4 Δx times.

Function implementation in C

```
void lineBresenham (int x1, int y1, int x2, int y2)
{
    int x, y, dx, dy, pk, k xEnd;
    dx=abs (x2-x1);
    dy=abs (y2-y1);
    if (x1>x2)
    {
        x = x2;
        y = y2;
        xEnd = x1;
    }
    else
    {
        x = x1;
        y = y1;
        xEnd = x2;
    }
    putpixel (x,y,1);
    pk=2*dy-dx;
    while (x<=xEnd)
    {
        if (pk<0)
        {
            x=x+1;
            y=y;
            pk=pk+2*dy;
        }
        else
        {
            x=x+1;
            y=y+1;
            pk= pk+2*dy-2*dx
        }
        putpixel (x,y,1);
    }
}
```

Brasenheim^s algorithm is generalized to lines with arbitrary slope by considering the symmetry between the various octants & quadrants of xy plane.

For a line positive slope greater than 1, we simply interchange the role of x & y.

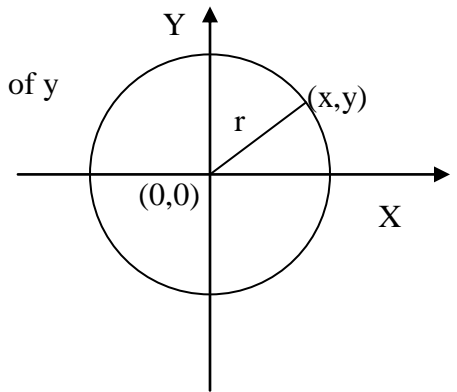
Algorithm for Circle

Simple Algorithm:

The equation of circle centered at origin and radius r is given by $x^2 + y^2 = r^2$

$$\Rightarrow y = \pm\sqrt{r^2 - x^2}$$

- Increment x in unit steps and determine corresponding value of y from the equation above. Then set pixel at position (x,y).
- The steps are taken from $-r$ to $+r$.
- In computer graphics, we take origin at upper left corner point on the display screen i.e. first pixel of the screen so any visible circle drawn would be centered at point other than (0,0). If center of circle is (xc,yc) then the calculated point from origin center should be moved to pixel position by (x+xc, y+yc).



In general the equation of circle centered at (xc,yc) and radius r is

$$(x - xc)^2 + (y - yc)^2 = r^2$$

$$\Rightarrow y = yc \pm \sqrt{r^2 - (x - xc)^2} \dots\dots\dots(1)$$

Use this equation to calculate the position of points on the circle. Take unit step from xc-r to xc+r for x value and calculate the corresponding value of y- position for pixel position (x,y). This algorithm is simple but,

- Time consuming – square root and squares computations
- Non – uniform spacing , due to changing slope of curve. If non-uniform spacing is avoided by interchanging x and y for slope $|m| > 1$, this leads to more computation.

Following program demonstrates the simple computation of circle using the above equation (1)

```
//program for circle
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#define SQUARE(x) ((x)*(x))
void drawcircle(int ,int,int);
void main()
{
    int gd,gm,err;
    int xc,yc,r;
    gd=DETECT;
    initgraph(&gd,&gm,"\\tc\\bgi");
    err=graphresult();
```

```

    if(err!=0)
    {
        printf("ERROR:%s",grapherrormsg(err));
        printf("\nPress a key..");
        getch();
        exit(1);
    }
    xc=getmaxx()/2;
    yc=getmaxy()/2;
    r=50;
    drawcircle(xc,yc,r);
    getch();
    closegraph();
} //end main
void drawcircle(int xc,int yc,int r)
{
    int i,x,y,y1;
    for(i=xc-r;i<=xc+r;i++)
    {
        x=i;
        y=yc+sqrt(SQUARE(r)-SQUARE(x-xc));
        y1=yc-sqrt(SQUARE(r)-SQUARE(x-xc));
        putpixel(x,y,1);
        putpixel(x,y1,1);
    }
}

```

Drawing circle using polar equations

If (x,y) be any point on the circle boundary with center (0,0) and radius r, then

$$x = r \cos \theta$$

$$y = r \sin \theta$$

i.e. $(x, y) = (r \cos \theta, r \sin \theta)$

To draw circle using these co-ordinates approach, just increment angle starting from 0 to 360. Compute (x,y) position corresponding to increment angle. Which draws circle centered at origin, but the circle centered at origin is not visible completely on the screen since (0,0) is the starting pixel of the screen. If center of circle is given by (xc,yc) then the pixel position (x,y) on the circle path will be computed as

$$x = xc + r \cos \theta$$

$$y = yc + r \sin \theta$$

Circle Function to draw circle using the polar transformation:

```

void drawcircle(int xc,int yc,int r)
{
    int x,y;
    float theta;
    const float PI=3.14;

    for(theta=0.0;theta<=360;theta+=1)
    {

```

```

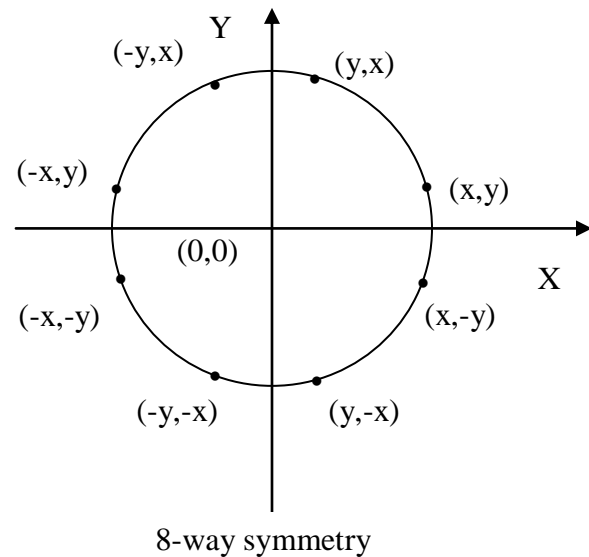
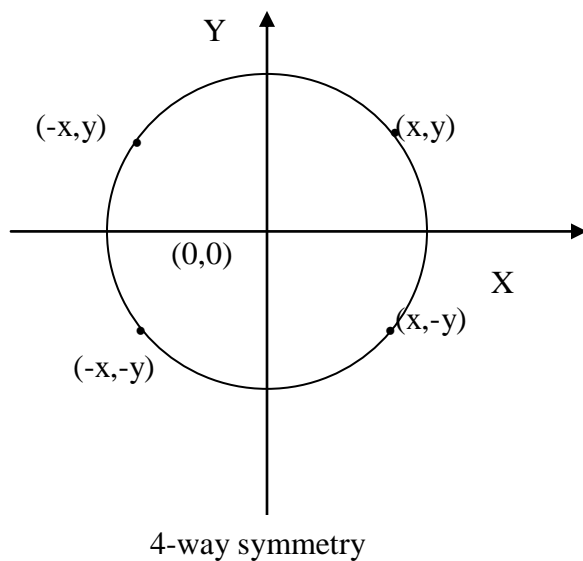
        x= xc+r*cos(theta*PI/180.0);
        y= yc+r*sin(theta*PI/180.0);
        putpixel(x,y,1);

    }
}

```

Symmetry in circle scan conversion:

We can reduce the time required for circle generation by using the symmetries in a circle e.g. 4-way or 8-way symmetry. So we only need to generate the points for one quadrant or octants and then use the symmetry to determine all the other points.



Problem of computation still persists using symmetry since there are square roots, trigonometric functions are still not eliminated in above algorithms.

Mid point circle Algorithm:

In mid point circle algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step.

For a given radius r , and screen center position (x_c, y_c) , we can set up our algorithm to calculate pixel positions around a circle path centered at $(0,0)$ and then each calculated pixel position (x, y) is moved to its proper position by adding x_c to x and y_c to y .

i.e. $x = x + x_c$, $y = y + y_c$.

To apply the mid point method, we define a circle function as:

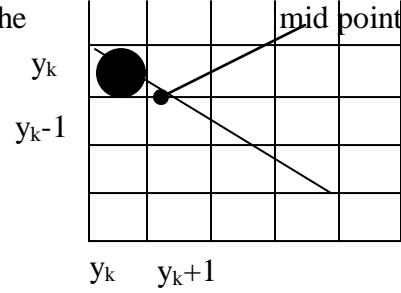
$$f_{circle} = x^2 + y^2 - r^2$$

To summarize the relative position of point (x, y) by checking sign of f_{circle} function,

$$f_{circle}(x, y) \begin{cases} < 0, \text{ if } (x, y) \text{ lies inside the circle boundary} \\ = 0, \text{ if } (x, y) \text{ lies on the circle boundary} \\ > 0, \text{ if } (x, y) \text{ lies outside the circle boundary.} \end{cases}$$

The circle function tests are performed for the mid positions between pixels near the circle path at each sampling step. Thus the circle function is decision parameter in mid point algorithm.

The figure, shows the mid point between the two candidate pixel at sampling position $x_k + 1$, Assuming we have just plotted the pixel (x_k, y_k) , we next need to determine whether the pixel at position $(x_k + 1, y_k)$ or $(x_k + 1, y_k - 1)$ is closer to the circle.



Our decision parameter is circle function evaluated at the mid point

$$p_k = f_{circle}(x_k + 1, y_k - \frac{1}{2})$$

$$= (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 = (x_k + 1)^2 + y_k^2 - y_k + \frac{1}{4} - r^2$$

If $p_k < 0$, then mid-point lies inside the circle, so point at y_k is closer to boundary otherwise, $y_k - 1$ closer to choose next pixel position.

Successive decision parameters are obtained by incremental calculation. The decision parameter for next position is calculated by evaluating circle function at sampling position $x_{k+1} + 1$ i.e. $x_k + 2$ as

$$p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$

$$= \{(x_{k+1} + 1)\}^2 + (y_{k+1} - \frac{1}{2})^2 - r^2$$

$$= (x_{k+1})^2 + 2x_{k+1} + 1 + (y_{k+1})^2 - (y_{k+1}) + \frac{1}{4} - r^2$$

$$\text{Now, } p_{k+1} - p_k = 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

$$\text{i.e. } p_{k+1} = p_k + 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Where y_{k+1} is either y_k or $y_k - 1$ depending upon sign of p_k . and $x_{k+1} = x_k + 1$

If p_k is negative, $y_{k+1} = y_k$ so we get,

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

If p_k is positive, $y_{k+1} = y_k - 1$ so we get,

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

$$\text{Where } 2x_{k+1} = 2x_k + 2$$

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

At the start position, (0,r), these two terms have the values 0 and 2r, respectively. Each successive values are obtained by adding 2 to the previous value of 2x and subtracting 2 from previous value of 2y.

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

$$\begin{aligned} p_0 &= f_{\text{circle}}(1, r - \frac{1}{2}) \\ &= 1 + (r - \frac{1}{2})^2 - r^2 \\ &= 1 + r^2 - r + \frac{1}{4} - r^2 \\ &= \frac{5}{4} - r \end{aligned}$$

If p_0 is specified in integer,

$$p_0 = 1 - r.$$

The Algorithm:

1. Input radius r and circle centre (x_c, y_c) , and obtain the first point on circle centered at origin as.

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

2. Calculate initial decision parameter

$$p_0 = \frac{5}{4} - r$$

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

If $p_k < 0$ next point along the circle centre at (0,0) is $(x_k + 1, y_k)$

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Otherwise, the next point along circle is $(x_k + 1, y_k - 1)$

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

$$\text{Where } 2x_{k+1} = 2x_k + 2. \quad \text{and} \quad 2y_{k+1} = 2y_k - 2.$$

4. Determine symmetry point on the other seven octants.
5. Move each calculated pixels positions (x, y) in to circle path centered at (x_c, y_c) as

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

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

Program for mid point circle algorithm in C

//mid point circle algorithm

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
void drawpoints(int,int,int,int);
void drawcircle(int,int,int);

void main(void)
{
    /* request auto detection */
    int gdriver = DETECT, gmode, errorcode;
    int xc,yc,r;
    /* initialize graphics and local
       variables */
    initgraph(&gdriver, &gmode, "\\tc\\bgi");

    /* read result of initialization */
    errorcode = graphresult();
    if (errorcode != grOk) /* an error
        occurred */
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1); /* terminate with an error
            code */
    }
    printf("Enter the center co-ordinates:");
    scanf("%d%d",&xc,&yc);
    printf("Enter the radius");
    scanf("%d",&r);
    drawcircle(xc,yc,r);
    getch();
    closegraph();
}

void drawpoints(int x,int y, int xc,int yc)
{
    putpixel(xc+x,yc+y,1);
    putpixel(xc-x,yc+y,1);
    putpixel(xc+x,yc-y,1);
    putpixel(xc-x,yc-y,1);
    putpixel(xc+y,yc+x,1);
    putpixel(xc-y,yc+x,1);
    putpixel(xc+y,yc-x,1);
    putpixel(xc-y,yc-x,1);
}

void drawcircle(int xc,int yc,int r)
{
    int p,x,y;
    x=0;
    y=r;
```

```

drawpoints(x,y,xc,yc);
p=1-r;
while(x<y)
{
    if(p<0)
    {
        x=x+1;
        p=p+2*x+1;
    }
    else
    {
        x=x+1;
        y=y-1;
        p=p+2*(x-y)+1;
    }
    drawpoints(x,y,xc,yc);
}
}

```

Ellipse Algorithm

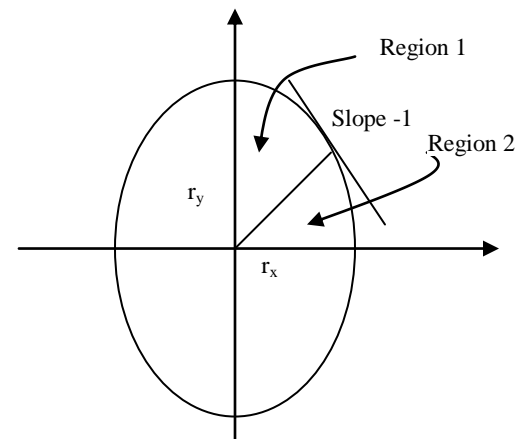
The basic algorithm for drawing ellipse is same as circle computing x and y position at the boundary of the ellipse from the equation of ellipse directly.

We have equation of ellipse centered at origin (0,0) is

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

$$y = \pm \frac{r_y}{r_x} \sqrt{(r_x^2 - x^2)} \dots\dots\dots(1)$$

Stepping unit interval in x direction from $-r_x$ to r_x we can get corresponding y value at each x position which gives the ellipse boundary co-ordinates. Plotting these computed points we can get the ellipse.



Figure

If center of ellipse is any arbitrary point (xc,yc) then the equation of ellipse can be written as

$$\frac{(x-xc)^2}{r_x^2} + \frac{(y-yc)^2}{r_y^2} = 1$$

$$\text{i.e. } y = yc \pm \frac{r_y}{r_x} \sqrt{r_x^2 - (x - xc)^2} \text{ -----(2)}$$

for any point (x,y) on the boundary of the ellipse. If major axis of ellipse with major axis along X-axis, the algorithm based on the direct computation of ellipse boundary points can be summarized as

1. Input the center of ellipse (xc,yc), x-radius xr and y-radius yr.
2. For each x position starting from xc-xr and stepping unit interval along x-direction, compute corresponding y positions as

$$y = yc \pm \frac{r_y}{r_x} \sqrt{r_x^2 - (x - xc)^2}$$

3. Plot the point (x, y).
4. Repeat step 2 to 3 until $x \geq xc + xr$

Computation of ellipse using polar co-ordinates:

Using the polar co-ordinates for ellipse, we can compute the (x,y) position of the ellipse boundary using the following parametric equations

$$\begin{aligned} x &= xc + r \cos \theta \\ y &= yc + r \sin \theta \end{aligned}$$

The algorithm based on these parametric equations on polar co-ordinates can be summarized as below.

1. Input center of ellipse (xc,yc) and radii xr and yr.
2. Starting θ from angle 0° step minimum increments and compute boundary point of ellipse as

$$\begin{aligned} x &= xc + r \cos \theta \\ y &= yc + r \sin \theta \end{aligned}$$

3. Plot the point at position (round(x), round(y))
4. Repeat until θ is greater or equal to 360° .

The *drawellipse(int,int,int,int)* function can be written as

```
void drawellipse(int xc,int yc,int rx,int ry)
{
    int x,y;
    float theta;
```



```

const float PI=3.14;
for(theta=0.0;theta<=360;theta+=1)
{
    x= xc+rx*cos(theta*PI/180.0);
    y= yc+ry*sin(theta*PI/180.0);
    putpixel(x,y,1);
}
}

```

The methods of drawing ellipses explained above are not efficient. The method based on direct equation of ellipse must perform the square and square root operations due to which there may be floating point number computation which cause rounding off to plot the pixels. Also the square root causes the domain error. Due to the changing slope of curve along the path of ellipse, there may be un-uniform separation of pixel when slope changes. To eliminate this problem, extra computation is needed. Although, the method based on polar co-ordinate parametric equation gives the uniform spacing of pixel due to uniform increment of angle but it also take extra computation to evaluate the trigonometric functions. So these algorithms are not efficient to construct the ellipse. We have another algorithm called mid- point ellipse algorithm similar to mid-point circle algorithm which is efficient algorithm for computing ellipse.

Mid-Point Ellipse Algorithm:

The mid-point ellipse algorithm decides which point near the boundary (i.e. path of the ellipse) is closer to the actual ellipse path described by the ellipse equation. That point is taken as next point .

It is applied to the first quadrant in two parts as in figure. Region 1 and Region 2. We process by taking unit steps in x-coordinates direction and finding the closest value for y for each x-steps in region 1.

In first quadrant at region 1, we start at position $(0, r_y)$ and incrementing x and calculating y closer to the path along clockwise direction . When slope becomes -1 then shift unit step in x to y and compute corresponding x closest to ellipse path at Region 2 in same direction.

Alternatively , we can start at position $(r_x, 0)$ and select point in counterclockwise order shifting unit steps in y to unit step in x when slope becomes greater than -1.

Here, to implement mid-point ellipse algorithm, we take start position at $(0, r_y)$ and step along the ellipse path in clockwise position throughout the first quadrant.

We define ellipse function center at origin i.e, $(x_c, y_c) = (0, 0)$ as

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

$$f_{ellipse}(x, y) = \begin{cases} < 0, & \text{if (x,y) lies inside boundary of ellipse.} \\ = 0 & \text{if (x,y) lies on the boundary of ellipse.} \\ > 0 & \text{if (x,y) lies outside the boundary of ellipse.} \end{cases}$$

So $f_{ellipse}$ function serves as decision parameter in ellipse algorithm at each sampling position. We select the next pixel position according to the sign of decision parameter.

Starting at $(0, r_y)$, we take unit step in x-direction until we reach the boundary between the region 1 and region 2. Then we switch unit steps in y over the remainder of the curve in first quadrant. At each step, we need to test the slope of curve. The slope of curve is calculated as;

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

At the boundary between region 1 and region 2,

$$\frac{dy}{dx} = -1 \text{ and } 2r_y^2 = 2r_x^2 \text{ Therefore, we move out of region 1 when } 2r_y^2 x \geq 2r_x^2$$

Assuming the position (x_k, y_k) is filled, we move x_{k+1} to determine next pixel. The corresponding y value for x_{k+1} position will be either y_k or $y_k - 1$ depending upon the sign of decision parameter. So the decision parameter for region 1 is tested at mid point of $(x_k + 1, y_k)$ and $(x_k + 1, y_k - 1)$ i.e.

$$p_{1k} = f_{ellipse}(x_{k+1}, y_k - \frac{1}{2})$$

$$\text{or } p_{1k} = r_y^2 (x_{k+1})^2 + r_x^2 (y_k - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$\text{or } p_{1k} = r_y^2 (x_{k+1})^2 + r_x^2 y_k^2 - r_x^2 y_k + \frac{r_x^2}{4} - r_x^2 r_y^2 \dots\dots\dots(1)$$

if $p_{1k} < 0$, the mid point lies inside boundary, so next point to plot is $(x_k + 1, y_k)$ otherwise, next point to plot will be $(x_k + 1, y_k - 1)$

The successive decision parameter is computed as

$$p_{1k+1} = f_{ellipse}(x_{k+1} + 1, y_{k+1} - \frac{1}{2})$$

$$= r_y^2 (x_{k+1} + 1)^2 + r_x^2 (y_{k+1} - \frac{1}{2})^2 - r_x^2 r_y^2$$

$$\text{or } p_{1k+1} = r_y^2 (x_{k+1}^2 + 2x_{k+1} + 1) + r_x^2 (y_{k+1}^2 - y_{k+1} + \frac{1}{4}) - r_x^2 r_y^2$$

$$\text{or } p_{1k+1} = r_y^2 x_{k+1}^2 + 2r_y^2 x_{k+1} + r_y^2 + r_x^2 y_{k+1}^2 - r_x^2 y_{k+1} + \frac{r_x^2}{4} - r_x^2 r_y^2 \dots\dots\dots(2)$$

Subtracting (2) - (1)

$$p_{1k+1} - p_{1k} = 2r_y^2 x_{k+1} + r_y^2 + r_x^2 (y_{k+1}^2 - y_k^2) - r_x^2 (y_{k+1} - y_k)$$

if $p_{1k} < 0$, $y_{k+1} = y_k$ then,

$$\therefore p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

Otherwise $y_{k+1} = y_k - 1$ then we get,

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}$$

At the initial position, $(0, r_y)$ $2r_y^2 x = 0$ and $2r_x^2 y = 2r_x^2 r_y$

In region 1, initial decision parameter is obtained by evaluating ellipse function at $(0, r_y)$ as

$$p_{10} = f_{\text{ellipse}}(1, r_y - \frac{1}{2})$$

$$\text{or } p_{10} = f_{\text{ellipse}}(1, r_y - \frac{1}{2})$$

$$= r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Similarly, over the region 2, the decision parameter is tested at mid point of $(x_k, y_k - 1)$ and $(x_k + 1, y_k - 1)$ i.e.

$$p_{2k} = f_{\text{ellipse}}(x_k + \frac{1}{2}, y_k - 1)$$

$$= r_y^2 (x_k + \frac{1}{2})^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2$$

$$\therefore p_{2k} = r_y^2 x_k^2 + r_y^2 x_k + \frac{r_y^2}{4} + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2 \dots\dots\dots(3)$$

if $p_{2k} > 0$, the mid point lies outside the boundary, so next point to plot is $(x_k, y_k - 1)$ otherwise, next point to plot will be $(x_k + 1, y_k - 1)$

The successive decision parameter is computed as evaluating ellipse function at mid point of

$$p_{2k+1} = f_{\text{ellipse}}(x_{k+1} + \frac{1}{2}, y_{k+1} - 1) \text{ with } y_{k+1} = y_k - 1$$

$$p_{2k+1} = r_y^2 (x_{k+1} + \frac{1}{2})^2 + r_x^2 [(y_k - 1) - 1]^2 - r_x^2 r_y^2$$

$$\text{or } p_{2k+1} = r_y^2 x_{k+1}^2 + r_y^2 x_{k+1} + \frac{r_y^2}{4} + r_x^2 (y_k - 1)^2 - 2r_x^2 (y_k - 1) + r_x^2 - r_x^2 r_y^2 \dots\dots\dots(4)$$

subtracting (4)-(3)

$$p_{2k+1} - p_{2k} = r_y^2 (x_{k+1}^2 - x_k^2) + r_y^2 (x_{k+1} - x_k) - 2r_x^2 (y_k - 1) + r_x^2$$

$$\text{or } p_{2k+1} = p_{2k} + r_y^2 (x_{k+1}^2 - x_k^2) + r_y^2 (x_{k+1} - x_k) - 2r_x^2 (y_k - 1) + r_x^2$$

if $p_{2k} > 0$, $x_{k+1} = x_k$ then

$$p_{2k+1} = p_{2k} - 2r_x^2 (y_k - 1) + r_x^2$$

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

$$p_{2k+1} = p_{2k} + r_y^2 [(x_k + 1)^2 - x_k^2] + r_y^2 (x_k + 1 - x_k) - 2r_x^2 (y_k - 1) + r_x^2$$

$$\text{or } p_{2k+1} = p_{2k} + r_y^2 (2x_k + 1) + r_y^2 - 2r_x^2 (y_k - 1) + r_x^2$$

$$\text{or } p_{2k+1} = p_{2k} + r_y^2 (2x_k + 2) - 2r_x^2 (y_k - 1) + r_x^2$$

$$\text{or } p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2 \text{ where } x_{k+1} = x_k + 1 \text{ and } y_{k+1} = y_k - 1$$

The initial position for region 2 is taken as last position selected in region 1 say which is (x_0, y_0) then initial decision parameter in region 2 is obtained by evaluating ellipse function at mid point of $(x_0, y_0 - 1)$ and $(x_0 + 1, y_0 - 1)$ as

$$\begin{aligned} p_{20} &= f_{\text{ellipse}}(x_0 + \frac{1}{2}, y_0 - 1) \\ &= r_y^2 (x_0 + \frac{1}{2})^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2 \end{aligned}$$

Now the mid point ellipse algorithm is summarized as;

1. Input center (x_c, y_c) and r_x and r_y for the ellipse and obtain the first point as

$$(x_0, y_0) = (0, r_y)$$

2. Calculate initial decision parameter value in Region 1 as

$$P_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

3. At each x_k position, in Region 1, starting at $k = 0$, compute

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

If $p_{1k} < 0$, then the next point to plot is

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

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

Otherwise next point to plot is

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

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1} \quad \text{with } x_{k+1} = x_k + 1 \text{ and } y_{k+1} = y_k - 1$$

4. Calculate the initial value of decision parameter at region 2 using last calculated point say (x_0, y_0) in region 1 as

$$p_{20} = r_y^2 \left(x_0 + \frac{1}{2}\right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

5. At each y_k position in Region 2 starting at $k = 0$, perform computation

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

if $p_{2k} > 0$, then

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

$$p_{2k+1} = p_{2k} - 2r_x^2 (y_k - 1) + r_x^2$$

Otherwise

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

$$p_{2k+1} = p_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2 \quad \text{where } x_{k+1} = x_k + 1 \text{ and } y_{k+1} = y_k - 1$$

6. Determine the symmetry points in other 3 quadrants.
7. Move each calculated point (x_k, y_k) on to the centered (x_c, y_c) ellipse path as

$$x_k = x_k + x_c;$$

$$y_k = y_k + y_c$$

8. Repeat the process for region 1 until $2r_y^2 x_k \geq 2r_x^2 y_k$ and region until $(x_k, y_k) = (r_x, 0)$

Two Dimensional Geometric Transformations

In computer graphics, transformations of 2D objects are essential to many graphics applications. The transformations are used directly by application programs and within many graphics subroutines in application programs. Many applications use the geometric transformations to change the position, orientation, and size or shape of the objects in drawing. Rotation, Translation and scaling are three major transformations that are extensively used by almost all graphical packages or graphical subroutines in applications. Other than these, reflection and shearing transformations are also used by some graphical packages.

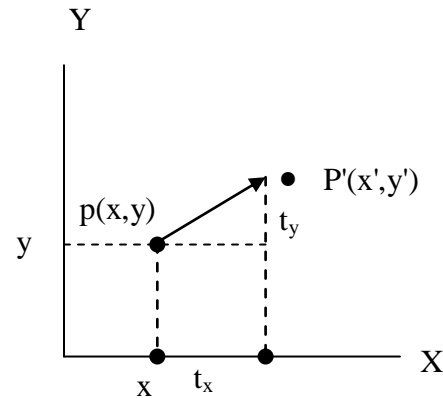
2D Translation

A translation is applied to an object by re-positioning it along a straight line path from one co-ordinate location to another. We translate a two-dimensional point by adding translation distances, t_x, t_y to the respective co-ordinate values of original co-ordinate position (x, y) to move the point to a new position (x', y') as:

$$x' = x + t_x$$

$$y' = y + t_y$$

The translation distance pair (t_x, t_y) is known as translation vector or shift vector. We can express translation equations as matrix representations as



$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\therefore P' = P + T$$

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

Some times matrix transformation are represented by co-ordinate rows vector instead of column vectors as.

$$P = (x, y) \quad T = (t_x, t_y) \quad P' = P + T.$$

For translation of any object in Cartesian plane, we transform the distinct co-ordinates by the translation vector and re-draw image at the new transformed location.

2D Rotation

The 2D rotation is applied to re-position the object along a circular path in XY-plane. To generate rotation, we specify a rotation angle θ , through which the co-ordinates are to be rotated. Rotation can be made by angle θ either clockwise or anticlockwise direction. Besides the angle of rotation θ , there should be a pivot point through which the object is to be rotated. The positive θ rotates object in anti-clockwise direction and the negative value of θ rotates the object in clock-wise direction.

A line perpendicular to rotating plane and passing through pivot point is called axis of rotation.

Let $P(x,y)$ is a point in XY-plane which is to be rotated with angle θ . Also let $OP = r$ (As in figure below) is constant distance from origin. Let r makes angle ϕ with positive X – direction as shown in figure.

When OP is rotated through angle θ Taking origin as pivot point for rotation, then OP' makes angle $\theta + \phi$ with X-axis.

Now ,

$$x' = r \cos(\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta$$

$$y' = r \sin(\phi + \theta) = r \sin \phi \cos \theta + r \cos \phi \sin \theta$$

But, $r \cos \phi = x, r \sin \phi = y$ therefore we get

$$x' = x \cos \theta - y \sin \theta \text{-----1}$$

$$y' = x \sin \theta + y \cos \theta \text{-----2} \quad \text{which are equation for rotation of (x,y) with angle } \theta \text{ and taking pivot as origin.}$$

Rotation from any arbitrary pivot point (x_r, y_r)

- Let $Q(x_r, y_r)$ is pivot point for rotation.
- $P(x,y)$ is co-ordinate of point to be rotated by angle θ .
- Let ϕ is the angle made by QP with X-direction.

Then angle made by QP' with X-direction is $\theta + \phi$

Hence,

$$\cos(\phi + \theta) = (x' - x_r) / r$$

$$\text{or } r \cos(\phi + \theta) = (x' - x_r)$$

$$\text{or } x' - x_r = r \cos \phi \cos \theta - r \sin \phi \sin \theta$$

$$\text{since } r \cos \phi = x - x_r, r \sin \phi = y - y_r$$

$$x' = x_r + (x - x_r) \cos \theta - (y - y_r) \sin \theta \text{.....(1)}$$

Similarly,

$$\sin(\phi + \theta) = (y' - y_r) / r$$

$$\text{or } r \sin(\phi + \theta) = (y' - y_r)$$

$$\text{or } y' - y_r = r \sin \phi \cos \theta + r \cos \phi \sin \theta$$

$$\text{since } r \cos \phi = x - x_r, r \sin \phi = y - y_r$$

$$y' = y_r + (x - x_r) \sin \theta + (y - y_r) \cos \theta \dots\dots\dots(2)$$

These equations (1) and (2) are the equations for rotation of a point (x,y) with angle θ taking pivot point (x_r, y_r) .

The rotation about pivot point (x_r, y_r) can be achieved by sequence of translation, rotation about origin and reverse translation.

- Translate the point (x_r, y_r) and $P(x, y)$ by translation vector $(-x_r, -y_r)$ which translates the pivot to origin and $P(x, y)$ to $(x - x_r, y - y_r)$.
- Now apply the rotation equations when pivot is at origin to rotate the translated point $(x - x_r, y - y_r)$ as:

$$x_1 = (x - x_r) \cos \theta - (y - y_r) \sin \theta$$

$$y_1 = (x - x_r) \sin \theta + (y - y_r) \cos \theta$$

- Re- translate the rotated point (x_1, y_1) with translation vector (x_r, y_r) which is reverse translation to original translation. Finally we get the equation after successive transformation as

$$x' = x_r + (x - x_r) \cos \theta - (y - y_r) \sin \theta \dots\dots\dots(1)$$

$$y' = y_r + (x - x_r) \sin \theta + (y - y_r) \cos \theta \dots\dots\dots(2)$$

Which are actually the equations for rotation of (x,y) from the pivot point (x_r, y_r)

2D

Scaling:

A scaling transformation alters the size of the object. This operation can be carried out for polygon by multiplying the co-ordinate values (x,y) of each vertex by scaling factor s_x and s_y to produce transformed co-ordinates (x', y') .

$$\text{i.e. } x' = x.s_x \quad \text{and } y' = y.s_y$$

Scaling factor s_x scales object in x- direction and s_y scales in y- direction. If the scaling factor is less than 1, the size of object is decreased and if it is greater than 1 the size of object is increased. The scaling factor = 1 for both direction does not change the size of the object. If both scaling factors have same value then the scaling is known as uniform scaling. If the value of s_x and s_y are different, then the scaling is known as differential scaling. The differential scaling is mostly used in the graphical package to change the shape of the object.

The matrix equation for scaling is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \text{ i.e. } P' = S.P$$

Homogeneous co-ordinate representation of 2D Transformation

- The homogeneous co-ordinate system provide a uniform frame-work for handling different geometric transformations, simply as multiplication of matrices.
- Its extension to 3D is straight forward which also helps to produce prespective projections by use of matrix multiplication. We simply add a third co-ordinate to 2D point i.e. $(x,y) = (x_h, y_h, h)$ where $x = x_h/h$, $y = y_h/h$ where h is 1 usually for 2D case.
- By using this homogeneous co-ordinate system a 2D point would be $(x,y,1)$. The matrix formulation for 2D translation for $T(t_x, t_y)$ is :

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ By which we can get}$$

$$x' = x + t_x$$

$$y' = y + t_y \quad Y$$

For Rotation: $R(\theta)$ about origin the homogeneous matrix equation will be

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ which gives the } Y$$

$$x' = x \cos \theta - y \sin \theta \text{-----1}$$

$$y' = x \sin \theta + y \cos \theta \text{-----2}$$

which are equation for rotation of (x,y) with angle θ and taking pivot as origin.

Rotation about an arbitrary fixed pivot point (x_r, y_r)

For a fixed pivot point rotation, we can apply composite transformation as

1. Translate fixed point (x_r, y_r) to the co-ordinate origin by $T(-x_r, -y_r)$.
2. Rotate with angle $\theta \rightarrow R(\theta)$.
3. Translate back to original position by $T(x_r, y_r)$

This composite transformation is represented as:

$$P' = T(x_r, y_r). R(\theta). T(-x_r, -y_r). P$$

Which can be represented in matrix equation using homogeneous co-ordinate system as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{Expanding this equation we get}$$

$$x' = x_r + (x - x_r) \cos \theta - (y - y_r) \sin \theta \dots\dots\dots(1)$$

$$y' = y_r + (x - x_r) \sin \theta + (y - y_r) \cos \theta \dots\dots\dots(2) \quad \text{Which are actually the equations for rotation of (x,y) from the pivot point (x}_r, y_r)$$

Scaling with scaling factors (s_x, s_y)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \text{which exactly gives the equations}$$

$$x' = x.s_x \quad \text{and} \quad y' = y.s_y$$

Fixed point scaling:

Objects transformed with standard scaling equation are scaled as well as re-positioned. Scaling factor with value less than 1 moves object closer to the origin where as value of scaling factor greater than 1 moves object away from origin.

- To control the location of scaled object we can choose the position called fixed point. Let co-ordinates of fixed point = (x_f, y_f). It can be any point on the object.
- A polygon is then scaled relative to (x_f, y_f) by scaling the distance from each vertex to the fixed point.
- For a vertex with co-ordinate (x,y), the scaled co-ordinates (x', y') are calculated as:

$$\begin{aligned} x' &= x_f + (x - x_f)s_x \\ y' &= y_f + (y - y_f)s_y \end{aligned} \quad \text{or equivalently,}$$

$$\begin{aligned} x' &= x.s_x + (1 - s_x)x_f \\ y' &= y.s_y + (1 - s_y)y_f \end{aligned}$$

Where $(1 - s_x)x_f$ and $(1 - s_y)y_f$ are constant for all points in object.

To represent fixed point scaling using matrix equations in homogeneous co-ordinate system, we can use composite transformation as in fixed point rotation.

1. Translate object to the origin so that (x_f, y_f) lies at origin by $T(-x_f, -y_f)$.

2. Scale the object with (s_x, s_y)
 3. Re- translate the object back to its original position so that fixed point (x_f, y_f) moves to its original position. In this case translation vector is $T(x_f, y_f)$.
- $\therefore P' = [T(x_f, y_f) \cdot S(s_x, s_y) \cdot T(-x_f, -y_f)]P$

The homogeneous matrix equation for fixed point scaling is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_f(1-s_x) \\ 0 & s_y & y_f(1-s_y) \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Directive scaling

Standard and fixed point scaling scales object along x and y axis only. Directive scaling scales the object in any direction.

Let S_1 and S_2 are given directions for scaling at angle Θ from co-ordinate axes as in figure below

1. First perform the rotation so that directions S_1 and S_2 coincide with x and y – axes.
2. Then the scaling transformation is applied to scale the object by given scaling factors (s_1, s_2) .
3. Re- apply the rotation in opposite direction to return to their original orientation.

For any point P in object, the directive scaling position P' is given by following composite transformation.

$P' = R^{-1}(\theta) \cdot S(s_1, s_2) \cdot R(\theta) \cdot P$ for which the homogeneous matrix equation is

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Reflection:

A reflection is a transformation that produce a mirror image of an object. In 2D-transformation, reflection is generated relative to an axis of reflection. The reflection of an object to an relative axis of reflection, is same as 180° rotation about the reflection axis.

1. Reflection about X-axis: The line representing x-axis is $y = 0$. The reflection of a point $P(x, y)$ on x-axis, changes the y-coordinate sign. i.e. Reflection about x-axis,

the reflected position of P(x,y) will be P'(x,-y). Hence, reflection in x-axis is accomplished with transformation equation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

gives the reflection of a point.

To reflect an 2D object, reflect each distinct points of object by above equation, then joining the points with straight line, redraws the image for reflected image.

2. Reflection about Y-axis: The line representing y-axis is $x = 0$. The reflection of a point P(x,y) on y-axis changes the sign of x-coordinate. i.e. P(x,y) changes to P'(-x,y).

Hence reflection of a point on y-axis is obtained by following matrix equation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For any 2D object, reflect each point and re-draw image joining the reflected images of all distinct points.

3. Reflection on an arbitrary axis: The reflection on any arbitrary axis of reflection can be achieved by sequence of rotation and co-ordinate axes reflection matrices.
 - First, rotate the arbitrary axis of reflection so that the axis of reflection and one of the co-ordinate axis coincide.
 - Reflect the image on the co-ordinate axis to which the axis of reflection coincides.
 - Rotate the axis of reflection back to its original position.

For example consider a line $y = x$ for axis of reflection, the possible sequence of transformation for reflection of 2D object are

4. Reflection about origin: The reflection on the line perpendicular to xy-plane and passing through flips x and y co-ordinates both. So sign of x and y co-ordinate value changes. The equivalent matrix equation for the point is:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Shearing:

A transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other is called shear.

X-direction Shear: An X-direction shear relative to x-axis is produced with transformation matrix equation.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ which transforms } x' = x + sh_x \text{ and } y' = y$$

A unit square transformed to a parallelogram using x-direction shear with $sh_x = 2$.

Y-direction shear: An y-direction shear relative to y-axis is produced by following transformation equations.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \text{ which transforms } x' = x \text{ and } y' = y + x.sh_y$$

Composite Transformation

With the matrix representation of transformation equations it is possible to setup a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of individual transformation.

For column matrix representation of coordinate positions we form composite transformation by multiplying matrices in order from right to left.

i. Two Successive Translation are Additive

Let two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P then

or, $P' = T(t_{x2}, t_{y2}) \cdot \{T(t_{x1}, t_{y1}) \cdot P\}$ and $P' = \{T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1})\} \cdot P$

Here the composite transformation matrix for this sequence of translation is

$$\begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{or, } T(t_{x2}, t_{y2}) \cdot T(t_{x1}, t_{y1}) = T(t_{x1} + t_{x2}, t_{y1} + t_{y2})$$

ii. Two successive Scaling operations are Multiplicative

Let (s_{x1}, s_{y1}) and (s_{x2}, s_{y2}) be two successive vectors applied to a coordinate position P then the composite scaling matrix thus produced is

$$\text{or, } P' = S(s_{x2}, s_{y2}) \cdot \{S(s_{x1}, s_{y1}) \cdot P\} \quad \text{and} \quad P' = \{S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})\} \cdot P$$

Here the composite transformation matrix for this sequence of translation is

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x2} \cdot s_{x1} & 0 & 0 \\ 0 & s_{y2} \cdot s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{or, } S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1}) = S(s_{x1} \cdot s_{x2}, s_{y1} \cdot s_{y2})$$

iii. Two successive Rotation operations are Additive

Let $R(\theta_1)$ and $R(\theta_2)$ be two successive rotations applied to a coordinate position P then the composite scaling matrix thus produced is

$$\text{or, } P' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\} \quad \text{and} \quad P' = \{R(\theta_2) \cdot R(\theta_1)\} \cdot P$$

Here the composite transformation matrix for this sequence of translation is

or,

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 \\ \sin\theta_2 & \cos\theta_2 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 \\ \sin\theta_1 & \cos\theta_1 \end{bmatrix}$$

or,

$$\begin{bmatrix} \cos\theta_2 \cdot \cos\theta_1 - \sin\theta_2 \cdot \sin\theta_1 & -\cos\theta_2 \cdot \sin\theta_1 - \sin\theta_2 \cdot \cos\theta_1 \\ \sin\theta_2 \cdot \cos\theta_1 + \sin\theta_1 \cdot \cos\theta_2 & \cos\theta_2 \cdot \cos\theta_1 - \sin\theta_2 \cdot \sin\theta_1 \end{bmatrix}$$

or,

$$\begin{bmatrix} \cos(\theta_2 + \theta_1) & -\sin(\theta_2 + \theta_1) \\ \sin(\theta_2 + \theta_1) & \cos(\theta_2 + \theta_1) \end{bmatrix}$$

$$\text{or, } R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$$

Affine Transformation

Linear 2D geometric transformation which maps variables (e.g. pixel intensity values located at point (x_1, y_1) in an input image) into new variables (e.g. (x_2, y_2) in an output image) by applying a linear combination of translation, rotation, scaling and/or shearing operations.

/*

Example : Program for translation of 2D object using homogeneous co-ordinates representation of object for its vertices. Which translates the rectangle by given translation vector.

*/

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
```

```

void matrixmultiply(int T[3][3],int V[3][5],int r[3][5]);
void main()
{
    int gdriver, gmode, errorcode;
    int i,j,k;
    int vertex[3][5]={100,100,200,200,100},
                    {100,200,200,100,100},
                    {1,1,1,1,1},
                    };
    int translate[3][3] ={{1,0,100},{0,1,200},{0,0,1}};
    int result[3][5];
    gdriver=DETECT; /* request auto detection */
    /* initialize graphics and local variables */
    initgraph(&gdriver, &gmode, "\\tc\\bgi");
    /* read result of initialization */
    errorcode = graphresult();
    /* an error occurred */
    if (errorcode != grOk)
    {
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key ....");
        getch();
        /* terminate with an error code */
        exit(1);
    }

    setbkcolor(2);
    for(i=0;i<4;i++)
    {
        setcolor(BLUE);

        line(vertex[0][i],vertex[1][i],vertex[0][i+1],vertex[1][i+1]);

    }

    printf("Press any key for the translated line.....\n");
    getch();
    matrixmultiply(translate,vertex,result);
    for(i=0;i<4;i++)
    {
        setcolor(YELLOW);

        line(result[0][i],result[1][i],result[0][i+1],result[1][i+1]);

    }
    getch();
    closegraph();
}
void matrixmultiply(int translate[3][3],int vertex[3][5],int
result[3][5])
{
    for(int i=0;i<=3;i++)
    {
        for(int j=0;j<=5;j++)
        {

```

```

        result[i][j]=0;
        for(int k=0;k<=3;k++)
            result[i][j]+=translate[i][k]*vertex[k][j];
    }
}

```

Filled Area Primitives

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

Polygons are easier to process due to linear boundaries

Two basic approaches to area filling on a raster system

1. Determine the overlap intervals for scan lines that cross the area.

Typically useful for filling polygons, circles, ellipses

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

Filling rectangles

Two things to consider

i. which pixels to fill

ii. with what value to fill

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

To fill rectangle with solid color

Set each pixel lying on scan line running from left edge to right with same

pixel value, each span from x_{\max} to x_{\min}

```

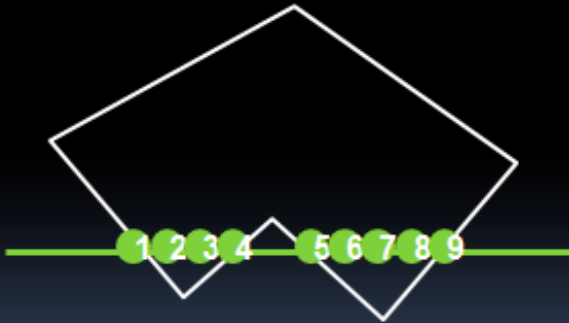
        for( y from  $y_{\min}$  to  $y_{\max}$  of rectangle)           /*scan line*/
            for( x from  $x_{\min}$  to  $x_{\max}$  of rectangle)         /*by pixel*/
                writePixel(x, y, value);

```


Filling Polygons

Filling Polygons

Scan-line fill algorithm
Inside-Outside tests



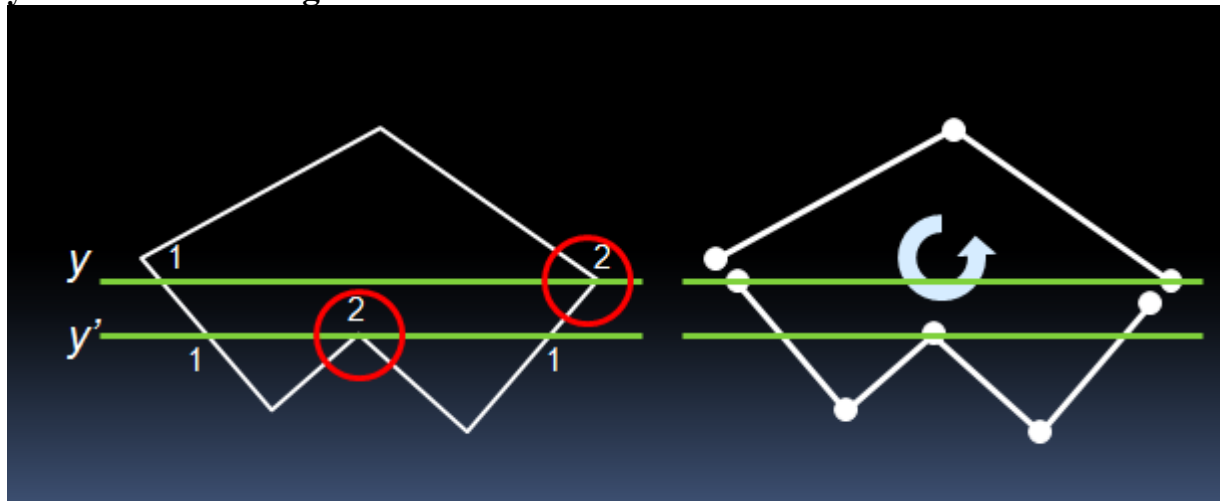
Boundary fill algorithm



Topological Difference between 2 Scan lines

y : intersection edges are opposite sides

y' : intersection edges are same side



Boundary Fill

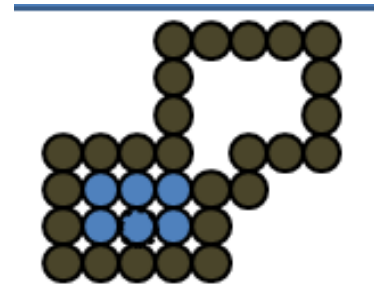
Start at a point inside a region and paint the interior outward toward the boundary.

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

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

Starting from (x, y) the procedure tests neighboring position to determine whether they are of the boundary color.
 If not they are painted with fill color and their neighbors are tested
 Process continues until all pixels up to boundary color for the area have been tested

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

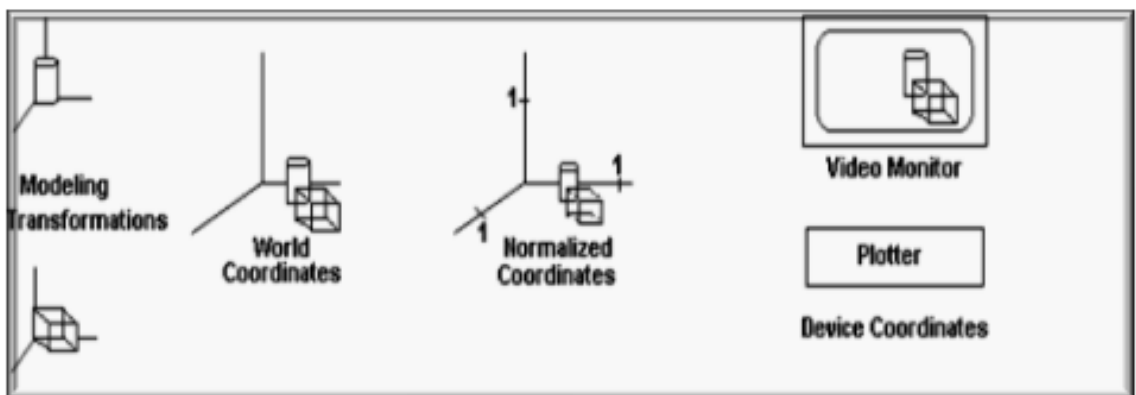
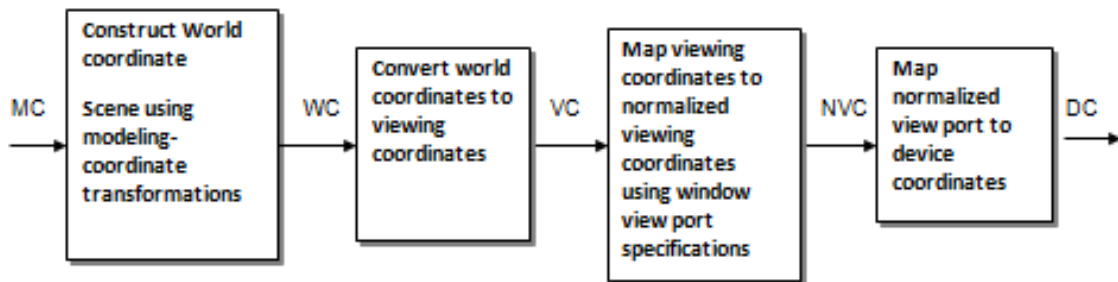


2D, 3D Clipping

Window and View ports

- A rectangular area specified in world coordinates is called a window.
- A rectangular area on the display device to which a window is mapped is called a view port.
- The window defines what is to be viewed; the view port defines where it is to be displayed.
- Often windows and view ports are rectangles in standard position with rectangle edges parallel to coordinate axes
- The mapping of a part of world coordinate scene to device coordinate is referred to as viewing transformation.

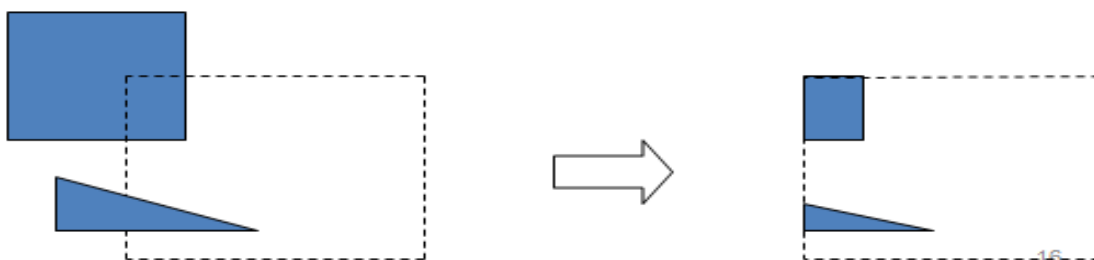
2D Viewing Pipeline



15

Clipping in Raster World

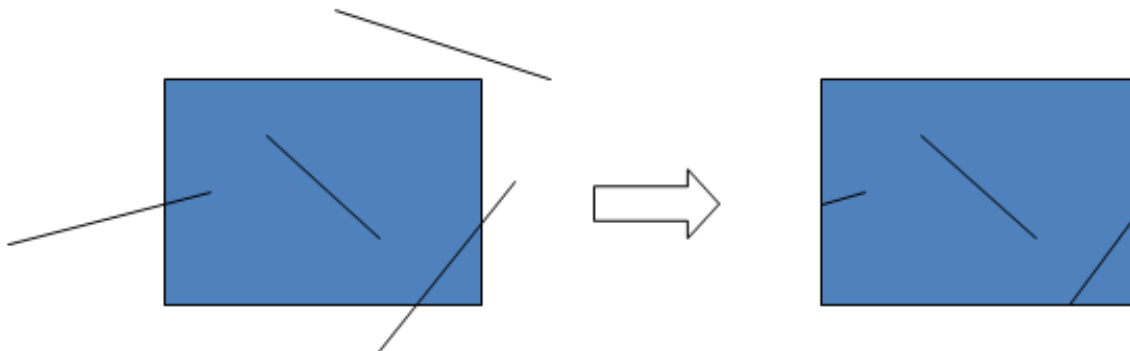
- Procedure that identifies those operations of picture that are either inside or outside
- The region against which an object is to be clipped is called a clip window.
- Depending on application it can be polygons or even curve surfaces.



16

- **Applications**

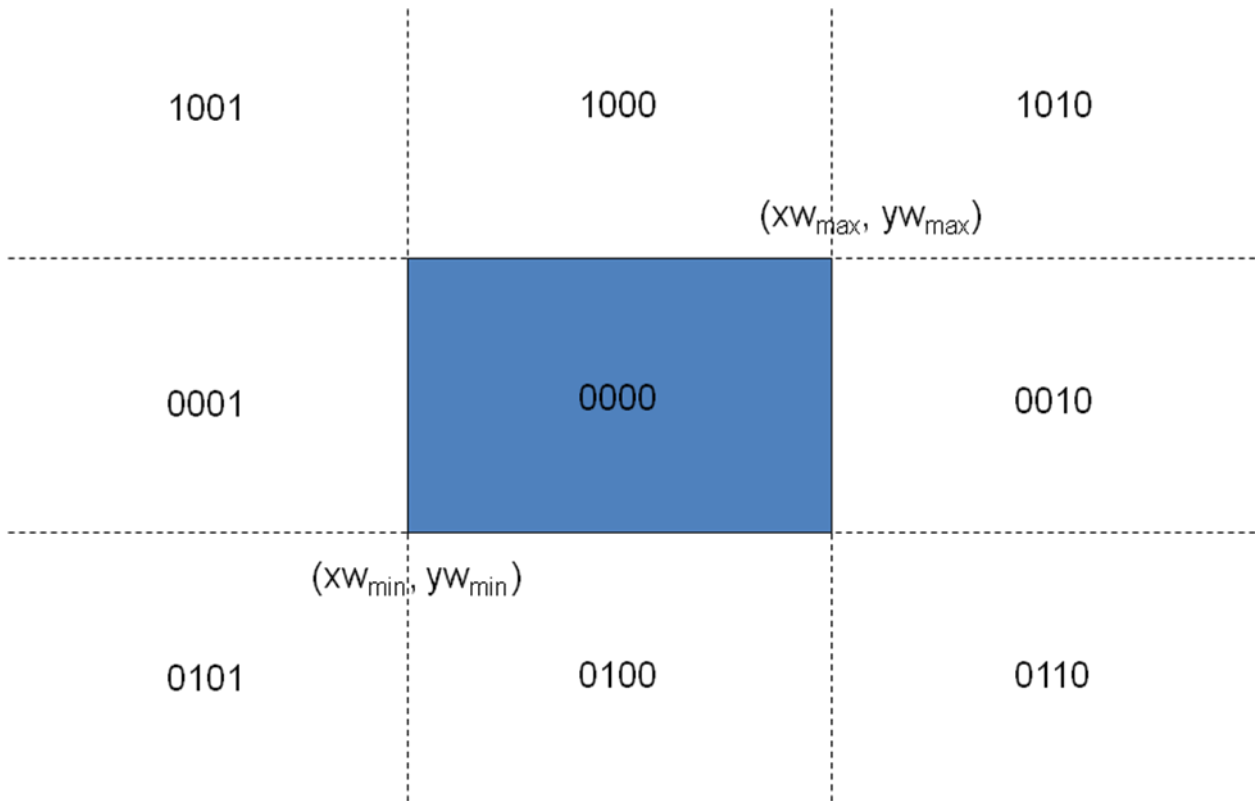
- i. Extracting parts of defined scene for viewing
 - ii. Identifying visible surfaces in three dimension views
- iii. Drawing, painting operations that allow parts of picture to be selected for copying, moving, erasing or duplicating etc.
- Point clipping easy: Just check the inequalities
 - $x_{\min} < x < x_{\max}$
 - $y_{\min} < y < y_{\max}$
- Line clipping more tricky



Cohen-Sutherland Line Clipping

- Divide 2D space into 3x3 regions.
- Middle region is the clipping window.
- Each region is assigned a 4-bit code.
- Bit 1 is set to 1 if the region is to the left of the clipping window, otherwise. Similarly for bits 2, 3 and 4.

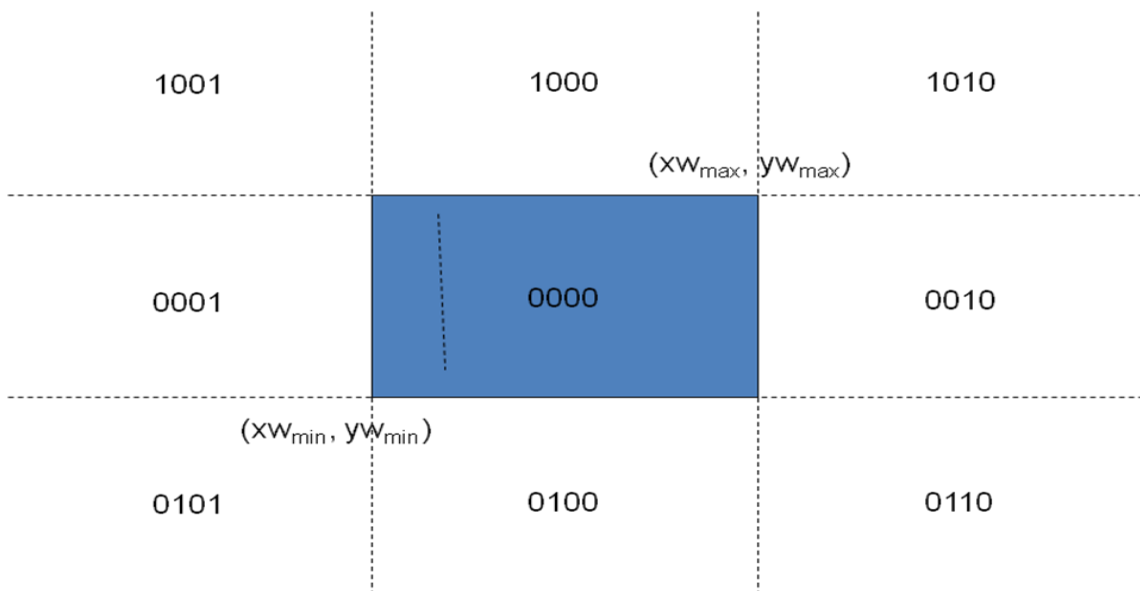
4	3	2	1
Top	Bottom	Right	Left



- To clip a line, find out which regions its two endpoints lie in.

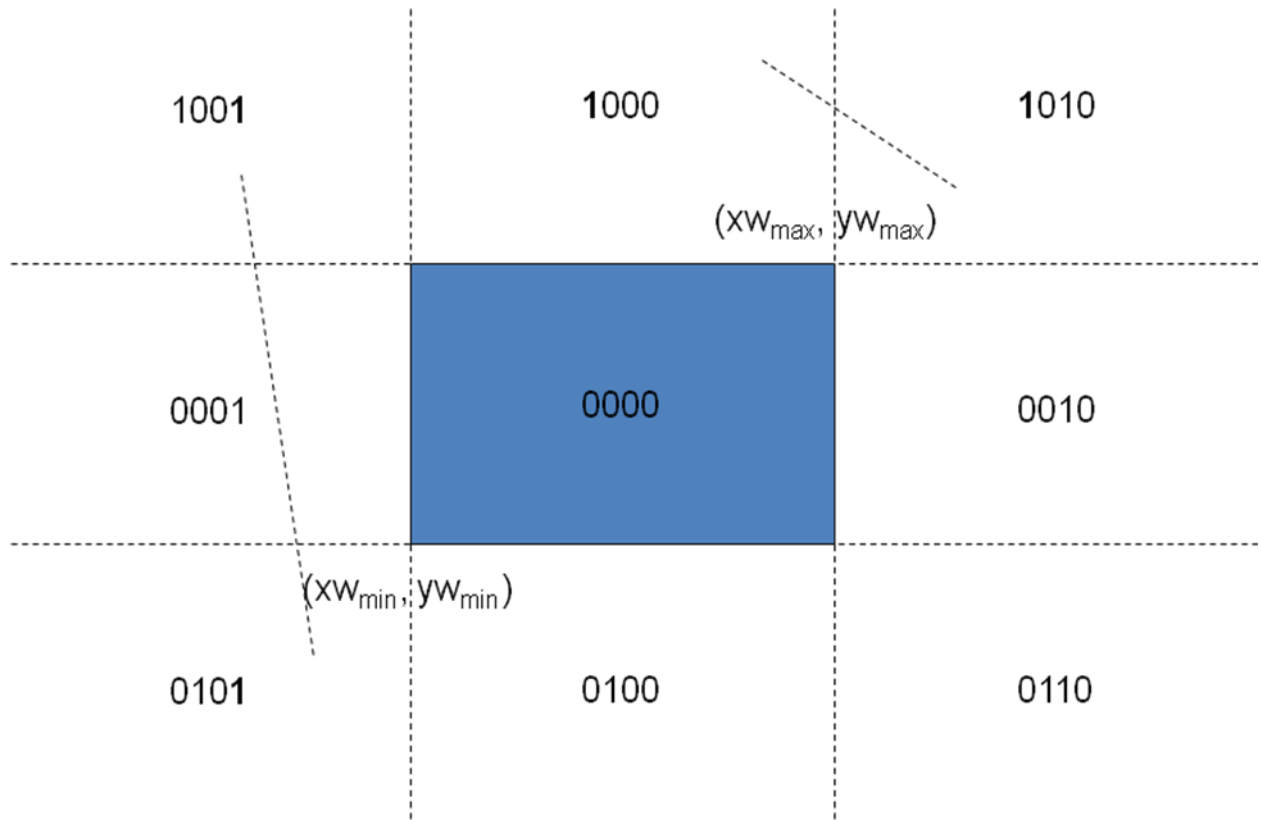
Case I

- If they are both in region 0000, then it's completely in.



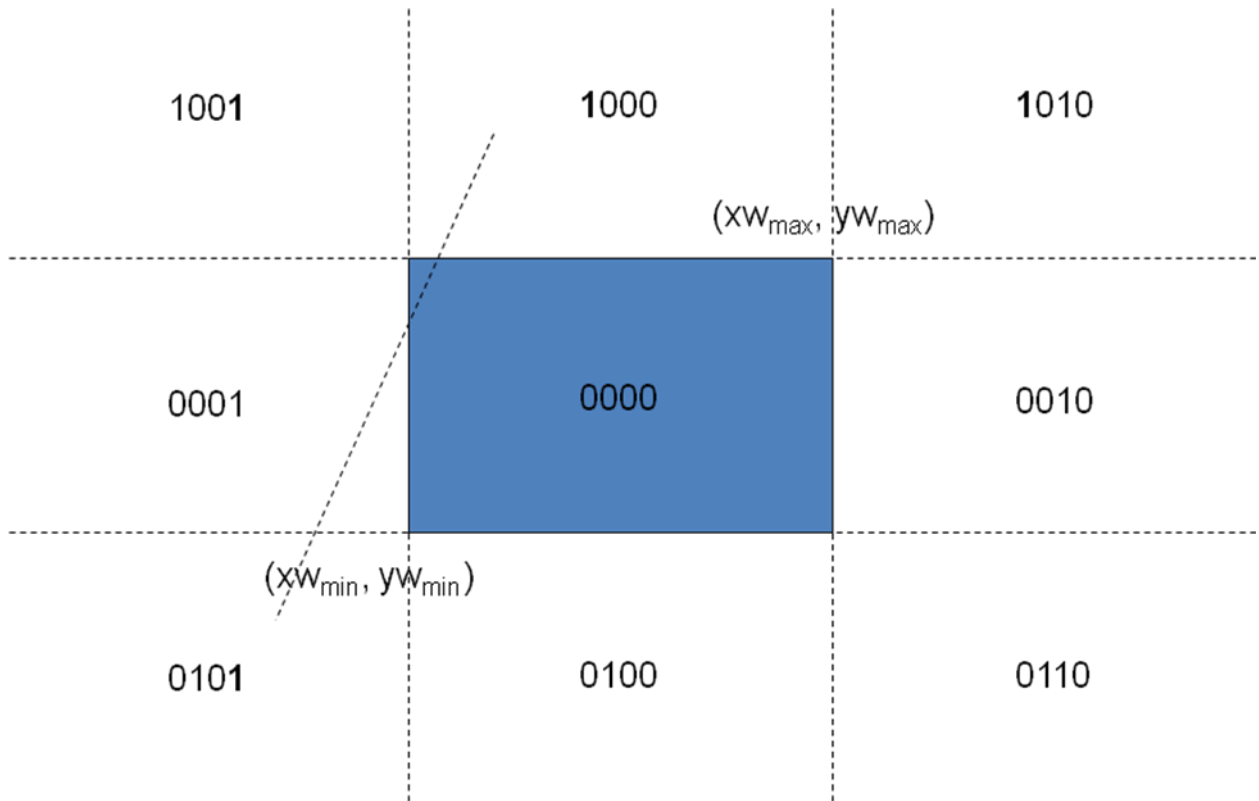
Case II

- If the two region numbers both have a 1 in the same bit position, the line is completely out.



Case III

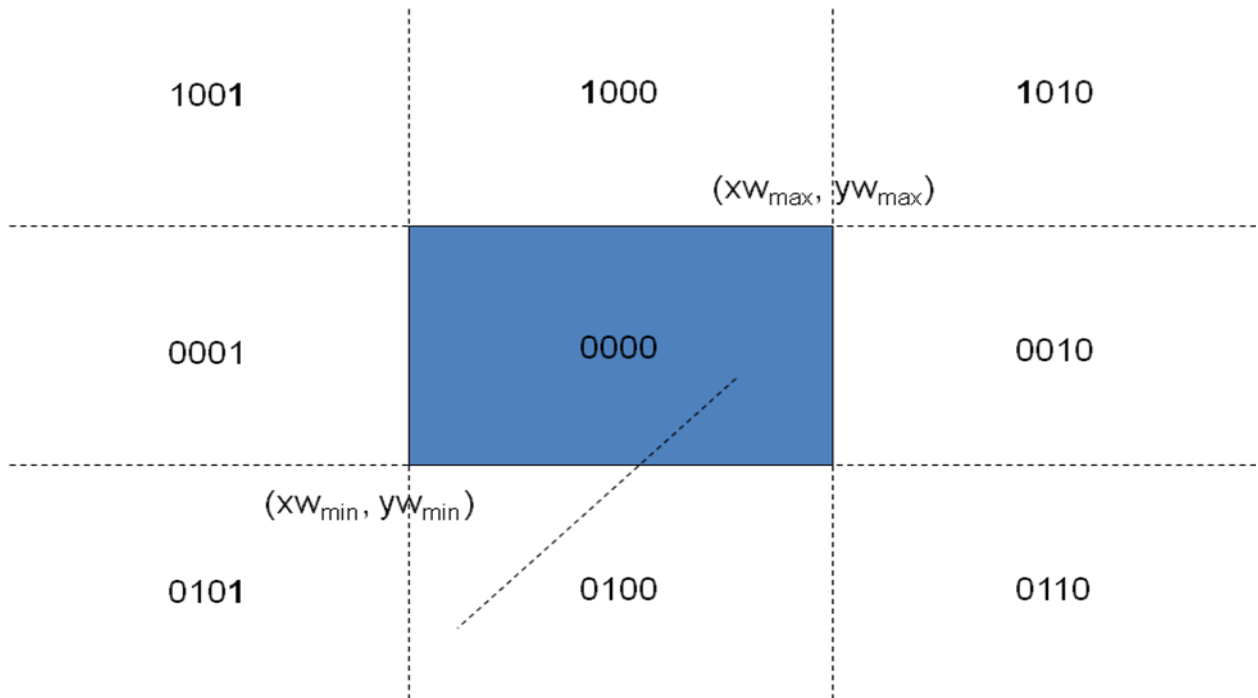
- If lines can not be identified as completely inside or outside we have to do some more calculations.
- Here we find the intersection points with a clipping boundary using the slope intercept form of the line equation



- For a line with end point coordinates (x_1, y_1) and (x_2, y_2) the y coordinate of the intersection point with a vertical boundary is

$$y = y_1 + m (x - x_1)$$

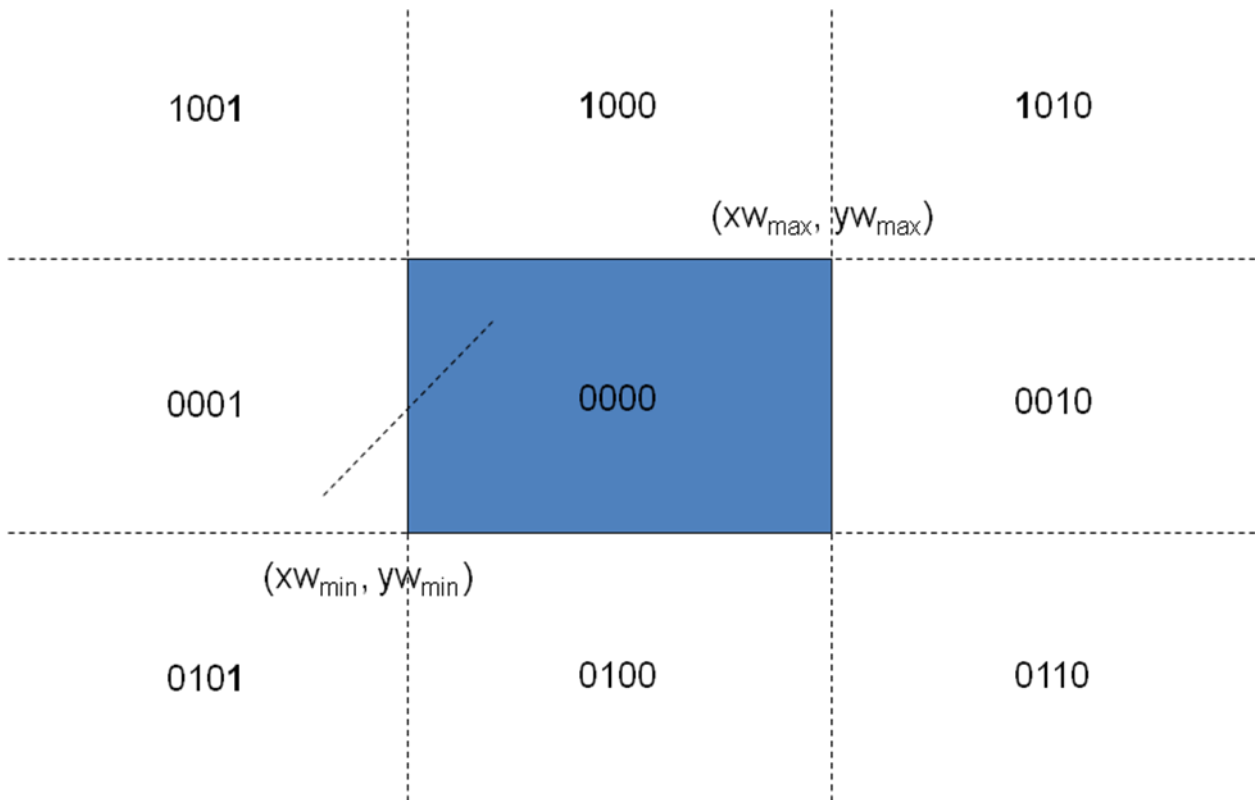
- Where x is set to either x_{wmin} or x_{wmax}



- Similarly for the intersection with a vertical boundary

$$x = x_1 + (y - y_1)/m$$

- Where y is set to either yw_{min} or yw_{max}



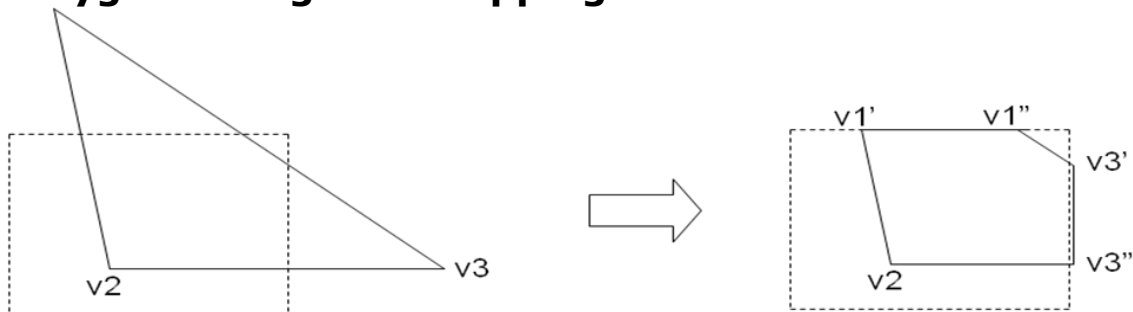
- For those lines that we cannot immediately determine, we successively clip against each boundary.
- Then check the new endpoint for its region code.
- **How to find boundary intersection:** To find y coordinate at vertical intersection, substitute x value at the boundary into the line equation of the line to be clipped.
- **Other algorithms:**
 - Faster: Cyrus-Beck
 - Even faster: Liang-Barsky
 - Even faster: Nichol-Lee-Nichol

Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(70,90) , B(60,85) against a clip window with its lower left corner at (50,80) and upper right corner at (90,120)

Hints:

- Find the region code for each end point of the line
- Find if it is above, below, left or right of the clip window
- Determine the intersection point of the line segment with the boundary

Polygon Filling Area Clipping

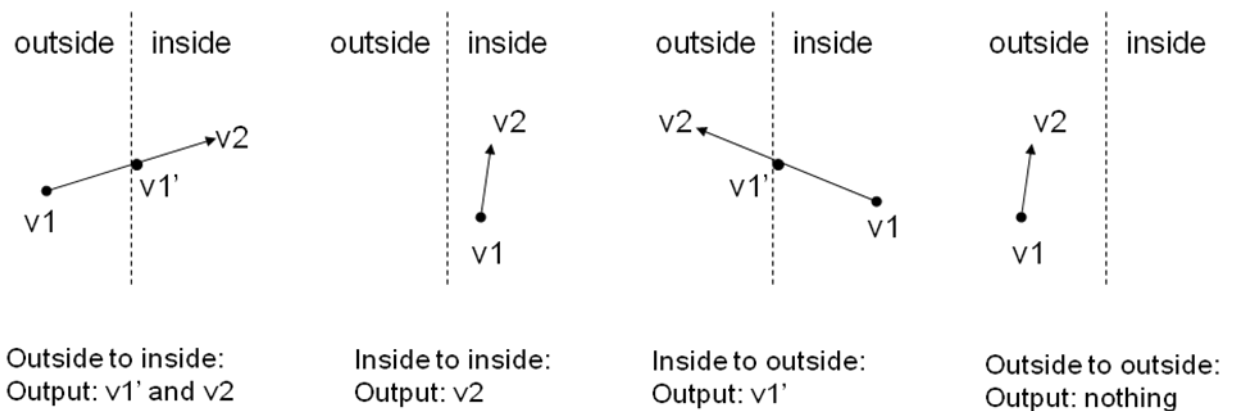


Note: Need to consider each of 4 edge boundaries

Sutherland-Hodgman Polygon Clipping

- Input each edge (vertex pair) successively.
- Output is a new list of vertices.
- Each edge goes through 4 clipper.
- The rule for each edge for each clipper is:
 - If first input vertex is outside, and second is inside, output the intersection and the second vertex
 - If first both input vertices are inside, then just output second vertex
 - If first input vertex is inside, and second is outside, output is the intersection
 - If both vertices are outside, output is nothing

Four possible scenarios at each clipper



Clipping in 3D

- Suppose the view volume has been normalized. Then the clipping boundaries are just:

$$xw_{\min} = -1 \quad xw_{\max} = 1$$

$$yw_{\min} = -1 \quad yw_{\max} = 1$$

$$zw_{\min} = -1 \quad zw_{\max} = 1$$

Clipping Homogeneous Coordinates in 3D

- Coordinates expressed in homogeneous coordinates
- After geometric, viewing and projection transformations, each vertex is: (x_h, y_h, z_h, h)
- Therefore, assuming coordinates have been normalized to a $(-1,1)$ volume, a point (x_h, y_h, z_h, h) is inside the view volume if:

$$-1 < \frac{x_h}{h} < 1 \quad \text{and} \quad -1 < \frac{y_h}{h} < 1 \quad \text{and} \quad -1 < \frac{z_h}{h} < 1$$

Suppose that $h > 0$, which is true in normal cases, then

$$-h < x_h < h \quad \text{and} \quad -h < y_h < h \quad \text{and} \quad -h < z_h < h$$

3D Cohen-Sutherland Region Codes

- Simply use 6 bits instead of 4.

6	5	4	3	2	1
Far	Near	Top	Bottom	Right	Left

Example: If $h + x_h < 0$, then bit 1 is set to 1.

This is because if $-h > x_h$, then the point is to the left of the viewing volume.

Clipping Polygons

- **First, perform trivial acceptance and rejection using, for example, its coordinate extents.**
- **Polygons usually split into triangle strips.**
- **Then each triangle is clipped using 3D extension of the Sutherland-Hodgman method**

Arbitrary Clipping Planes

- **Can specify an arbitrary clipping plane: $Ax + By + Cz + D = 0$.**
- **Therefore, for any point, if $Ax + By + Cz + D < 0$, it is not shown.**
- **To clip a line against an arbitrary plane,**
 - **If both end-points are in, then the line is in**
 - **If both end-points are out, then the line is out**
 - **If one end-point is in, and one is out, then we need to find the intersection of the line and the plane**

Intersection of Line and Plane

- **First, given two end-points of a line, P_1 and P_2 , form a parametric representation of the line:**

$$P = P_1 + (P_2 - P_1) u, \text{ where } 0 < u < 1$$

Equation of the clipping plane: $N.P + D = 0$, where $N = (A, B, C)$

Substituting, $N.(P_1 + (P_2 - P_1)u) + D = 0$

$$u = \frac{-D - N.P_1}{N.(P_2 - P_1)}$$

Window to Viewport Transformation

Windows are areas on screen where graphical information can be displayed

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

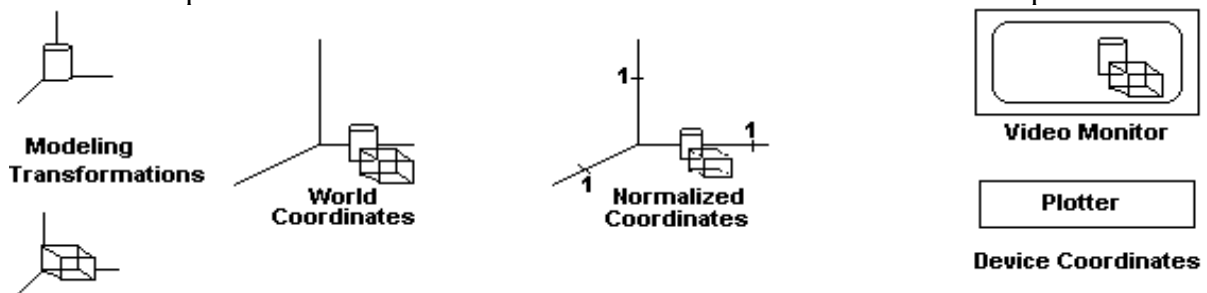
View ports are or can be of different sizes but are always smaller than size of window.

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 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 referred 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 object in space

The coordinates so defined are world coordinates

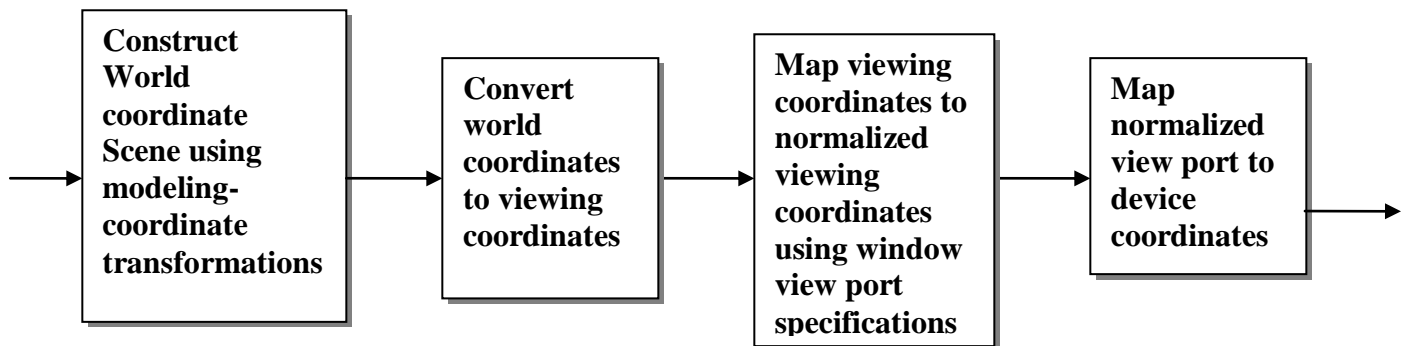
It may sometimes be desirable to select a part of object or drawing for display in order to obtain sufficient detail of the object on display

The part of interest is enclosed in 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)

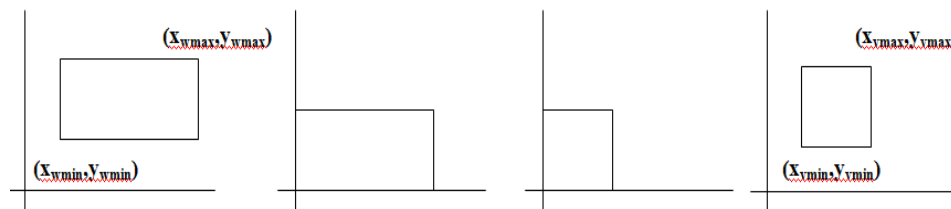
$$x_v = x_{vmin} + (x_w - x_{wmin}) \frac{(x_{vmax} - x_{vmin})}{(x_{wmax} - x_{wmin})} = x_{vmin} + (x_w - x_{wmin}) \cdot s_x$$

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

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



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 device are only 2D

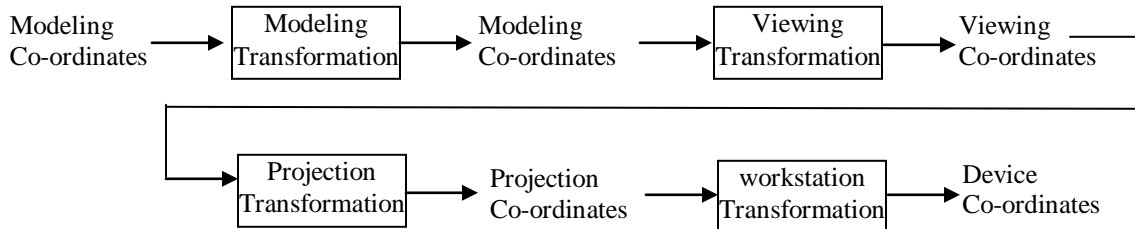
Solution is accomplished by introducing projections that transform 3D objects onto 2D plane

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

3D Viewing pipeline:

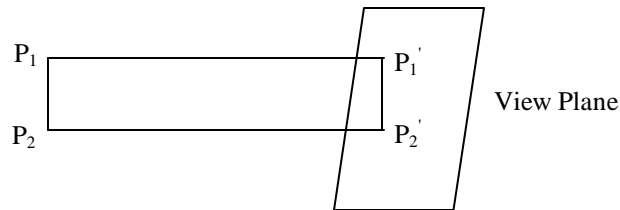
The steps for computer generation of a view of 3D scene are analogous to the process of taking photograph by a camera. For a snapshot, we need to position the camera at a particular point in space and then need to decide camera orientation. Finally when we snap the shutter, the seen is cropped to the size of window of the camera and the light from the visible surfaces is projected into the camera film.



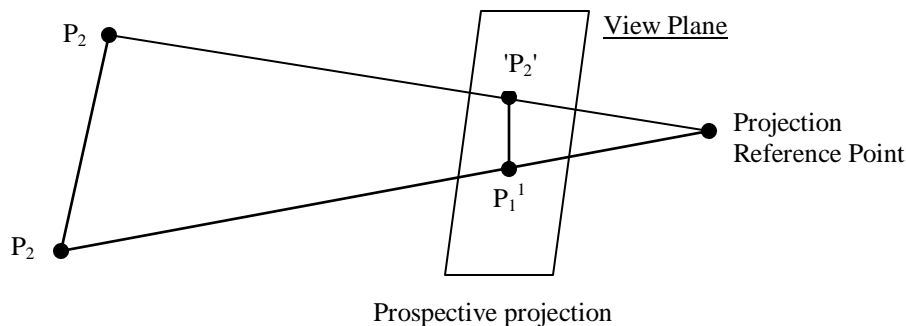
Projections:

Once world co-ordinate description of the objects in a scene are converted to viewing co-ordinates, we can project the three dimensional objects onto the two dimensional view plane. There are two basic projection methods:

Parallel projection: In parallel projection, co-ordinates positions are transformed to the view plane along parallel lines.



Prospective projection: In prospective projection, objects positions are transformed to the view plane along lines that converge to a point called projection reference point (centre of projection). The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.

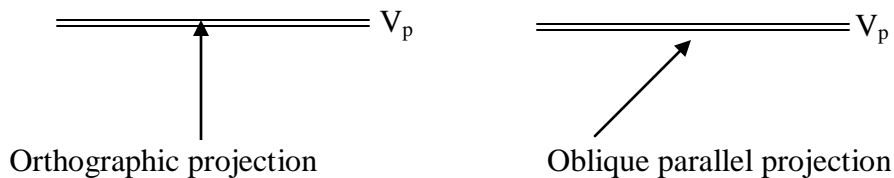


A parallel projection preserve relative proportions of objects and this is the method used in drafting in drafting to produce scale drawing of three-dimensional objects. Accurate view of various sides of 3D object is obtained with parallel projection. But it does not given a realistic appearance of a 3D-object.

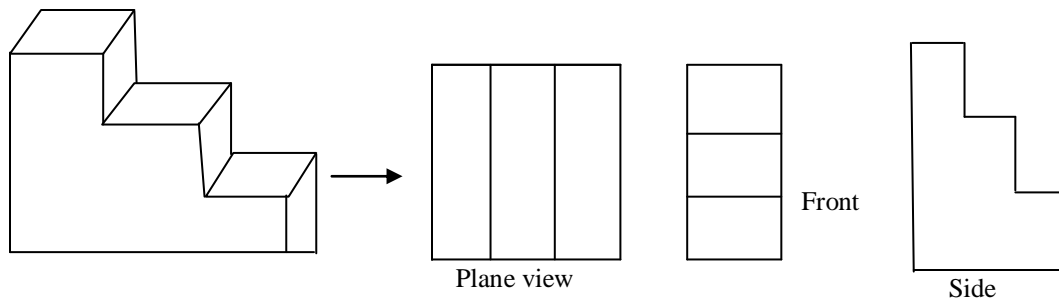
A perspective projection, on the other hand, produces realistic views but does not preserve relative proportions. Projections of distance objects from view plane are smaller than the projections of objects of the same size that are closer to the projection place.

Parallel Projection: We can specify parallel projection withy projection vector that specifies the direction of projection line. When the projection lines are perpendicular to view plane, the projection is orthographic parallel projections.

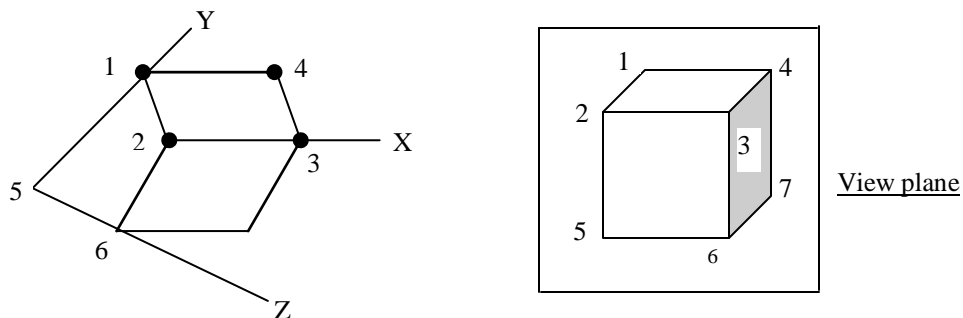
It projection line are not parallel to view plane then it is oblique parallel projection.



- Orthographic projections are most often used to produce the front, side, and top views of an object. Front, side and rear orthographic are called elevations and the top orthographic view of object is known as plan view. Engineering and Architectural drawings commonly employ these orthographic projections.



We also from orthographic projections that display more than one face of an object. Such views are called axonometric orthographic projections. the most commonly used axonometric projection is the isometric projection.



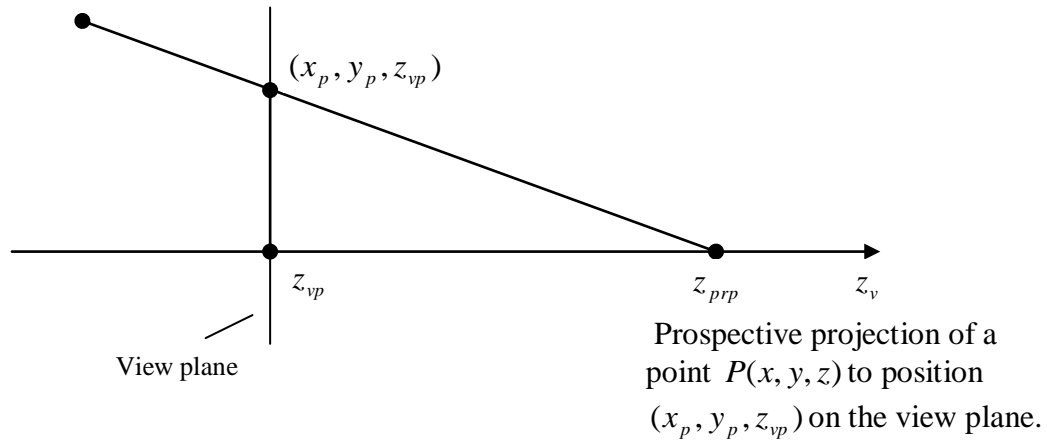
The transformation eq^n for orthographic projection is

$x_p = x$, $y_p = y$, z – Coordinate value is preserved for the depth information:

Prospective projections:

To obtain a prospective projection of a three-dimensional object, we transform points along projection lines that meet at a point called projection reference point.

Suppose we set the projection reference point at position Z_{prp} along the Z_v axis, and we place the view plane at Z_{vp} as shown in fig



We can write equations describing co-ordinates positions along this prospective projection line in parametric form as

$$x' = x - xu$$

$$y' = y - yu$$

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

Where u takes value from 0 to 1. If $u = 0$, we are at position $P = (x, y, z)$. If $u = 1$, we have projection reference point $(0, 0, z_{prp})$. On the view plane, $z' = z_{vp}$ then

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

Substituting these value in eq^n for x' , y' .

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

Similarly,

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

Where $dp = z_{prp} - z_{vp}$ is the distance of the view plane from projection reference point.

Using 3-D homogeneous Co-ordinate representation, we can write perspective projection transformation matrix as

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & z_{vp}/dp & -z_{vp}/dp \\ 0 & 0 & 1/dp & z_{prp}/dp \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In this representation the homogeneous factor $h = \frac{z - z_{prp}}{dp}$

Projection co-ordinates,

$$xp = xh/h, yp = yh/h$$

There are special case for perspective transformation.

When $z_{prp} = 0$;

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

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

Some graphics package, the projection point is always taken to be viewing co-ordinate origin. In this ease, $z_{prp} = 0$

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

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

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 3 space as well.

Thus instead of representing a point as (x, y, z), we represent it as (x, y, z, H), where two these quadruples represent the same point if 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 $(x/H, y/H, z/H, 1)$.

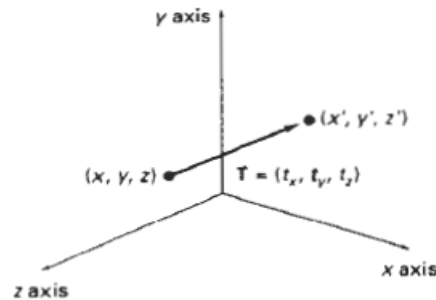
Transforming the point to this form is called homogenizing.

Translation:

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:

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

Scaling:

Scaling changes size of an object and repositions the object relative to the coordinate origin.

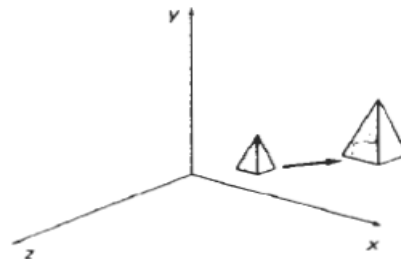
If transformation parameters are not all equal then figure gets distorted

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 :

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

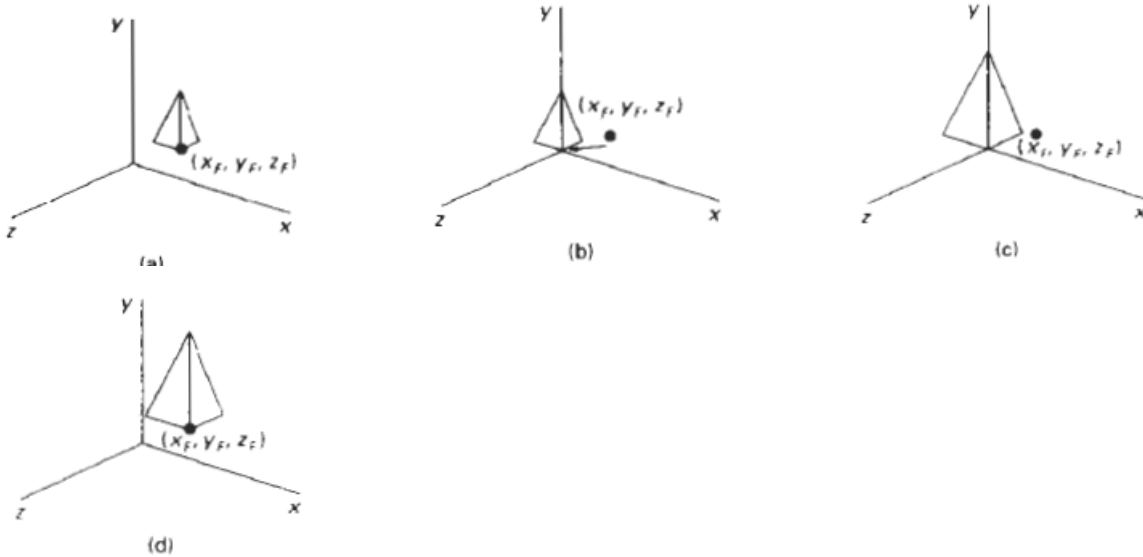
or $P' = S . P$



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

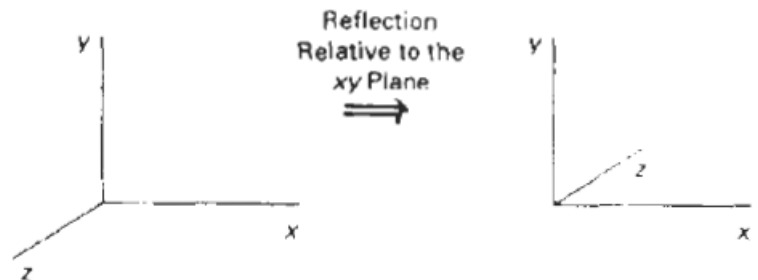
or $T(x_f, y_f, z_f) \cdot S(s_x, s_y, s_z) \cdot T(-x_f, -y_f, -z_f)$



Reflection:

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

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



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

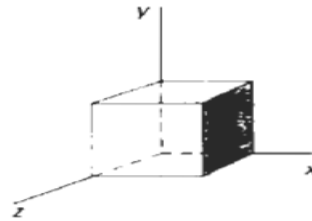
Transformation matrices for inverting x and y values are defined similarly, as reflections relative to the yz plane and xz plane.

Shearing:

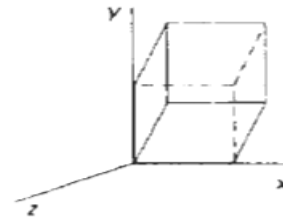
Shearing transformations are used to modify object shapes.

E.g. shears relative to the z axis:

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



(a)



(b)

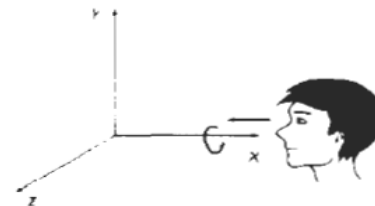
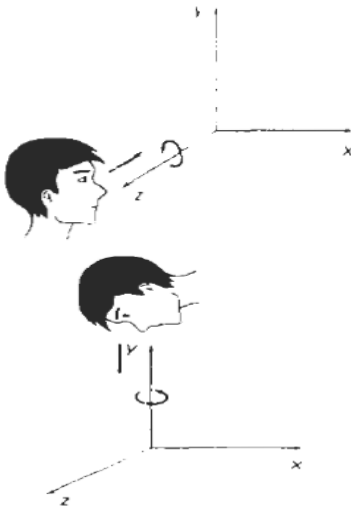
where parameters a and b assume any real values.

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 .



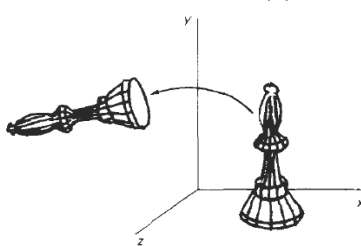
Axes that are parallel to the coordinate axes are easy to handle.

Coordinate axes Rotations:

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

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

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

$$\begin{matrix} \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \end{matrix} \quad P' = R_z(\theta) \cdot P$$


or,

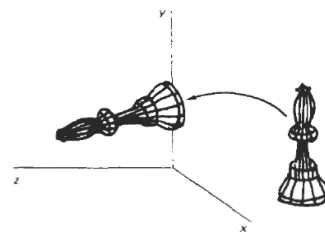
Cyclic permutation of the coordinate parameters x, y and z are used to get transformation equations for rotations about the other two coordinates

$x \rightarrow y \rightarrow z \rightarrow$ so,

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

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

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

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


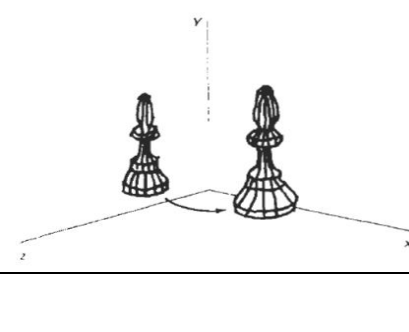
or,

$$P' = R_x(\theta) \cdot P$$

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

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

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

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


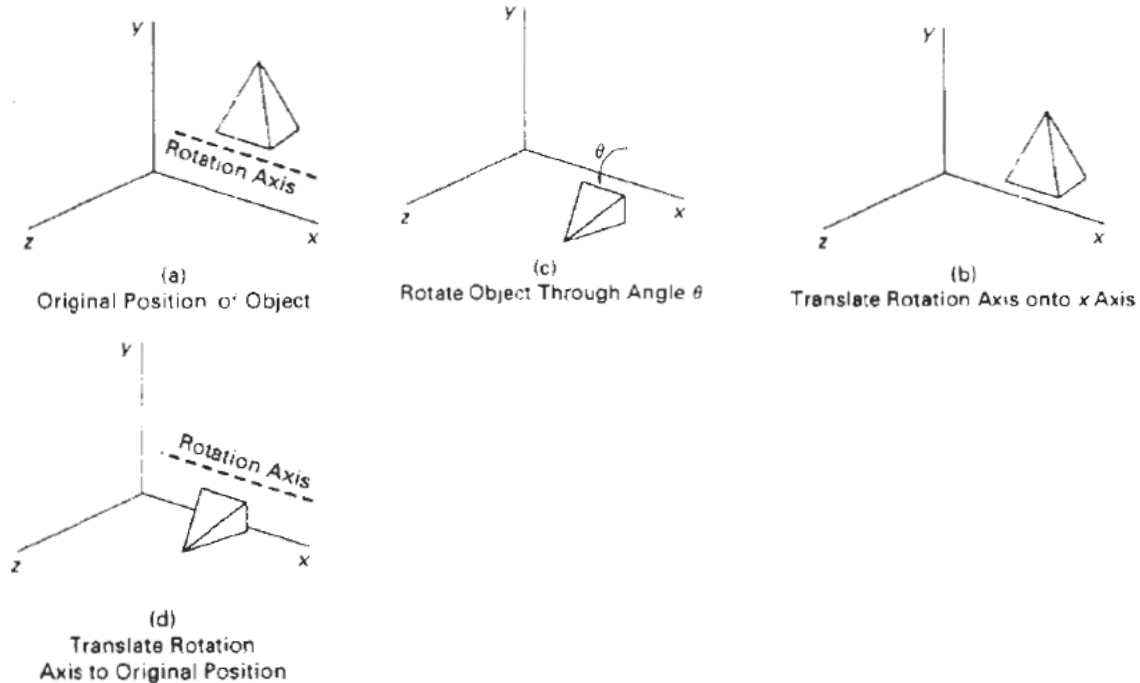
$$P' = R_y(\theta) \cdot P$$

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

Steps:

- Translate object so that rotation axis coincides with the parallel coordinate axis.
- Perform specified rotation about that axis
- Translate object back to it's original position.

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

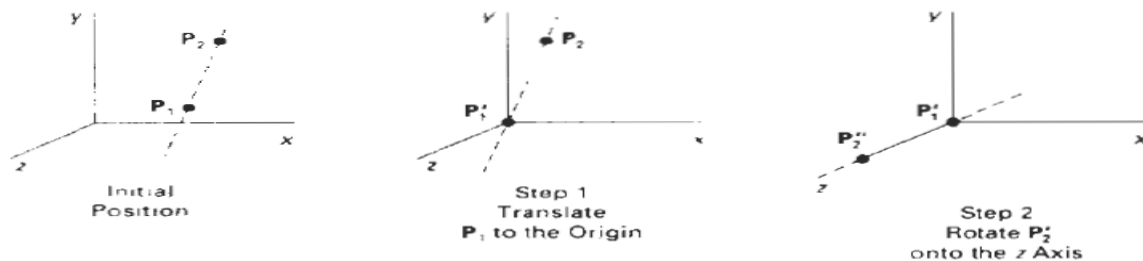


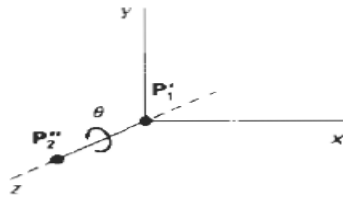
Rotation about an arbitrary axis :

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

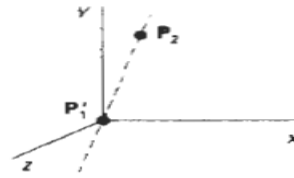
Steps required for rotation about this axis by some angle θ are:

- Translate so that point (x_0, y_0, z_0) is at the origin of the coordinate system
- Perform rotations to make axis of rotation coincident with the z axis
- Rotate about the z axis by the angle θ .
- Perform the inverse of the combined rotation transformation
- Perform the inverse of the translation.

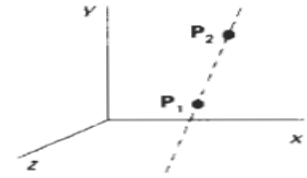




Step 3
Rotate the
Object Around the
z Axis



Step 4
Rotate the Axis
to the Original
Orientation



Step 5
Translate the
Rotation Axis
to the Original
Position

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,

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

so, transformation matrix for rotation about x axis is:

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

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

$$\cos = d \quad \sin = c_x$$

so, transformation matrix for rotation about y axis is:

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

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,

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

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,

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

Reflection thru an arbitrary plane :

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 through the $z = 0$ coordinate plane.

i.e.

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

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

So the general transformation matrix is:

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

3D Object representations

Graphical scenes can contain many different kinds of objects like trees, flowers, rocks, waters...etc. There is no one method that we can use to describe objects that will include all features of those different materials. To produce realistic display of scenes, we need to use representations that accurately model object characteristics.

- Simple Euclidean objects like polyhedrons and ellipsoids can be represented by polygon and quadric surfaces.
- Spline surface are useful for designing aircraft wings, gears and other engineering objects.
- Procedural methods and particle systems allow us to give accurate representation of clouds, clumps of grass, and other natural objects.
- Octree encodings are used to represent internal features of objects. Such as medical CT images.

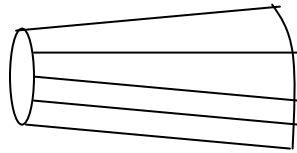
Representation schemes for solid objects are often divided into two broad categories:

1. **Boundary representations**: describes a 3D object as a set of polygonal surfaces, separate the object interior from environment.
2. **Space-partitioning representation**: used to describe interior properties, by partitioning the spatial region, containing an object into a set of small, non overlapping, contiguous solids. e.g. 3D object as Octree representation.

Boundary Representation: Each 3D object is supposed to be formed its surface by collection of polygon facets and spline patches. Some of the boundary representation methods for 3D surface are:

1. Polygon Surfaces: It is the most common representation for 3D graphics object. In this representation, a 3D object is represented by a set of surfaces that enclose the object interior. Many graphics system use this method. Set of polygons are stored for

object description. This simplifies and speeds up the surface rendering and display of object since all surfaces can be described with linear equations.



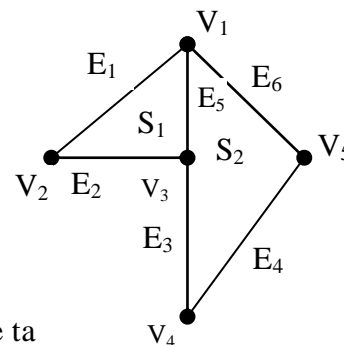
A 3D object represented by polygons

The polygon surface are common in design and solid-modeling applications, since wire frame display can be done quickly to give general indication of surface structure. Then realistic scenes are produced by interpolating shading patterns across polygon surface to illuminate.

Polygon Table: A polygon surface is specified with a set of vertex co-ordinates and associated attribute parameters. A convenient organization for storing geometric data is to create 3 lists:

- A vertex table
 - An edge table
 - A polygon surface table.
- Vertex table stores co-ordinates of each vertex in the object.
 - The edge table stores the Edge information of each edge of polygon facets.
 - The polygon surface table stores the surface information for each surface i.e. each surface is represented by edge lists of polygon.

Consider the surface contains polygonal facets as shown in figure (only two polygon are taken here)



→ S_1 and S_2 are two polygon surface that represent the boundary of some 3D object.

For storing geometric data, we can use following three tables

VERTEX TABLE
$V_1: x_1, y_1, z_1$
$V_2: x_2, y_2, z_2$
$V_3: x_3, y_3, z_3$
$V_4: x_4, y_4, z_4$
$V_5: x_5, y_5, z_5$

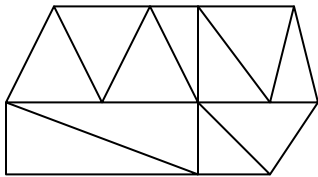
EDGE TABLE
$E_1: V_1, V_2$
$E_2: V_2, V_3$
$E_3: V_3, V_4$
$E_4: V_4, V_5$
$E_5: V_1, V_3$
$E_6: V_5, V_1$

POLYGON SURFACE TABLE
$S_1: E_1, E_2, E_3$
$S_2: E_3, E_4, E_5, E_6$

The object can be displayed efficiently by using data from tables and processing them for surface rendering and visible surface determination.

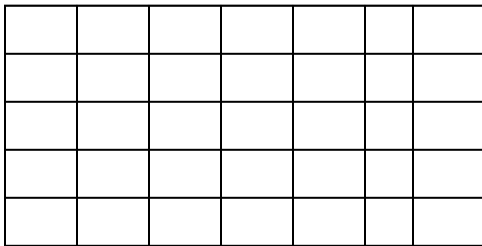
Polygon Meshes: A polygon mesh is collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons. An edge connects two vertices and a polygon is a closed sequence of edges. An edge can be shared by two polygons and a vertex is shared by at least two edges.

When object surface is to be tiled, it is more convenient to specify the surface facets with a mesh function. One type of polygon mesh is triangle strip. This function produce $n-2$ connected triangles.



Triangular Mesh

Another similar function is the quadrilateral mesh, which generates a mesh of $(n-1)$ by $(m-1)$ quadrilaterals, given the co-ordinates for an $n \times m$ array of vertices.



6 by 8 vertices array , 35
element quadrilateral mesh

- If the surface of 3D object is planer, it is comfortable to represent surface with meshes.

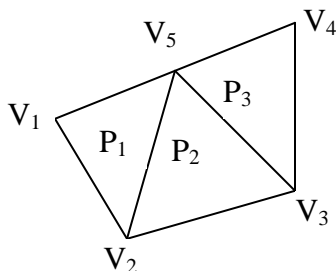
Representing polygon meshes

In explicit representation, each polygon is represented by a list of vertex co-ordinates.

$$P = ((x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n))$$

The vertices are stored in order traveling around the polygon. There are edge between successive vertices in the list and between the last and first vertices.

- For a single polygon it is efficient but for polygon mesh it is not space efficient since no of vertices may duplicate.
- So another method is to define polygon with pointers to a vertex list. So each vertex is stored just once, in vertex list $V = \{v_1, v_2, \dots, v_n\}$ A polygon is defined by list of indices (pointers) into the vertex list e.g. A polygon made up of vertices 3,5,7,10 in vertex list be represented as $P_1 = \{3,5,7,10\}$



➤ Representing polygon mesh with each polygon as vertex list.

- $P_1 = \{v_1, v_2, v_5\}$
- $P_2 = \{v_2, v_3, v_5\}$
- $P_3 = \{v_3, v_4, v_5\}$

Here most of the vertices are duplicated so it is not efficient.

➤ Representation with indexes into a vertex list

$$V = \{v_1, v_2, v_3, v_4, v_5\} = \{(x_1, y_1, z_1), \dots, (x_5, y_5, z_5)\}$$

$$P_1 = \{1, 2, 3\}$$

$$P_2 = \{2, 3, 5\}$$

$$P_3 = \{3, 4, 5\}$$

➤ **Defining polygons by pointers to an edge list**

In this method, we have vertex list V, represent the polygon as a list of pointers not to the vertex list but to an edge list. Each edge in edge list points to the two vertices in the vertex list. Also to one or two polygon, the edge belongs.

Hence we describe polygon as

$$P = (E_1, E_2, \dots, E_n)$$

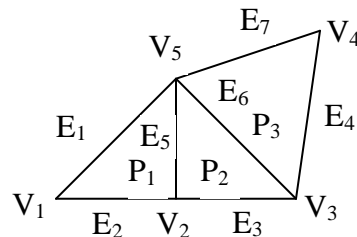
and an edge as

$$E = (V_1, V_2, P_1, P_2)$$

2Here if edge belongs to only one polygon, either

Then P_1 or P_2 is null.

For the mesh given below,



$$V = \{v_1, v_2, v_3, v_4, v_5\} = \{(x_1, y_1, z_1), \dots, (x_5, y_5, z_5)\}$$

$$E_1 = (V_1, V_5, P_1, N)$$

$$E_2 = (V_1, V_2, P_1, N)$$

$$E_3 = (V_2, V_3, P_2, N)$$

$$E_4 = (V_3, V_4, P_3, N)$$

$$E_5 = (V_2, V_5, P_1, P_2)$$

$$E_6 = (V_3, V_5, P_1, P_3)$$

Here N represents Null.

$$E_7 = (V_4, V_5, P_3, N)$$

$$P_1 = (E_1, E_2, E_3)$$

$$P_2 = (E_3, E_6, E_5)$$

$$P_3 = (E_4, E_7, E_6)$$

Polygon mesh defined with edge lists for each polygon.

3D-object representation

1. Polygon Surface: Plane Equation Method

Plane equation method is another method for representation the polygon surface for 3D object. The information about the spatial orientation of object is described by its individual surface, which is obtained by the vertex co-ordinates and the equation of each surface. The equation for a plane surface can be expressed in the form,

$$Ax + By + Cz + D = 0$$

Where (x,y,z) is any point on the plane, and A,B,C,D are constants describing the spatial properties of the plane. The values of A,B,C,D can be obtained by solving a set of three plane equations using co-ordinate values of 3 non collinear points on the plane.

Let (x_1, y_1, z_1) , (x_2, y_2, z_2) and (x_3, y_3, z_3) are three such points on the plane, then-

$$Ax_1 + By_1 + Cz_1 + D = 0$$

$$Ax_2 + By_2 + Cz_2 + D = 0$$

$$Ax_3 + By_3 + Cz_3 + D = 0$$

The solution of these equations can be obtained in determinant from using Cramer's rule as:-

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_1 & y_2 & 1 \\ x_1 & y_3 & 1 \end{vmatrix} \quad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_1 & y_2 & z_2 \\ x_1 & y_3 & z_3 \end{vmatrix}$$

For any points (x,y,z)

If $Ax + By + Cz + D \neq 0$, then (x,y,z) is not on the plane.

If $Ax + By + Cz + D < 0$, then (x,y,z) is inside the plane i. e. invisible side

If $Ax + By + Cz + D > 0$, then (x,y,z) is lies out side the surface.

2. Wireframe Representation:

In this method 3D objects are represented as a list of straight lines, each of which is represented by its two end points (x_1, y_1, z_1) and (x_2, y_2, z_2) . This method only shows the skeletal structure of the objects.

It is simple and can see through the object and fast method. But independent line data structure is very inefficient i.e. don't know what is connected to what. In this method the scenes represented are not realistic.

3. Blobby Objects:

Some objects don't maintain a fixed shape but change their surface characteristics in certain motions or when proximity to another objects e.g. molecular structures, water droplets, other liquid effects, melting objects, muscle shaped in human body etc. These objects can be described as exhibiting "blobbiness" and are referred as blobby objects.

Several models have been developed for representing blobby objects as distribution functions over a region of space. One way is to use Gaussian density function or bumps. A surface function is defined as:

$$f(x, y, z) = \sum_k b_k e^{-a_k r_k^2} - T = 0$$

Where $r_k = \sqrt{x_k^2 + y_k^2 + z_k^2}$, T = Threshold and a and b are used to adjust amount of blobbiness.

Other method for generating for generating blobby objects uses quadratic density function as:

$$f(x) = \begin{cases} b(1 - \frac{3r^3}{d^2}) \\ \frac{3}{2}b(1 - \frac{r}{d})^2 \\ 0 \end{cases}$$

Advantages

- Can represent organic, blobby or liquid line structures
- Suitable for modeling natural phenomena like water, human body
- Surface properties can be easily derived from mathematical equations.

Disadvantages:

- Requires expensive computation
- Requires special rendering engine
- Not supported by most graphics hardware

4. Spline Representation

A Spline is a flexible strips used to produce smooth curve through a designated set of points. A curve drawn with these set of points is spline curve. Spline curves are used to model 3D object surface shape smoothly.

Mathematically, spline are described as pice-wise cubic polynomial functions. In computer graphics, a spline surface can be described with two set of orthogonal spline curves. Spline is used in graphics application to design and digitalize drawings for storage in computer and to specify animation path. Typical CAD application for spline includes the design of automobile bodies, aircraft and spacecraft surface etc.

Interpolation and approximation spline

- Given the set of control points, the curve is said to interpolate the control point if it passes through each points.
- If the curve is fitted from the given control points such that it follows the path of control point without necessarily passing through the set of point, then it is said to approximate the set of control point.

Cubic spline:

It is most often used to set up path for object motions o tot provide a representation for an existing object or drawing. To design surface of 3D object any spline curve can be represented by piece-wise cubic spline.

Cubic polynomial offers a reasonable compromise between flexibility and speed of computation. Cubic spline requires less calculations with comparison to higher order polynomials and require less memory. Compared to lower order polynomial cubic spline are more flexible for modeling arbitrary curve shape.

Given a set of control points, cubic interpolation spines are obtained by fitting the input points with a picewise cubic polynomial curve that passes through every control points.

Suppose we have $n+1$ control points specified with co-ordinates.

$$p_k = (x_k, y_k, z_k), \quad k = 0, 1, 2, 3, \dots, n$$

A cubic interpolation fit of those points is

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of parametric equations.

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \quad (0 \leq u \leq 1)$$

There are three equivalent methods for specifying a particular spline representation.

1. Set of boundary conditions.

For the parametric cubic polynomial for the x-coordinate along the path of spline section

$$x(u) = a_x U^3 + b_x U^2 + c_x U + d \quad 0 \leq U \leq 1$$

Boundary condition for this curve be the set on the end point coordinate $x(0)$ and $x(1)$ and in the first derivatives at end points $x'(0)$ and $x'(1)$. These four boundary condition are sufficient to determine the four coefficient a_x, b_x, c_x, d_x .

2. From the boundary condition, we can obtain the characterizing matrix for spline. Then the parametric equation can be written as-

$$x(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix}$$

3. Blending function that determines how specified geometric constraints on the curve are combined to calculate position along the curve path.

$$x(u) = \sum_{k=0}^3 g_k \cdot BF_k(u)$$

Where g_k are the geometric constraint parameters such as control points co-ordinate and slope of the curve at control point. $BF_k(u)$ are the polynomial Blending functions.

Bezier curve and surface

This is spline approximation method, developed by the French Engineer Pierre Bezier for use in the design of automobile body. Beizer spline has a number of properties that make them highly useful and convenient for curve and surface design. They are easy to implement. For this reason, Bezier spline is widely available in various CAD systems.

In general Bezier curve can be fitted to any number of control points. The number of control points to be approximated and their relative position determine the degree of Bezier polynomial. The Bezier curve can be specified with boundary condition, with characterizing matrix or blending functions. But for general blending function specification is most convenient. Suppose we have $n+1$ control points: $p_k(x_k, y_k, z_k)$, $k = 0, 1, 2, 3, 4, \dots, n$. These co-ordinate points can be blended to produce the following position vector $p(u)$ which describes path of an approximating Bezier polynomial function p_0 and p_n .

$$p(u) = \sum_{k=0}^n p_k \cdot BEZ_{k,n}(u), \quad 0 \leq u \leq 1$$

The Bezier belending function $BEZ_{k,n}(u)$ are the Bernstein polynomial as,

$$BEZ_{k,n}(u) = c(n,k)u^k(1-u)^{n-k}$$

The vector equation (1) represents a set of three parametric equation for individual curve conditions.

$$\begin{aligned} \text{i.e.} \quad x(u) &= \sum_{k=0}^n x_k \cdot BEZ_{k,n}(u) \\ y(u) &= \sum_{k=0}^n y_k \cdot BEZ_{k,n}(u) \end{aligned}$$

$$z(u) = \sum_{k=0}^n z_k \cdot BEZ_{k,n}(u)$$

Bezier curve is a polynomial of degree one less than control points i.e. 3 points generate parabola, 4 points a cubic curve and so on.

Properties of Bezier Curve:

1. It always pass through initial and final control points. i.e $p(0) = p_0$ and $p(1) = p_n$.
2. Values of the parametric first derivatives of a Bezier curve at the end points can be calculated from control points as-

$$p'(0) = -np_0 + np_1$$

$$p'(1) = -np_{n-1} + np_n$$
3. The slope at the beginning of the curve is along the line joining the first two points and slope at the end of curve is along the line joining last two points.
4. Parametric second derivative at a Bezier curve at end points are-

$$p''(0) = n(n-1)[(p_2-p_1) - (p_1-p_0)]$$

$$p''(1) = n(n-1)[(p_{n-2}-p_{n-1}) - (p_{n-1}-p_n)]$$

Quadric Surface

Quadric Surface is one of the frequently used 3D objects surface representation. The quadric surface can be represented by a second degree polynomial. This includes:

1. Sphere: For the set of surface points (x,y,z) the spherical surface is represented as:

$$x^2 + y^2 + z^2 = r^2$$
, with radius r and centered at co-ordinate origin.
2. Ellipsoid: $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$, where (x,y,z) is the surface points and a,b,c are the radii on X,Y and Z directions respectively.
3. Elliptic paraboloid: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = z$
4. Hyperbolic paraboloid: $\frac{x^2}{a^2} - \frac{y^2}{b^2} = z$
5. Elliptic cone: $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$
6. Hyperboloid of one sheet: $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$
7. Hyperboloid of two sheet: $\frac{x^2}{a^2} - \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$

Octree Representation: (Solid-object representation)

This is the space-partitioning method for 3D solid object representation. This is hierarchical tree structures (octree) used to represent solid object in some graphical system. Medical imaging and other applications that require displays of object cross section commonly use this method. E.g.: CT-scan

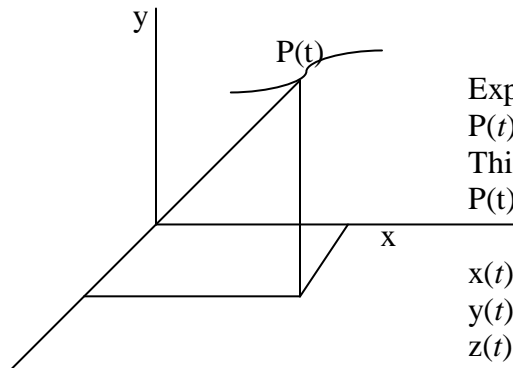
It provides a convenient representation for storing information about object interiors.

An octree encoding scheme divides region of 3D space into octants and stores 8 data elements in each node of the tree. Individual elements are called volume element or voxels. When all voxels in an octant are of same type, this type value is stored in corresponding data elements. Any heterogeneous octants are subdivided into octants again.

Parametric Cubic Curve

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

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



Expanding equation (i) yields

$$P(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0 \text{ -----(ii)}$$

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

$$\begin{aligned} x(t) &= a_{3x} t^3 + a_{2x} t^2 + a_{1x} t + a_{0x} \\ y(t) &= a_{3y} t^3 + a_{2y} t^2 + a_{1y} t + a_{0y} \\ z(t) &= a_{3z} t^3 + a_{2z} t^2 + a_{1z} t + a_{0z} \text{ -----(iii)} \end{aligned}$$

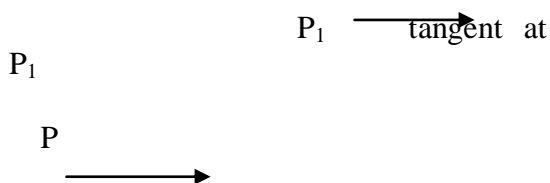
z

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

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

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

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

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

$$\begin{aligned} a_0 &= P(0) & a_1 &= P'(0) \\ a_2 &= -3P(0) - 3P(1) - 2P'(0) - P'(1) & a_3 &= 2P(0) - 2P(1) + P'(0) + P'(1) \end{aligned}$$

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

Solid Modeling :

representation of solid objects unambiguously. Represent only one object

Solid representation

Divide Euclidean space into two regions interior and exterior to it separated from each other by the boundary of the object

Properties of Solid models :

Bounded boundary limits and contain the interior of the solid

Homogenously 3D no dangling edges or faces presented boundary is always in contact with the interior of the solid

Finite solid is not infinite in size can be described by a limited amount of information

Basics of solid modeling theory

Geometry , topology,

(x,y,z) coordinates of vertices : geometry

Connectivity matrix: topology

Geometric closure,

Set theory and Operations ,

A set is a collection of objects, in the context of geometric representation the basic element of a set is a point. Two sets are commonly represented

W universal set containing all the elements of all sets

\emptyset null set containing no elements.

$A \subset B$, A is a subset of B

Inequality equality

Regularized Boolean set operations:

Union difference intersection etc

Set membership classification

Given two sets x and s as being on its interior, exterior or on its boundaries

x is partitioned into subsets x_{ins} , x_{ons} , x_{outs} inside on boundary or outside S

Boundary Representation

Idea: physical object is enclosed by a set of faces, which themselves belong to closed and orientable surfaces

Information on both geometric and topological elements is stored in the B-rep database

Euler's law gives a quantitative relationship among faces edges vertices faces etc

$$F - E + V - L = 2(B - G)$$

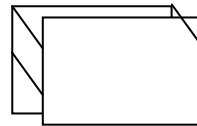
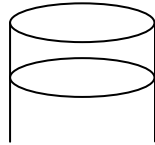
$$F - E + V = 2$$

Two vertices

Three edges

Three faces

$$F - E + V = 2$$



Sweep representation

Used for creating solids with uniform thickness in a particular direction and axisymmetric solids by translational and rotational sweeping

Sweeping requires

A surface to be moved and a trajectory (along which the movement should occur)

Recent Advancements In solid modeling

NURBS (Non Uniform Rational BSplines)

Solid modeler for handling free form surface definitions and not just polyhedral and quadric models

Visible Surface Detection Methods

(Hidden surface elimination)

Visible surface detection or Hidden surface removal is major concern for realistic graphics for identifying those parts of a scene that are visible from a chosen viewing position. Several algorithms have been developed. Some require more memory, some require more processing time and some apply only to special types of objects.

Visible surface detection methods are broadly classified according to whether they deal with objects or with their projected images.

These two approaches are

- **Object-Space methods:** Compares objects and parts of objects to each other within the scene definition to determine which surface as a whole we should label as visible.
- **Image-Space methods:** Visibility is decided point by point at each pixel position on the projection plane.

Most visible surface detection algorithm use image-space-method but in some cases object space methods are also used for it.

BACK-FACE DETECTION(Plane Equation method)

A fast and simple object space method used to remove hidden surface from a 3D object drawing is known as "Plane equation method" and applied to each side after any rotation of the object takes place. It is commonly known as back-face detection of a polyhedron is based on the "inside-outside" tests.

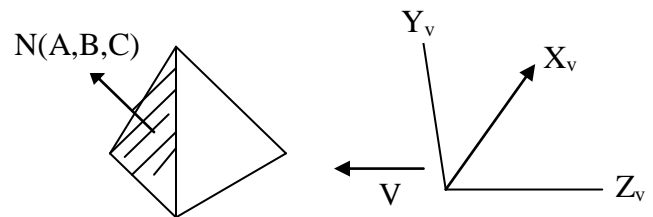
A point (x, y, z) is inside a polygon surface if

$$Ax + By + Cz + D < 0$$

We can simplify this test by considering the normal vector N to a polygon surface which has Cartesian components (A, B, C)

If V is the vector in viewing direction from the eye position then this polygon is a back face if,

$$V \cdot N > 0$$



In the equation $Ax + By + Cz + D = 0$, if A, B, C remains constant, then varying value of D results in a whole family of parallel planes. One of which ($D = 0$) contains the origin of the co-ordinates system and ,

If $D > 0$, plane is behind the origin(Away from observer)

If $D < 0$, plane is in front of origin(towards the observer)

If we clearly defined our object to have centered at origin, the all those surface that are viewable will have negative D and unviewable surface have positive D .

So , simply our hidden surface removal routine defines the plane corresponding to one of 3D surface from the co-ordinate of 3 points on it and computing D , visible surface are detected.

DEPTH-BUFFER-METHOD:

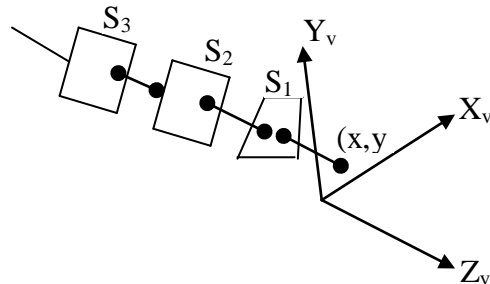
Depth Buffer Method is the commonly used image-space method for detecting visible surface. It is also know as z-buffer method. It compares surface depths at each pixel position on the projection plane. It is called z-buffer method since object depth is usually measured from the view plane along the z-axis of a viewing system.

Each surface of scene is processed separately, one point at a time across the surface. The method is usually applied to scenes containing only polygon surfaces, because depth values can be computed very quickly and method is easy to implement. This method can be apply to non planer surfaces.

With object description converted to projection co-ordinates, each (x, y, z) position on polygon surface corresponds to the orthographic projection point (x, y) on the view plane. Therefore for each pixel position (x, y) on the view plane, object depth is compared by z. values.

With objects description converted to projection co-ordinates, each (X, Y, Z) position on polygon surface correspond to the orthographic projection point (X, Y) on the view plane. The object depth is compared by Z-values.

In figure, three surface at varying distance from view plane $X_v Y_v$, the projection along (x, y) surface S_1 is closest to the view-plane so surface intensity value of S_1 at (x, y) is saved.



In Z-buffer method, two buffers area are required. A depth buffer is used to store the depth value for each (x, y) position or surface are processed, and a refresh buffer stores the intensity value for each position. Initially all the position in depth buffer are set to 0, and refresh buffer is initialize to background color. Each surface listed in polygon table are processed one scan line at a time, calculating the depth (z-val) for each position (x, y) . The calculated depth is compared to the value previously stored in depth buffer at that position. If calculated depth is greater than stored depth value in depth buffer, new depth value is stored and the surface intensity at that position is determined and placed in refresh buffer.

Algorithm: Z-buffer

1. Initialize depth buffer and refresh buffer so that for all buffer position (x, y)
depth $(x, y) = 0$, refresh $(x, y) = I_{\text{background}}$.

2. For each position on each polygon surface, compare depth values to previously stored value in depth buffer to determine visibility.

- Calculate the depth Z for each (x,y) position on polygon
- If $Z > \text{depth}(x,y)$ then
 $\text{depth}(x,y) = Z$
 $\text{refresh}(x,y) = I_{\text{surface}}(x,y)$

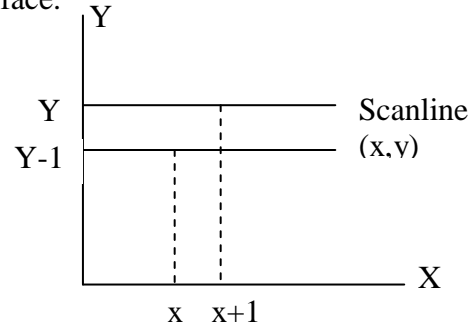
Where $I_{\text{background}}$ is the intensity value for background and $I_{\text{surface}}(x,y)$ is intensity value for surface at pixel position (x,y) on projected plane. After all surface are processed, the depth buffer contains the depth value of the visible surface and refresh buffer contains the corresponding intensity values for those surface. The depth value of the surface position (x,y) are calculated by plane equation of surface.

$$Z = \frac{-Ax - By - D}{C}$$

Let Depth Z' at position $(x+1,y)$

$$Z' = \frac{-A(x+1) - By - D}{C}$$

$$\Rightarrow Z' = Z - \frac{A}{C} \quad \text{--- (1)}$$

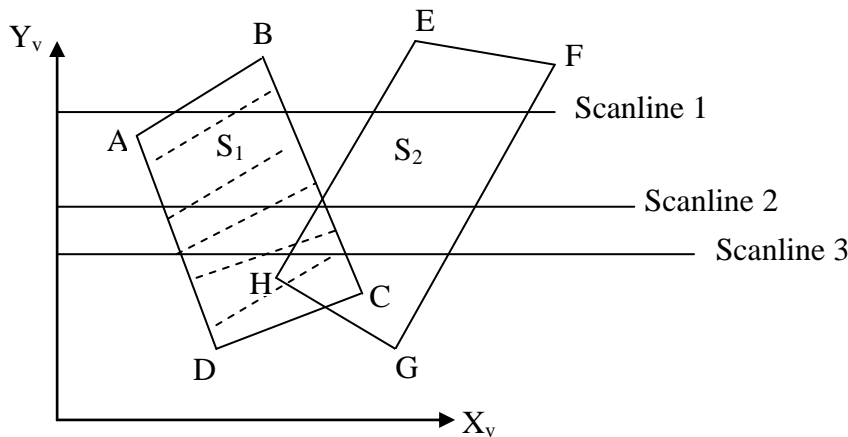


$-\frac{A}{C}$ is constant for each surface so succeeding depth value across a scan line are obtained from preceding values by simple calculation.

SAN LINE METHOD :

This is image-space method for removing hidden surface which is extension of the scan line polygon filling for polygon interiors. Instead of filling one surface we deal with multiple surface here.

As each scan line is processed, all polygon surface intersecting that line are examined to determine which are visible. We assume that polygon table contains the co-efficient of the plane equation for each surface as well as vertex edge, surface information, intensity information for the surface, and possibly pointers to the edge table.



- In figure above, the active edge list for scan line 1 contains information from edge table for edge AB, BC, EH, FG.
 - For positions along this scan line between edge AB and BC, only the flag for surface S_1 is on
 - Therefore no depth calculation must be made using the plane coefficients for two surface and intensity information for surface S_1 is entered from the polygon table into the refresh buffer.
- Similarly between EH&FG. Only the flag for S_2 is on. No other positions along scan line 1 intersect surface. So intensity values in the other areas are set to background intensity
- For Scanline 2 & 3, the active edge list contains edges AD, EH BC, FG. Along scanline 2 from edge AD, to edge EH only surface flag for S_1 is on, but between edges EH& BC, the flags for both surface is on. In this interval, depth calculation is made using the plane coefficients for the two surfaces.
- For example, if Z of surface S_1 is less than surface S_2 , So the intensity of S_1 is loaded into refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off and intensities for surface S_2 are stored until edge FG is passed.
- Any no of overlapping surface are processed with this scan line methods.

DEPTH SORTING METHOD:

This method uses both object space and image space method. In this method the surface representation of 3D object are sorted in of decreasing depth from viewer. Then sorted surface are scan converted in order starting with surface of greatest depth for the viewer.

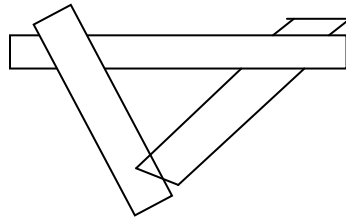
The conceptual steps that performed in depth-sort algorithm are

1. Sort all polygon surface according to the smallest (farthest) Z co-ordinate of each.
2. Resolve any ambiguity this may cause when the polygons Z extents overlap, splitting polygons if necessary.
3. Scan convert each polygon in ascending order of smaller Z-co-ordinate i.e. farthest surface first (back to front)

In this method, the newly displayed surface is partly or completely obscure the previously displayed surface. Essentially, we are sorting the surface into priority order such that surface with lower priority (lower z, far objects) can be obscured by those with higher priority (high z-value).

This algorithm is also called "Painter's Algorithm" as it simulates how a painter typically produces his painting by starting with the background and then progressively adding new (nearer) objects to the canvas.

Problem: One of the major problem in this algorithm is intersecting polygon surfaces. As shown in fig. below.



- Different polygons may have same depth.
- The nearest polygon could also be farthest.

We cannot use simple depth-sorting to remove the hidden-surfaces in the images.

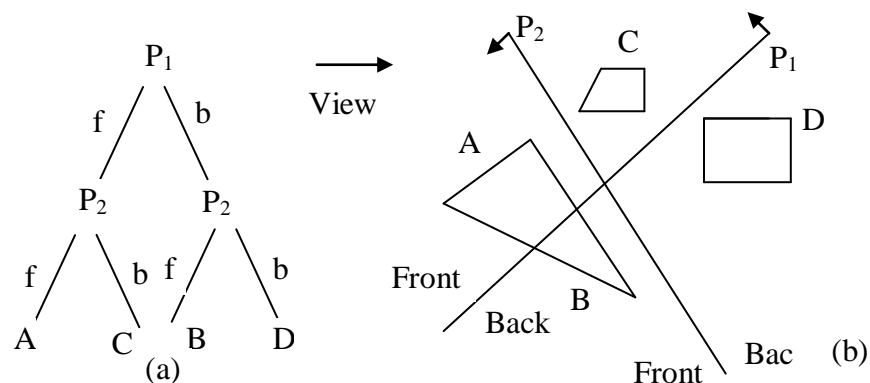
Solution: For intersecting polygons, we can split one polygon into two or more polygons which can then be painted from back to front. This needs more time to compute intersection between polygons. So it becomes complex algorithm for such surface existence.

BSP TREE METHOD:

A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm. The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position.

Applying a BSP tree to visibility testing involves identifying surfaces that are "inside" or "outside" the partitioning plane at each step of space subdivision relative to viewing direction. It is useful and efficient for calculating visibility among a static group of 3D polygons as seen from an arbitrary viewpoint.

In the following figure,

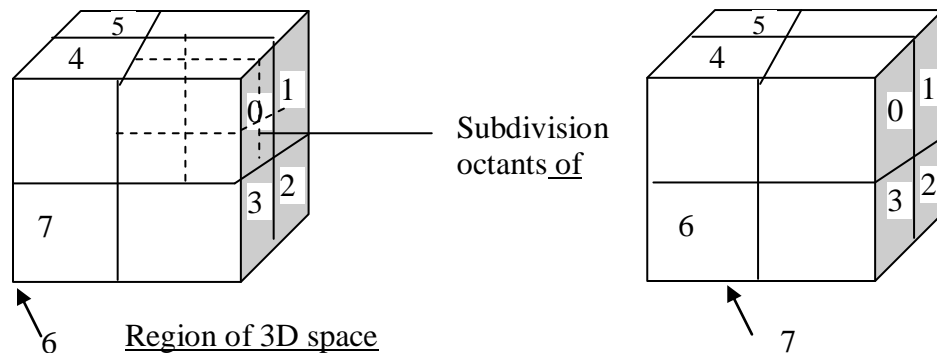


Here plane P_1 partitions the space into two sets of objects, one set of object is back and one set is in front of partitioning plane relative to viewing direction. Since one object is intersected by plane P_1 , we divide that object into two separate objects labeled A and B. Now object A&C are in front of P_1 , B and D are back of P_1 .

We next partition the space with plane P_2 and construct the binary tree as fig (a). In this tree, the objects are represented as terminal nodes, with front object as left branches and behind object as right branches.

When BSP tree is complete, we process the tree by selecting surface for displaying in order back to front. So foreground objects are painted over background objects.

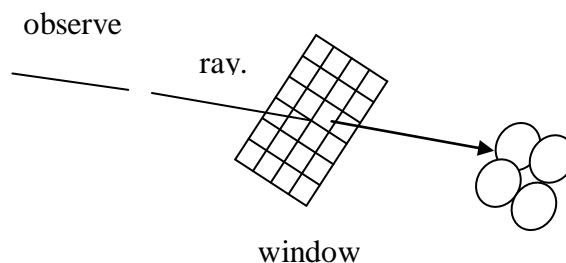
Octree Method: When an octree representation is used for viewing volume, hidden surface elimination is accomplished by projecting octree nodes into viewing surface in a front to back order. Following figure is the front face of a region space is formed with octants 0,1,2,3. Surface in the front of these octants are visible to the viewer. The back octants 4,5,6,7 are not visible. After octant sub-division and construction of octree, entire region is traversed by depth first traversal.



RAY TRACING:

Ray tracing also known as ray casting is efficient method for visibility detection in the objects. It can be used effectively with the object with curved surface. But it also used for polygon surfaces.

- Trace the path of an imaginary ray from the viewing position (eye) through viewing plane to object in the scene.
- Identify the visible surface by determining which surface is intersected first by the ray.
- Can be easily combined with lighting algorithms to generate shadow and reflection.
- It is good for curved surface but too slow for real time application.



Illumination Models and Surface Rendering Methods

Introduction:

Realistic displays of a scene are obtained by perspective projection and applying natural light effects to the visible surfaces an illumination model (lighting model) and sometimes called shading model, is used to calculate the intensity of light that we should see at a given point on the surface of an object.

A Surface Rendering Algorithm uses the intensity calculation from an illumination model to determine the light intensity for all projected pixel positions for various surfaces in a scene.

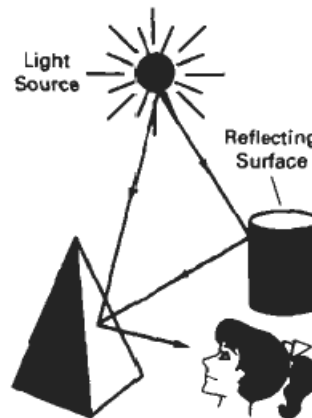
Light sources

Point source: tungsten filament bulb image based on the optical properties of surfaces, the background lighting conditions, and the can be seen

Distributed light source: fluorescent light

When light is incident on an opaque surface part of light is reflected part of light is absorbed and is dependent upon type of material. Surface that are rough or grainy tend to scatter reflected light in all direction is called diffuse reflection .

Light sources create highlights or bright spots called specular-reflection.



Basic Illumination Models

It describes about method of calculating light intensities. Lighting calculations are light source specifications. Optical parameters are used to set surface properties, such as glossy, matte, opaque, and transparent. This controls the amount of reflection and absorption of incident light. All light sources are considered to be point sources, specified with a coordinate position and an intensity value(color).

Ambient Light

Ambient light surface directly not exposed directly but visible if nearby objects are illuminated

Combination of light reflections from various surfaces to produce a uniform illumination called the ambient light or background light(no shadow's produced)

It has no spatial or direction characteristics and amount on each object is a constant for all surfaces and over all directions. But the intensity of reflected light for each surface depends on optical properties of the surface.

Diffuse reflection

Ambient light is an approximation of global diffuse, light effects .

Diffuse reflections are constant over each surface in a scene independent of viewing direction

k_d or diffuse reflection coefficient or diffuse reflectivity (0 to 1)

k_d is nearly 1 for highly reflective surface and k_d is 0 where light absorbs (black surfaces)

Diffuse reflection intensity at any point on the surface as

$$I_{\text{ambDiff}} = k_d \cdot I_d$$

Where I_{ambDiff} is ambient light due to diffusion and I_d is light due to diffusion assuming diffuse reflections from the surface are scattered with equal intensity in all directions independent of the

Viewing direction (called “ideal diffuse reflectors”) also called Lambertian reflectors and governed by Lambert’s Cosine Law.

If “angle of incidence” between incoming light direction and surface normal is θ

$$I_{\text{LDiff}} = k_d \cdot I_L \cos \theta$$

where I_{LDiff} is light due to diffusion

If N is unit normal vector to a surface and L is unit direction vector to the point light source then

$$I_{\text{LDiff}} = k_d \cdot I_L (N \cdot L)$$

In addition many graphics packages introduce an ambient reflection coefficient k_a to modify ambient light intensity k_a

then

$$I_{\text{Diff(i.e. total)}} = k_a \cdot I_a + k_d \cdot I_L (N \cdot L)$$

Specular Reflection and Phong Model

When we look at an illuminated shiny surface, such as polished metal, an apple etc we see a highlight or bright spot , at certain viewing direction this phenomenon is called “specular reflection” and is the result of total or near total reflection of the incident light in a concentrated region around the “specular reflection angle”.

This angle is equal to the angle of incidence.

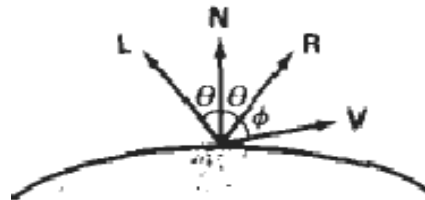
N – unit normal surface vector

R – unit vector in the direction of ideal specular reflection

L – unit vector directed towards point light source

V – unit vector pointing to viewer from the surface position

ϕ – viewing angle relative to specular reflection direction R



For ideal reflector (perfect mirror) incident light is reflected only in the specular reflection direction i.e. V and R coincide ($\theta = 0$).

Shiny surfaces have narrow θ and dull surfaces have wider θ

An empirical model for calculating specular reflection range was developed by Phong Bui Tuong called “Phong specular reflection” model and it sets the intensity of specular reflection directly proportional to $\cos^n \theta$ $\theta \rightarrow 0$ to 90

Specular reflection parameter n_s is determined by type of surface

Very shiny surface has large n_s value

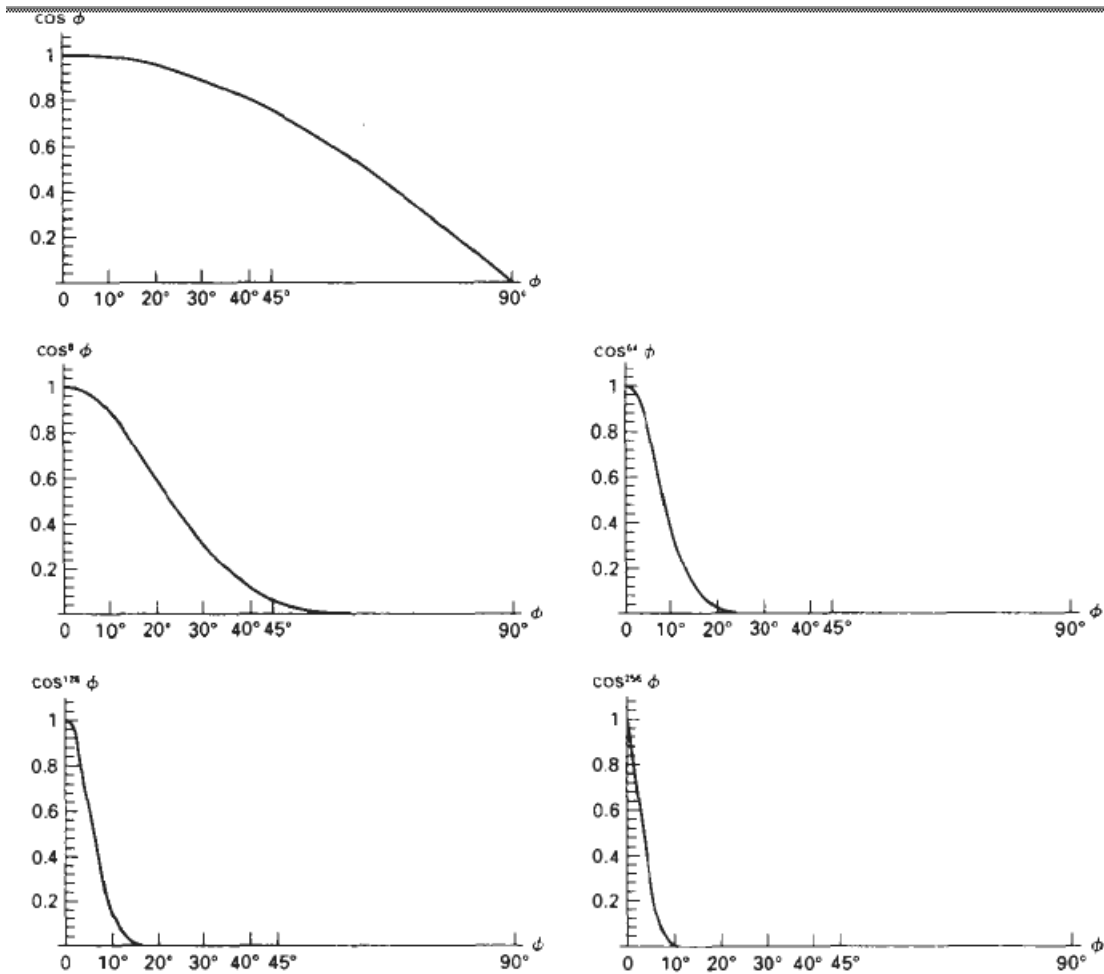
Very dull surface has smaller n_s value (down to 1)



Rough surface, e.g. chalk has $n_s = 1$

Intensity of specular reflection depends on material properties of surface, other factors such as polarization, color of incident light.

For monochromatic specular reflections intensity variations can be approximated by SR coefficient $W(\theta)$



$W(\theta)$ tends to increase as θ increases, at $\theta = 90^\circ$ $W(\theta) = 1$ and all incident light is reflected.

Fresnel's law of reflection describes specular reflection intensity with θ and using $W(\theta)$, Phong specular reflection model as

$$I_{\text{spec}} = W(\theta) I_L \cos^n \phi$$

where I_L is intensity of light source

ϕ is viewing angle relative to the specular reflection direction R .

So transparent materials like glass exhibit specular reflection as θ approaches 90° . At $\theta = 0^\circ$ about 4 percent of the incident light on a glass surface is reflected.

Now also $I_{\text{spec}} = W(\theta) I_L (V \cdot R)$ as $V \cdot R = \cos \phi$

R can be calculated in term of N and L

$$\mathbf{R} + \mathbf{L} = (2\mathbf{N} \cdot \mathbf{L})\mathbf{N} \quad \text{or} \quad \mathbf{R} = (2\mathbf{N} \cdot \mathbf{L})\mathbf{N} - \mathbf{L}$$

Simplified Phong model is obtained by halfway vector \mathbf{H} between \mathbf{L} and \mathbf{V} to calculate the range of SRs

Replacing $\mathbf{V} \cdot \mathbf{R}$ in equation with $\mathbf{N} \cdot \mathbf{H} \rightarrow \cos \phi$ replaced by $\cos \alpha$ Half way vector \mathbf{H}

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|}$$

If both viewer and light source are sufficiently far from surface both \mathbf{V} and \mathbf{L} are constant over the surfaces. Hence \mathbf{H} is constant.

For non Planar surfaces $\mathbf{N} \cdot \mathbf{H}$ requires less computation than $\mathbf{V} \cdot \mathbf{R}$

If \mathbf{V} is coplanar with \mathbf{L} and \mathbf{R} (also \mathbf{N}) then $\alpha = \phi / 2$

If $\mathbf{V}, \mathbf{L}, \mathbf{N}$ are non coplanar then $\alpha > \phi / 2$

Combined diffuse and specular reflections with multiple light sources

For single point light source

$$I = I_{\text{diffuse}} + I_{\text{spec}} = K_a I_a + K_d I_L (\mathbf{N} \cdot \mathbf{L}) + K_s I_L (\mathbf{N} \cdot \mathbf{H}) n_s$$

For multiple light sources

$$I = K_a I_a + \sum_{i=1}^n I_{Li} [(K_d (\mathbf{N} \cdot \mathbf{L}_i) + K_s (\mathbf{N} \cdot \mathbf{H}_i) n_s)]$$

Intensity Attenuation (I A)

Intensity is attenuated by the factor $1/d^2$ (d – distance that light has traveled)

Graphics package have a general inverse quadratic attenuation function

$$F(d) = \frac{1}{a_0 + a_1 d + a_2 d^2}$$

User can fiddle with the coefficients a_0, a_1, a_2 to obtain a variety of lighting effects for a scene. The a_0 can be adjusted to prevent $f(d)$ from becoming too large when d is very small. This is an effective method for limiting intensity values when a single light source is used to illuminate a scene.

Basic illumination model is modified as

$$I = k_a \cdot I_a + \sum_{i=1}^n f(d_i) I_{Li} [k_d (N \cdot L_i) + K_s (N \cdot H_i) n_s]$$

Color Considerations

To incorporate color, we need to write the intensity equation as a function of the color properties of the light sources and object surfaces.

Diffuse reflection coefficient vector for RGB component (K_{dR}, K_{dG}, K_{dB})

For blue reflectivity component ($K_{dR} = K_{dG} = 0$)

$$I_B = k_{aB} \cdot I_{aB} + \sum_{i=1}^n f_i(d) I_{LBi} [k_{dB} (N \cdot L_i) + K_{sB} (N \cdot H_i) n_s]$$

Surfaces are typically illuminated with white light sources, and in general we can set surface color so that the reflected light has nonzero values for all three RGB components. Calculated intensity levels for each color component can be used to adjust the corresponding electron gun in an RGB monitor.

In his original specular-reflection model, Phong set parameter K_s to a constant value independent of surface color. This produces specular reflections that are same color as the incident light (usually white), which gives the surface plastic appearance. For a non-plastic material, the color of the specular reflection is a function of surface properties and may be different from the color of the incident light and the color of the diffuse reflections.

Another method diffuse and specular color vector

$$I_B = k_a S_{dB} \cdot I_{aB} + \sum_{i=1}^n f_i(d) I_{LBi} [k_d S_{dB} (N \cdot L_i) + K_s S_{sB} (N \cdot H_i) n_s]$$

Color specification with its spectral wavelength

$$I = k_a S_d \cdot I_a + \sum_{i=1}^n f_i(d) I_{Li} [k_d S_d (N \cdot L_i) + K_s S_s (N \cdot H_i) n_s]$$

Transparency

A transparent surface in general produces both reflected and transmitted light. The relative contribution of the transmitted light depends on the degree of transparency of the surface and whether any light sources are behind the transparent surface. Both diffuse and specular transmission can take place at the surfaces of a transparent object.

θ_r = angle of refraction n_r - index of refraction

θ_i = angle of incidence n_i - index of incidence

using Snell's law

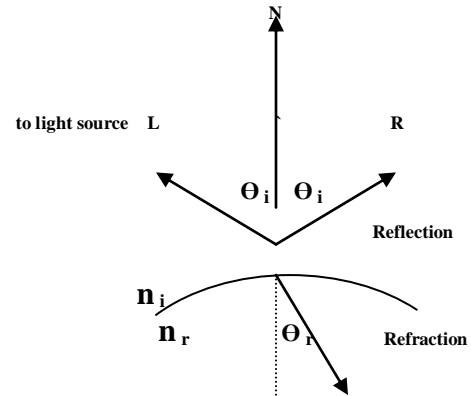
$$\frac{\sin \theta_r}{n_i} = \sin \theta_i$$

Direction

Also, $I = (1 - k_t)I_{\text{reflect}} + k_t I_{\text{trans}}$

Direction

Where, I_{trans} is transmitted intensity
 I_{reflect} is reflected intensity
 k_t transparency coefficient
 $(1 - k_t)$ is the opacity factor



For highly transparent object k_t is near 1

For opaque objects k_t is near 0

Shadows

Hidden surface method can be used to locate area where light sources produce shadows. By applying a hidden surface method with a light source at the view position, we can determine which surface sections cannot be seen from the light source.

Once we have determined the shadow area for all light sources, the shadows could be treated as surface patterns and store in pattern arrays . Shadow patterns generated by a hidden surface method are valid for any selected viewing position, as long as the light source positions are not changed.

6.3 Polygon Rendering Methods

Application of an illumination model to the rendering of the standard graphics objects those formed with polygon surfaces

The objects are usually polygon mesh approximation of curved surface objects but they may also be polyhedra that are not curved surface approximations

Scan line algorithms typically apply a lighting model to obtain polygon surface rendering in one or two ways each polygon can be rendered with a single intensity or the intensity can be obtained at each point of the surface using an interpolation scheme

A. Constant Intensity Shading

Fast and simple method for rendering of an object with polygon surfaces in CIS also called flat shading

Single intensity calculated for each polygon and useful for quickly displaying the general appearance of curved surface

This method is accurate if
the object is a polyhedron and is not an approximation of an object with a curved surface
all light sources illuminating the object are sufficiently far from the surface
the viewing position is sufficiently far from the surface so that V.R is constant over the surface

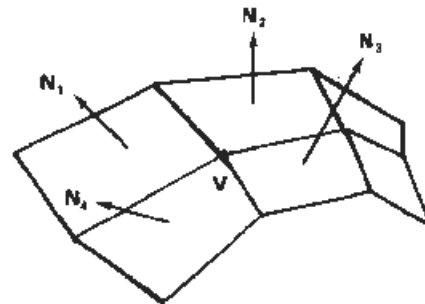
Even if all conditions are not true , we can still reasonable approximate surface lighting effects using small polygon facets with flat shading and calculate the intensity for each facet at the center of the polygon.

B. Gouraud Shading

This intensity interpolation scheme developed by Gouraud renders a polygon surface by linearly interpolating intensity values across the surface

Intensity values for each polygon are matched with the values of the adjacent polygon along the common edge thus eliminating the intensity discontinuities occur in “flat shading”

Calculation for each polygon surfaces
Determine the average unit normal vector at each polygon vertex.
Apply an illumination model to each vertex to calculate the vertex intensity
Linearly interpolate the vertex intensities over the surface of the polygon



N1 normal to ABCD plane , N2 normal to CDEF plane and so on .

For any vertex position V normal unit vector

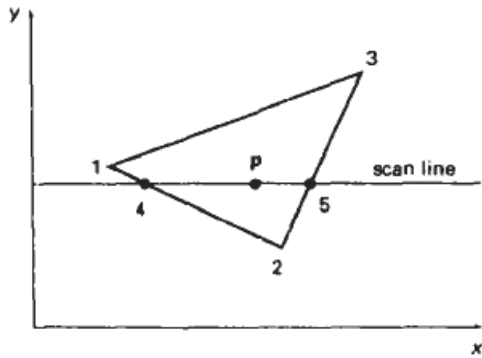
$$N_v = \frac{\sum_{k=1}^n N_k}{\left| \sum_{k=1}^n N_k \right|}$$

once Nv is known intensity at vertices can be obtained from lighting model

Next step: Interpolating intensities along polygon edges
fast method to find intensity at 4 using 1 and 2 using only vertical displacement

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} \cdot I_1 + \frac{y_1 - y_4}{y_1 - y_2} \cdot I_2$$

Similar process for I5, using 3 and 2



For interior point p interpolated from the bounding intensities at point 4 & 5

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

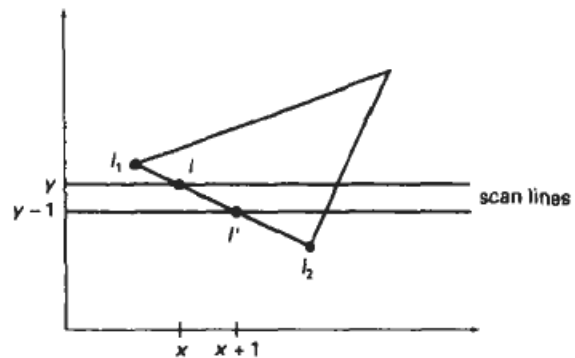
Easier than this is incremental calculations

for successive edge intensity values

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

for next scan line $y - 1$

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$



Similar calculation to obtain intensities at successive horizontal pixel positions along each scan line

For color, intensity of each color component is calculated

Gouraud shading can be combined with a hidden surface algorithm to fill in the visible polygon

Advantages:

removes discontinuities of intensity at the edge compared to constant shading model

Disadvantages:

highlights on the surface are sometimes displayed with anomalous shapes and linear intensity

interpolation can cause bright or dark intensity streaks called Mach Bands to appear on the surfaces.

Mach bands can be reduced by dividing the surface into a greater number of polygon faces or Phong

shading (requires more calculation)

C. Phong Shading

More accurate method for rendering a polygon surface is to interpolate normal vector and then apply the illumination model to each surface point called "Phong Shading" or "Normal Vector Interpolation Shading".

It displays more realistic highlights on a surface and greatly reduces Mach band effect.

Steps

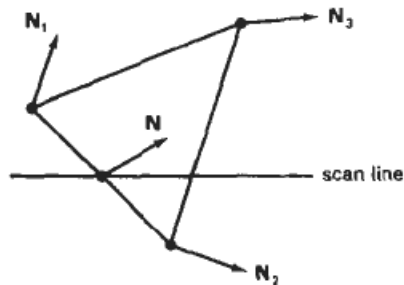
Determine the average unit vector normal at each polygon vertex

Linearly interpolate the vertex normals over polygon surface

Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points

N can be obtained by vertically
interpolating between edge end
point normals (N1 and N2)

$$N = \frac{y - y_2}{y_1 - y_2} \cdot N_1 + \frac{y_1 - y}{y_1 - y_2} \cdot N_2$$



Incremental methods are used to evaluate normals between scan lines and along each individual scan line (as in Gouraud) at each pixel position along a scan line the illumination model is applied to determine the surface intensity at that point

Produces accurate results than the direct interpolation but it requires considerable more calculations

D. Fast Phong Shading (FPS)

FPS approximates the intensity calculations using a Taylor series expansion and triangular surface patches

Surface normal at any point (x,y) over a triangle as $N = Ax + By + C$

A,B,C are determined from three vertex equations $N_k = Ax_k + By_k + C \dots k = 1,2,3$ (x_k,y_k vertex position)

Omitting reflexivity and attenuation parameters

$$I_{diff}(x,y) = \frac{L \cdot N}{|L| \cdot |N|} = \frac{L \cdot (Ax + By + C)}{|L| \cdot |Ax + By + C|} = \frac{(L \cdot A)x + (L \cdot B)y + L \cdot C}{|L| |Ax + By + C|}$$

We can write

$$I_{diff}(x,y) = \frac{ax + by + c}{(dx^2 + exy + fy^2 + gx + hy + i)^{1/2}} \dots (i)$$

Where a,b,c,d are used to represent the various dot products eg $a = L \cdot A$

Finally denominator in eq(i) can be expressed as Taylor series expansion and retain terms up to second degree in x and y. This yields

$$I_{diff}(x,y) = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0 \dots (ii)$$

Where each T_k is a function of parameters a ,b ,c and so forth

Using forward difference we can evaluate (ii) with only two additions for each pixel position (x,y) once the initial forward different parameter have been evaluated

FPS is two times slower than Gouraud shading, Normal Phong shading is 7 times slower than Gouraud

FPS can be extended to include specular reflections, FPS algorithms can be generalized to include polygons other than triangles and infinite viewing positions.

Introduction to Virtual Reality and Animation

Virtual Reality

What VR is not:

- VR is not just any form of *Computer Graphics*

What VR is:

- “A system for providing an interactive exploration of a three dimensional virtual environment”
- “The use of 3D graphics displays to explore a computer generated world”
- An attempt to model the real world as believably as possible
- An advanced form of human computer interface

Virtual Reality Systems

A typical VR system consists of six main components grouped into two:

a. Internal Components:

- | | |
|------------------------|---------------------|
| i. Virtual world | ii. Graphics Engine |
| iii. Simulation Engine | iv. User interface |

b. External Components:

- | | |
|----------------|------------------|
| i. User inputs | ii. User outputs |
|----------------|------------------|

Virtual World

A scene database containing the geometric representations and attributes for all objects within the environment

Graphics Engine

- Responsible for actually generating the image or scene, which a viewer will see
- Usually the scene database and the viewer’s current position and orientation is taken into account
- It also includes other information from the scene data base e.g. sounds, special effects textures etc

Simulation Engine

- **Does most of the work required to maintain virtual environment**
- **Concerned purely with the dynamics of the environment**
 - **how it changes over time**
 - **how it responds to the user's actions**
- **This includes handling interactions, physical simulations (gravity, inertia)**

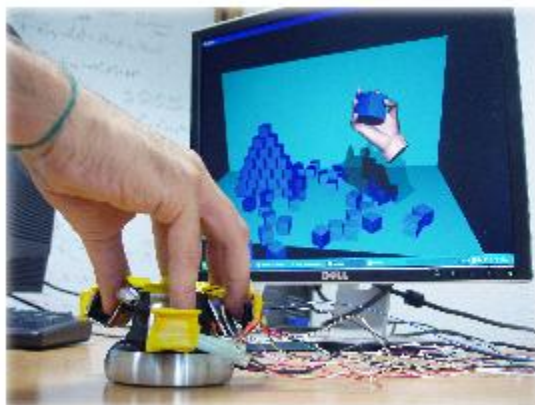
User Interface

- **Controls how the user navigates and interacts with this virtual environment**

Interaction Styles (referring to the way the simulated/virtual environment is represented)

i. Desktop VR

- **Based on the concept that the potential user interacts with the computer screen without being fully immersed and surrounded by the computer generated environment**
- **Applications domains involve architecture, industrial design, data visualization**
- **Less cost and involves less use of interacting technology**



ii. Projected VR

- **Based on overlapping of the image of the real user on the computer generated world**
- **A special movement tracking device can capture the movements of the user and enter them so that they can cause actions and reactions in the virtual world**

- **Often used in VR Art shows**



iii. Immersive VR

- **The user appears to be fully inserted in the computer generated environment**
- **Illusion rendered by providing HMD (Head Mounted Device) with 3D viewing and a system of head tracking that gives the exact correspondence and coordination of the user's movements with the feed-back of the environment**
- **The goal is to completely immerse the user within a synthetic environment or make them feel a part of that environment**



iv. CAVE or Fish Tank VR

- Cave is a small room where computer generated world is projected on the front and side walls using projectors
- Suitable for collective VR experience (*allows different people to share the same experience at the same time*)
- e.g. cockpit simulations



v. *Tele-presence*

- Here, users can influence and operate in a world that is real but in a different location
- Telepresence is used for remote surgical operations and for exploration and manipulation of hazardous environments (space, underwater)
- Remote robots used in bomb disposal operations

vi. *Augmentation*

- mixed reality provides a half way point between an non immersive and fully immersive VR system
- a user's view of the world is supplemented with virtual objects and items whose meaning is aimed at enriching the information content of the real environment
- e.g. Head Up Displays (HUD)
 - Used in modern military aircraft
 - These superimpose flight data such as altitude, air speed upon the pilots field of view
 - This can be on a cockpit mounted display or upon the pilot's helmet visor

Virtual Reality Software

- Software packages exist that allows users to either experience virtual worlds or even create and edit them
- 3D graphics engines and immersive environments has occurred in the gaming industry
- Most VR Packages are costly and require high specification workstations to run properly

Factors in VR Systems

Factors that can attribute to a realistic and believable virtual environment:

Visual realism:

- The level of realism in a scene helps considerably in making a believable environment

- With best applications, the viewer does not notice any transition between real footage and computer generated effects
- But it requires a lot of rendering time

Image Resolution

- Image resolution is closely linked with visual realism
- Computer generated images contain pixels , the size and number of these are dependent on the display size and resolution
- The color and intensity at each pixel must be generated individually, putting a heavier load on the graphics system

Frame Rate

- To give the impression of a dynamic picture, the system updates the display very frequently with a new image
- Images stop flickering at frequencies above CFF which can be as low as 20 Hz
- Normal TV broadcasts update at a frequency of 50 Hz in the UK, 60 in the US
- Achieving this refresh rate puts a heavy load on the graphics system

APPLICATIONS OF VR

Flight Simulation

- For Pilot training
- Safe and realistic
- Risk free

Engineering and design

- CAD and CAM
- View products as it would be seen when manufactured

Human factor modeling

- Used to model human behavior in the design of new products or buildings
e.g. simulation of fire in a building and a user can view how the virtual occupants react to the emergency
- Helps in designing escape strategies, fire modeling, human behavior

Visualization

- **Data visualization**

Trends in Computer Graphics

1. **Computer Animation:** Computer Animation generally refers to any time sequences of visual changes in a scene. In addition to changing object position with translations or rotations, a computer –generated animations could display time variations in object size, color, transparency, of surface texture. Advertising animation often transition one object shape into another (e.g. motor oil into Automobile Engine)

It can also be generated by changing camera parameters, lighting effects and illumination model and rendering methods.

Application:

1. **Entertainment (Motion picture and cartoons)**
2. **Advertising**
3. **Scientific and Engineering studies**
4. **Training and Education**

It provides realistic display and visual effects (displayed with exaggerated shaper and unrealistic motions and transformations).

Design of Animation Sequence:

In general, an animation sequence is designed with following steps:

1. **Storyboard layout**
2. **Object definitions**
3. **Key frame specification**
4. **Generation of in between frames**

For” frame by animations”, each frame of scene is separately generated and stored later, the frames can be recorded on film or they can be consecutively displayed in “real time playback” mode.

The story board is an outline of the action. It defines motion sequence as a set of basic events that are to take place (rough sketches) basic idea for the motion).

An object definition is given for each participant in action. Object can be defined in terms of basic shapes, such as polygons or splines and associative movements.

A key frame is a detail drawing of the scene at a certain time in the animation sequence. Within each key frame each object is positioned according to the time for that frame.

In between are the intermediate frames between key frames. The number of in between needed is determined by the media to be used to display the animation. Film requires 24 frames per second. There might be 3-5 in between.

The other tasks include motion verification, editing and producing and synchronization of a sound track.

Computer Animation Languages

Design and control of animations sequences are handled with a set of animation routines. A generally purpose language such as C, LISP, Pascal, or FORTAN is often used to program the animation functions have been developed. Animation functions include a graphics editor, a key frame generator, an in-between generator and standard graphics routines. The graphics editor allows us to design and modify object shapes, using spines surfaces, constructive solid geometry methods or other representation schemes.

A typical task in an animation specification is scene descriptive (positioning of object and light source defining the photometric parameters and camera parameters).

Another standard function is action specification (motion paths for the object and camera)

Key frame systems are specialized animation language designed simply to generate the in between from the user specified key frames. Usually, each object in the scene is defined as a set of rigid bodies connected at the joints and with limited number of freedom.

Parameterized systems allow object motion character to be specified as a part of object definitions. The adjustable parameters control such object characteristics as a degree of freedom, motion limitations and allowable shape changes.

Scripting system allow specification and animation sequences to be defined with a user input script. From the script, a library of various objects and motions can be constructed.

Morphing:

Transformation of object shapes from one form to another is called morphing, which is shortened form of metamorphosis. Morphing method can be applied to any motion or transition involving change in shape.

Given the key frames for an object transformation, we first adjust the object specification in one of the frames so that the number of polygon edges (vertices) is the same for two frames.

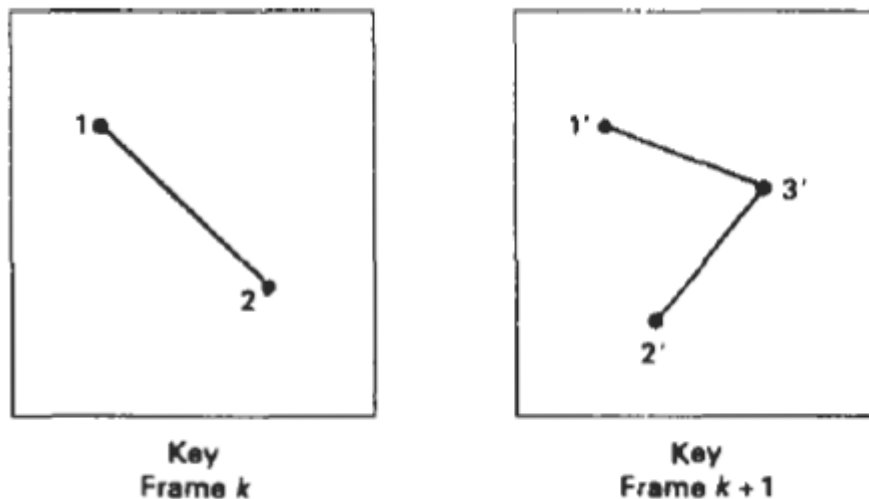


Fig: An edge with vertex position 1 and 2 in key frame k evolves into two connected edges in key frame k+1.

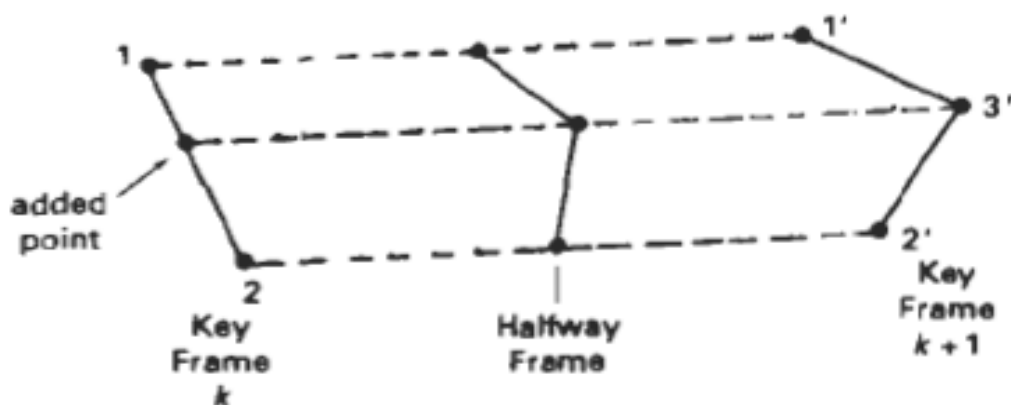


Fig: Linear interpolation for transforming a line segment in key frame k into two connected line segment in key frame k+1.

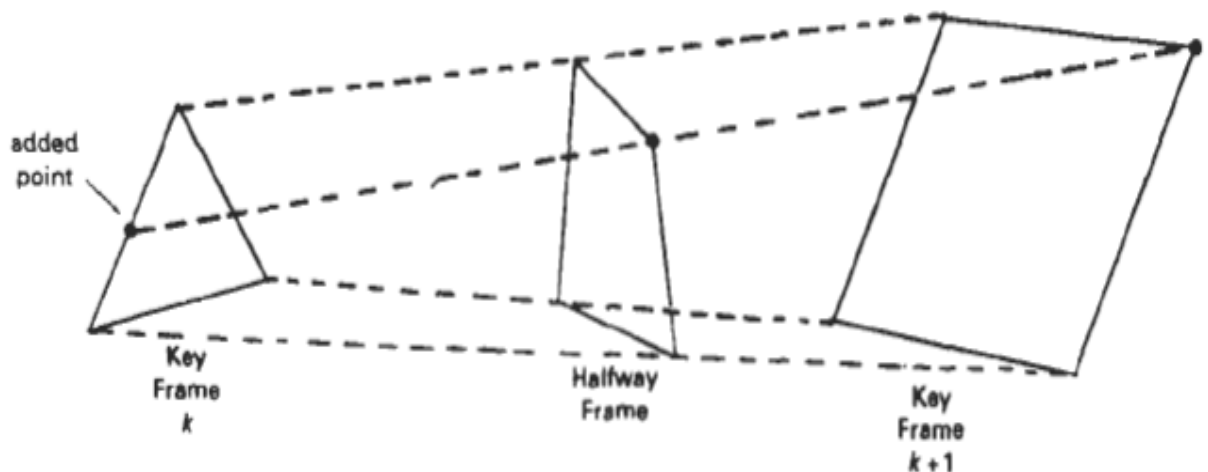


Fig: linear interpolation frame for transforming a triangle into quadrilateral

We can state general preprocessing rule for equalizing key frames in terms of either the number of edges or the number of vertices to be added to a key frame.

Simulating Acceleration:

Curve fitting techniques are often used to specify the animation paths between key frames. Given the vertex positions at the key frames, we can hit the positions with linear or non-linear paths. To simulate accelerations, we can adjust time spacing for the in-betweens.

For constant speed (zero acceleration), we use equal interval time spacing for the in-between for key frames at time t_1 and t_2 . The time interval between key frames is then divided into $(n+1)$ subintervals, yielding an in-between spacing of

$$\Delta t = (t_2 - t_1) / (n + 1)$$

We can calculate the time for any in-between as

$$t_{Bj} = t_1 + j\Delta t, j=1,2,\dots,n$$

And determine the values for the coordinate positions color, and other physical parameters. Non zero accelerations are used to produce realistic displays of speed changes, particularly at the beginning and end of the motion sequence. We can model the starting and slow down portion of an animation path with spline or trigonometric functions. Parabolic and cubic time functions have been applied to acceleration modeling, but trigonometric functions are more commonly used in animation packages.

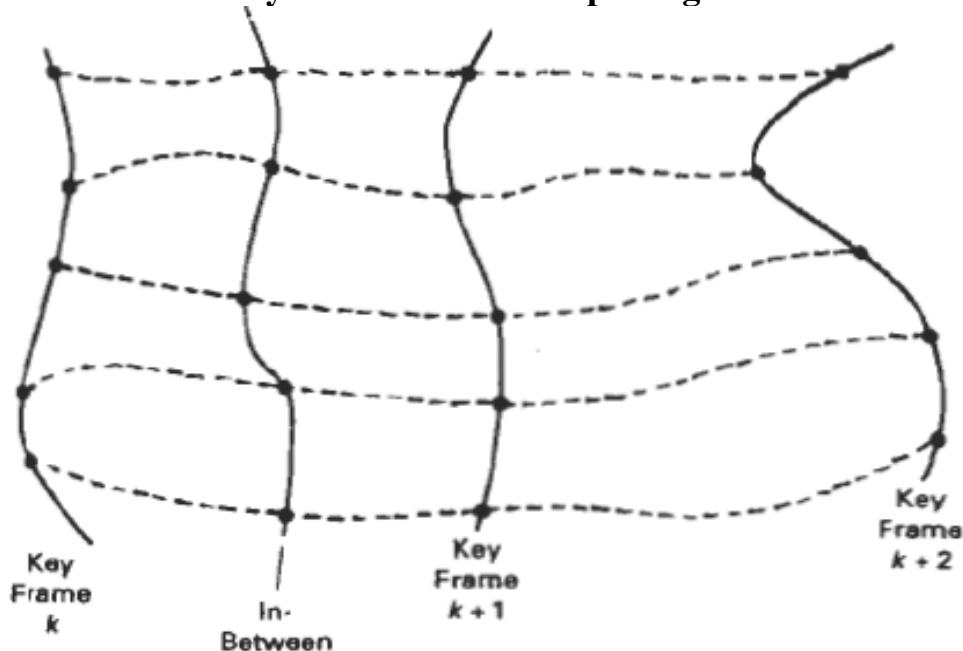


Fig: fitting key frame vertex positions with non-linear splines.

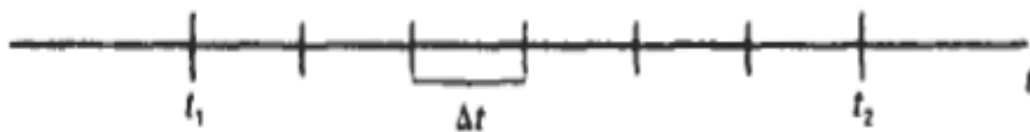


Fig: in-between for motion at constant speed

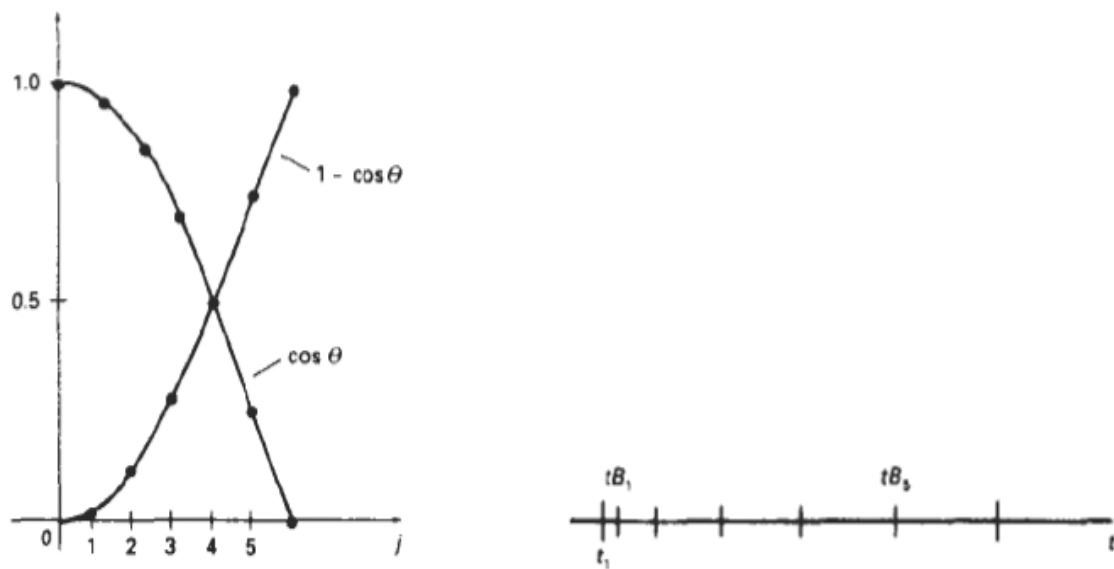
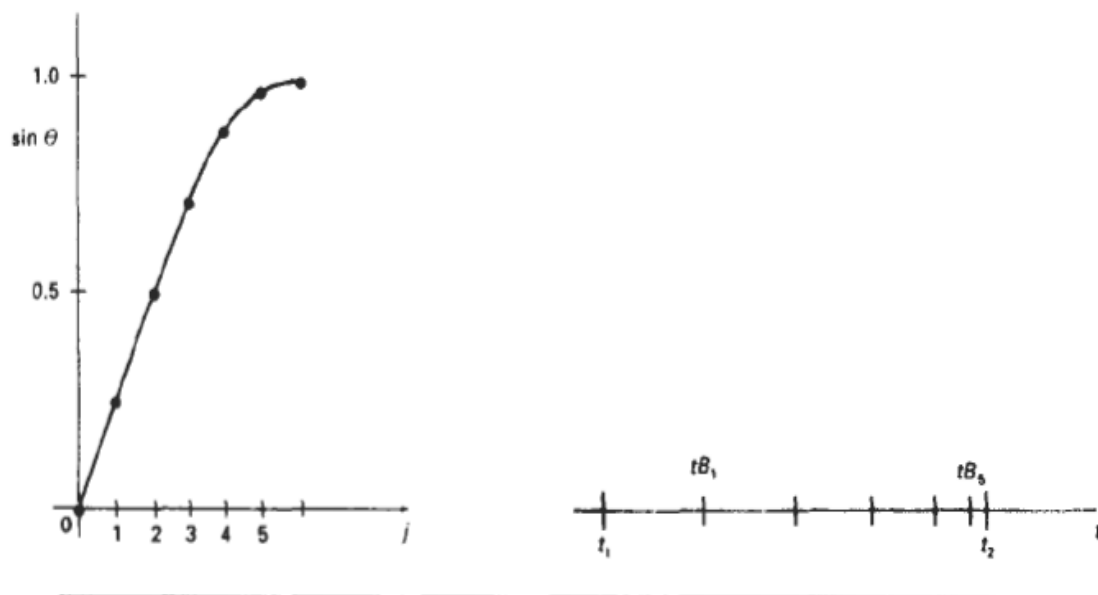


Fig: in-between for motion of variable speed.



//// scaling program////

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
void matrixmulti(int translate[3][3],int vertex[3][5],int result[3][5]);
void main()
{
    int gd=DETECT,gm,errorcode;
    initgraph(&gd,&gm,"\\tc\\bgi");
    errorcode=graphresult();
    if(errorcode!=grOk)
    {
        printf("Graphics ERROR!: %s",grapherrormsg(errorcode));
        printf("\nPress any Key .....");
        getch();
        exit(1);
    }
    setbkcolor(2);
    int vertex[3][5]={ { 100,100,200,200,100},
    { 100,200,200,100,100},
    { 1,1,1,1,1 } };
    for(int i=0;i<4;i++)
    {
        setcolor(BLUE);
        line(vertex[0][i],vertex[1][i],vertex[0][i+1],vertex[1][i+1]);
    }

    printf("SCALE X=");
    int sx,sy;
    scanf("%d",&sx);
    printf("SCALE Y:=");
    scanf("%d",&sy);
    int translate[3][3]={ { 2,0,0},{ 0,1,0},{ 0,0,1 } };
    translate[0][0]=sx;
    translate[1][1]=sy;
    printf("Press any key for the translated line.....\n");
    getch();
    int result[3][5];
    matrixmulti(translate,vertex,result);
    for(i=0;i<4;i++)
    {
        setcolor(YELLOW);
        line(result[0][i],result[1][i],result[0][i+1],result[1][i+1]);
    }
    getch();
    closegraph();
}

```

```

}
void matrixmulti(int translate[3][3],int vertex[3][5],int result[3][5])
{
    for(int i=0;i<=3;i++)
    {
        for(int j=0;j<=5;j++)
        {
            result[i][j]=0;
            for(int k=0;k<=3;k++)
                result[i][j]+=translate[i][k]*vertex[k][j];
        }
    }
}

```

Translation program

```

#include<stdio.h>
#include<conio.h>85

#include<stdlib.h>
#include<graphics.h>
void matrixmulti(int translate[3][3],int vertex[3][5],int result[3][5]);
void main()
{
    int gd=DETECT,gm,errorcode;
    initgraph(&gd,&gm,"\\tc\\bgi");
    errorcode=graphresult();
    if(errorcode!=grOk)
    {
        printf("Graphics ERROR!: %s",grapherrormsg(errorcode));
        printf("\nPress any Key .....");
        getch();
        exit(1);
    }
    setbkcolor(2);
    int
vertex[3][5]={ { 100,100,200,200,100},{ 100,200,200,100,100},{ 1,1,1,1,1 } };
    for(int i=0;i<4;i++)
    {
        setcolor(BLUE);
        line(vertex[0][i],vertex[1][i],vertex[0][i+1],vertex[1][i+1]);

    }

    int translate[3][3]={ { 1,0,100},{ 0,1,200},{ 0,0,1 } };
    printf("Press any key for the translated line.....\n");
    getch();
}

```

```
int result[3][5];
matrixmulti(translate,vertex,result);
for(i=0;i<4;i++)
{
    setcolor(YELLOW);
    line(result[0][i],result[1][i],result[0][i+1],result[1][i+1]);

}
getch();
closegraph();
}
void matrixmulti(int translate[3][3],int vertex[3][5],int result[3][5])
{
    for(int i=0;i<=3;i++)
    {
        for(int j=0;j<=5;j++)
        {
            result[i][j]=0;
            for(int k=0;k<=3;k++)
                result[i][j]+=translate[i][k]*vertex[k][j];
        }
    }
}
```

//lab 1 Graphics practice

```
#include<graphics.h>
#include<stdio.h>
#include<process.h>
#include<dos.h>
#include<conio.h>
void main()
{
    int gd,gm,errorcode;
    int x,y;
    gd=DETECT;
    initgraph(&gd,&gm,"\\BC45\\BGI");
    errorcode=graphresult();
    if(errorcode!=grOk)
    {
        printf("Graphic Error:%s",grapherrormsg(errorcode));
        printf("Press any key....");
        getch();
        exit(1);
    }
    x=getmaxx();
    y=getmaxy();
    for(int i=1;i<=15;i++)
    {
        for(int j=1;j<=11;j++)
        {
            if(kbhit())
                exit(1);
            setcolor(i);
            setfillstyle(j,i+1);
            circle(x/2,y/2,75);
            floodfill(x/2,y/2,i);
            rectangle(10,10,200,200);
            floodfill(15,15,i);
            //delay(1000);
            fillellipse(400,400,100,75);
            floodfill(400,400,i);
            delay(1000);
            cleardevice();
        }
    }
}
```


// lab 2 Graphics practice

```
#include<graphics.h>
#include<stdio.h>
#include<process.h>
#include<dos.h>
#include<conio.h>
void main()
{
    int gd,gm,errorcode;
    int x,y;
    gd=DETECT;
    initgraph(&gd,&gm,"\\tc\\bgi");
    errorcode=graphresult();
    if(errorcode!=grOk)
    {
        printf("Graphic Error:%s",grapherrormsg(errorcode));
        printf("Press any key....");
        getch();
        exit(1);
    }
    x=getmaxx();
    y=getmaxy();
    for(int i=1;i<=15;i++)
    {
        for(j=1;j<=11;j++)
        {
            setcolor(i);
            setfillstyle(j,i+1);
            circle(x/2,y/2,150);
            floodfill(x/2,y/2);
            rectangle(10,10,200,200);
            floodfill(15,9);
            delay(1000);
            cleardevice();
        }
    }
}
```

//lab 3 Graphics practice

```
#include<graphics.h>
#include<stdio.h>
#include<process.h>
#include<dos.h>
#include<conio.h>
void main()
{
    int gd,gm,errorcode;
    int x,y;
    gd=DETECT;
    initgraph(&gd,&gm,"\\bc45\\bgi");
    errorcode=graphresult();
    if(errorcode!=grOk)
    {
        printf("Graphic Error:%s",grapherrormsg(errorcode));
        printf("Press any key....");
        getch();
        exit(1);
    }
    x=getmaxx();
    y=getmaxy();
    while(!kbhit())
    {

        for(int i=x/2;i<x;i+=2)
        for(int j=y/2;j<y;j+=2)
        {
            if(kbhit())
            exit(1);
            setcolor(BLUE);
            setfillstyle(5,RED);
            circle(i,j,50);
            floodfill(i,j,BLUE);
            delay(50);
            cleardevice();
        }
    }
```