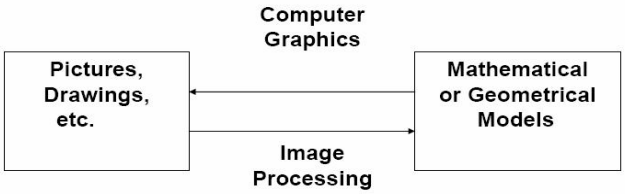
CHAPTER – 1

Computer graphics

1.1 Introduction

Computer graphics is a field or art related to the generation of graphics using digital computers. It includes the *creation; storage and manipulation of images* of objects come from diverse fields such as physical, mathematical engineering, architectural abstract structures and natural phenomenon. We use different input and display devices to generate & perform the modification on the digital images. Some such devices are keyboard, mouse, or touch sensitive panel, monitors etc. Some considerable terminologies on the computer graphics are:

- *Modeling*: creating and representing the geometry of objects in the 3D world
- *Rendering*: generating 2D images of the 3D objects
- *Animation*: describing how objects change in time



Difference between image processing and CG are listed below:

Computer Graphics (CG)	Image Processing (IP)
1. It is the field related to the generation of images using computers	1. It applies technique to modify or interpret existing pictures
2. It synthesizes picture from mathematical or geometrical models	2. It analyze picture to derive description in mathematical or geometrical forms
3. It includes the creation storage and manipulation of images or objects	3. It is the part of computer graphics that handles image manipulation or interpretation
4. E.g., drawing a picture.	4. E.g., making blurred image visible.

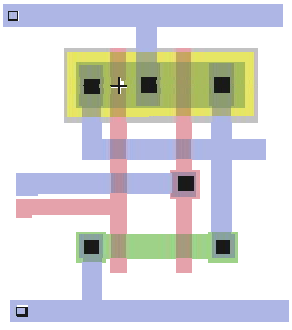
1.2 History of Computer Graphics

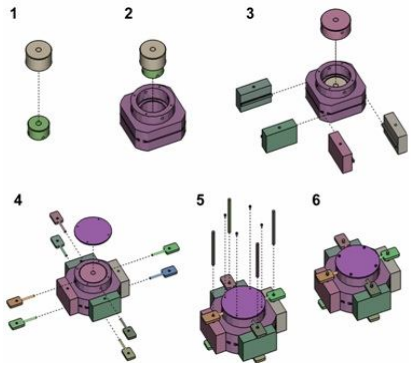
The history of the computer graphics illustrates a chronically development of hardware and software. The past principals and techniques are still applicable in present and future computer graphics technology. The evolutions of graphics can be explained under following points.

1950	First graphic images were created on oscilloscope using analog electronic beams.
1951	Mainframe computer with a CRT monitor that can plot blips based on radar data.
1955	CRT monitor with vector graphics and invention of light pen as an input device.
1959	First industrial CAD (Computer-Aided Design) system for cars design.
1961	*Spacewar*, the first video game, is developed.
1963	First hidden-line and hidden-surface removal algorithms. The mouse was invented.
1965	The digital line drawing algorithm by Jack Bresenham
1969	Raster video displays using frame-buffer (with 3 bits per pixel)
1973	First use of 2D animation on a film *Westworld*
1975	Specular illumination model known as Phong shading
1980	Ray tracing with reflection, refraction, antialiasing, and shadows.
1982	Autodesk is founded and AutoCAD released.
1985	Introduces noise functions for creating natural patterns such as marble, wood, etc.
1992	Open Graphics Library (OpenGL) specification defining a standard cross-language cross-platform API for computer graphics.
2001	*The Spirits Within,* digital film that includes photorealistic digital actors
Nowadays	Many graphics-hardware vendors; many console manufacturers, many software packages, many animation studios, and many game companies.

1.3 Main applications of CG

1. **Graphical User Interface:** The GUI application on the computer system provide the facility of selection of menu items, icons & object on screen, directly either by touching it (in case of touch panel) or just by clicking on it. This is more efficient mechanism rather than the assembly language coding.
2. **Plotting:** To present meaningfully and concisely the ***trends and patterns*** of complex data, we use histograms bar, pie charts, task-scheduling charts etc.
3. **Computer Aided Drafting & Design:** One of the major uses of computer graphics is to design components and systems of mechanical, electrical, electromechanical and electronic devices, including structures such as buildings, automobiles bodies, and airplane and ship hulls, very large-scale integrated (VLSI) chips, optical systems and telephone and computer networks. Computer Aided Design (CAD) and Computer Aided Manufacture(CAM) are important terms in the field of design and manufacture
   - **Computer Aided Design (CAD)** involves the use of computer hardware and graphics software to generate design drawings. A wide range of computer based tools are used to assist engineers, architects and other design professionals in their design activities. It is also known as CADD, which stands for computer aided design and drafting. Modern CAD equipment enables the designer to quickly produce very accurate and realistic images of products to be manufactured. CAD is usually employed when simple drafting is not able to do the job such as in design of automobiles, airplanes, ships and other industrial designs.
   - **Computer Aided Manufacturing (CAM)** is a system of automatically producing finished products by using computer controlled production machines. CAD and CAM work together in that the digital model generated in CAD is inputted to the CAM software package. The CAM software needs to know the physical shape of the product (CAD model) before it can compose a proper set of fabrication instructions to a production machine. The major difference between CAD and CAM lies in the end user. While CAD software is mostly used by an engineer, CAM is used by a trained machinist.
4. **Simulation:** Simulation is the ***imitation*** of the conditions encountered in real life. The simulation provides the virtual reality of any system & helps us to train people without accidental loss. Such type of training may be pilot training, military weapon training, space training, electronic circuit design training, VLSI training etc.
5. **Entertainment:** Computer graphics are used for cartooning, game playing, animation etc.
6. **Art & Commerce:** The pictures of new car models, new products in market, awareness arts, traffic symbols etc. produced using computer graphics express a message and attract attention of people at public places, transportation terminals, supermarkets, hotels etc.
7. **Cartography:** Cartography is a subject which deals with the making of maps and charts of the geographical data in accurate and schematic way.
8. **Medical Imaging:** The range of application spans from tools for teaching and diagnosis, all the way to treatment. Computer graphics is tool in medical applications
9. **Education and Training:** Different computer graphics and images are used in schools and many training center for the better understanding the subject of interest.
10. **Other applications:** Multimedia systems, Animation for entertainment, Robotics & virtual reality, Image processing, etc.



CHAPTER – 2

Hardware Concepts

2.1 Input Devices

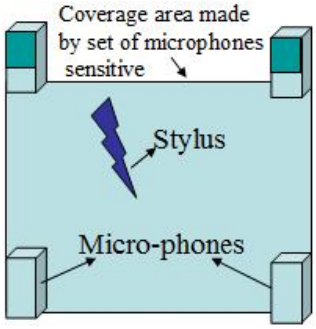
Input device are used to feed data or information into a computer system. They are usually used to provide input to the computer upon whose reaction, outputs are generated. The various type of input devices are keyboard, mouse, light pens, touch panels etc.

1. Tablet

A tablet is a digitizer which is used to scan over an object and to put 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 (like mouse with cross hairs for sighting positions) held in the user's hand. A tablet is flat surface and its size 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 three types of tablets

- **Electric tablet:** A grid of wires on ¼ to ½ inch centers is embedded in the tablet surface, and electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induced 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 roughly how far the stylus is from the tablet. They cannot digitize the bulky objects because the movement of stylus over the object cannot sense by the electric tablet surface.



- **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 spark occurs and when its sound arrives at each microphone. They can digitize the bulky objects because the position point out by the stylus by generating the sound wave can easily encounter by the system with the help of microphones.

- **Resistive tablet:** The tablet is transparent and flexible with a thin layer of conducting material. When a battery-powered stylus is activated at certain position, it emits high-frequency radio signals which induce the radio signals on the conducting layer. The strength of the signal received at the edges of the tablet is used to calculate the position of the stylus. They can also able to digitize some object of screen.

2. Touch Panel

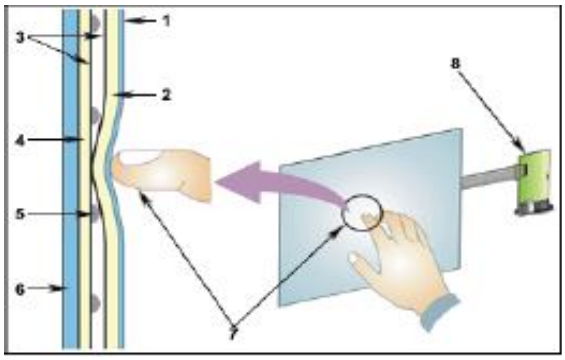
A touch panel is an input device that accepts user input by means of a touch sensitive screen directly with a finger to move the cursor around the screen or to select the icons. Because of their compact nature and ease-of-use, touch panels are typically deployed for user interfaces in automation systems, such as high-end residential and industrial control.

1. Polyester Film
2. Upper Resistive Circuit Layer

3. Conductive ITO (Transparent Metal Coating)
4. Lower Resistive Circuit Layer
5. Insulating Dots
6. Glass/Acrylic Substrate
7. Touching the overlay surface causes (2) Upper Resistive Circuit Layer to contact the (4) Lower Resistive Circuit Layer, producing a circuit switch from the activated area.
8. The touch screen controller gets the alternating voltages between the (7) two circuit layers and converts them into the digital X and Y coordinates of the activated area.

Input Devices: Touch Screens

plasma panels with touch screens



Followings are mostly used touch panels:

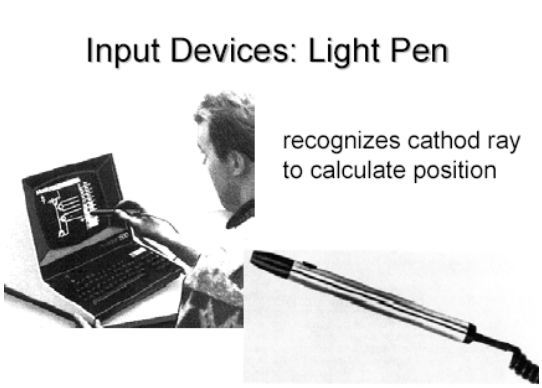
- **Optical touch panel:** It uses a series of infrared light emitting diodes (LED) along one vertical edge and along one horizontal edge of the panel. The opposite vertical and horizontal edges contain photo detectors to form a grid of invisible infrared light beams over the display area. Touching the screen breaks one or two vertical and horizontal light beams, thereby indicating the finger's position. This is low resolution panel, which offers 10 to 50 positions in each direction.

- **Sonic panel:** Burst of high-frequency sound waves traveling alternately horizontally and vertically generated at the edge of the panel. Touching the screen causes part of each wave to 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 source. This is a high-resolution touch pane having about 500 positions in each direction.

- **Electric touch panel:** It consists of slightly separated two transparent plates one coated with a thin layer conducting material and the other with resistive material. When the panel is touched with finger, the two plates are forced to touch at the point of contact thereby creating the voltage drop across the resistive plate which is then used to calculate the co-ordinate of the touched position. The resolution of this panel is similar to that of sonic touch panel.

3. Light pen

It is a pencil-shaped device to determine the coordinate of a point on the screen (i.e. digitizer). In raster display, Y is set at Ymax and X changes from 0 to Xmax the first scanning line. For second line, Y decreases by one and X again changes from 0 to Xmax, 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 an 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 positions seen by the light pen. Because of the following drawbacks the light pens are not popular.



- Light pen ***obscures*** or hides the screen images as it is pointed to the required spot.
- Prolong use of it can cause ***arm fatigue***.
- It cannot report the coordinates of a point that is completely black.
- It gives sometimes false reading due to background lighting in room.

4. Keyboard

A keyboard creates a code such as ASCII uniquely corresponding to a pressed key (i.e. work on ***Hall's effect***). It usually consists of alphanumeric key, function keys, cursor-control keys, and separate numeric pad.

5. Mouse

A mouse is a small hand held device used to position the cursor on the screen. Following are the mice, which are mostly used in computer graphics.

- **Mechanical mouse:** It moves the curser position on the screen according as the moment of the roller in the base of this mechanical mouse.
- **Optical mouse:** A LED in the bottom of the mouse directs a beam of light down onto the pad from which it is reflected and sensed by the detectors.

6. Barcode reader

A barcode reader (or barcode scanner) is an electronic device for reading printed barcodes. Like a flatbed scanner, it consists of a light source, a lens and a light sensor translating optical impulses into electrical ones. Additionally, nearly all barcode readers contain ***decoder circuitry*** analyzing the barcode's image data provided by the sensor and sending the barcode's content to the scanner's output port. The barcode reader can be categories as:

- **Pen-type readers**
  - Pen-type readers consist of a light source and a photodiode that are placed next to each other in the tip of a pen or wand.
  - To read a bar code, the tip of the pen moves across the bars in a steady motion.
  - The photodiode measures the intensity of the light reflected back from the light source and generates a waveform that is used to measure the widths of the bars and spaces in the bar code.
  - Dark bars in the bar code absorb light and white spaces reflect light so that the voltage waveform generated by the photo diode is a representation of the bar and space pattern in the bar code.
  - This waveform is decoded by the scanner in a manner similar to the way Morse code dots and dashes are decoded.

- **Laser scanners**
  - Laser scanners work the same way as pen type readers except that they use a laser beam as the light source and typically employ either a reciprocating mirror or a rotating prism to scan the laser beam back and forth across the bar code.
  - As with the pen type reader, a photodiode is used to measure the intensity of the light reflected back from the bar code.

7. Data Glove

A data glove is an interactive device, resembling a glove worn on the hand, which facilitates physical sensing and fine-motion control in ***robotics*** and ***virtual reality***. Data gloves are one of several types of electromechanical devices used in ***haptic applications***. Tactile sensing involves simulation of the sense of human touch and includes the ability to perceive ***pressure, linear force, torque, temperature,*** and ***surface texture.***

2.2 Output Devices

All the **output devices** can be categorized as hardcopy and softcopy devices.

- **Hard copy devices** are those that give the output in the tangible form. Printers and Plotters are two common hard copy devices.
- **Soft copy devices** give output in the intangible form or the virtual form, e.g. something displayed on a screen. All the computer monitors are covered under this category.

A. Hardcopy Devices

Printers

All the printers irrespective of the technology used can be categorized as:

- **Impact printers:** There is a direct contact between the printing head and the paper on which the print is produced. They work by striking a head or a needle against an inked ribbon which leaves a mark on the paper. These printers produce a lot of noise when printing, because of the head striking the paper. Examples are ***Dot Matrix, Daisy Wheel*** and ***Line printers***.
- **Non-impact printers:** the printing head never comes in direct contact with the paper. These printers work by spraying ink on the paper. Electrostatic or electromagnetic charge is used in these printers. Examples are ***Ink-Jet*** and ***Laser*** printers.

Plotter

The plotter is a computer printer for printing vector graphics. Plotters differ from printers in that they draw lines using a pen. As a result, they can produce continuous lines, whereas printers can only simulate lines by printing a closely spaced series of dots. Multicolor plotters use different-colored pens to draw different colors. Thus, plotters are considerably more expensive than printers. The various type of plotter is: Drum plotter, Flatbed plotter, Electrostatic plotter etc.

B. Display Devices

The devices which can give the user interface of the inputted data are the display devices. In the case of computer, the monitor is most common display device that can be categories as the Cathode Ray Tube (CRT) monitor and the flat panel monitor. To display the image, computers have some assigned memory called the frame buffer. The graphic cards also have the memory buffer and the graphics processors so that it can increase the display strength of the computer graphics.

- ***Pixel:*** Graphic displays are like very large ***dot matrices***. Each dot in a graphic display is called ***picture element***, *pixel* or ***pel***. The capabilities of a graphic display depend on number of pixels horizontally and vertically.
- ***Dpi*** =Dot per inch
- ***Ppi*** = Point per inch

Typical Term for Display Devices

- **Fluorescence/ Phosphorescence**

  When the electron beam strikes the phosphor coated screen of the CRT, the individual electrons are moving with kinetic energy proportional to the accelerating voltage. Some of this energy is dissipated as heat, but rest is transferred to the electrons of phosphor atoms making jump to higher quantum energy levels. In returning to their previous quantum levels, these excited electrons give up their extra energy in the form of light of different frequencies i.e. colored light, predicted by the quantum theory.

  - ***Phosphorescence*** is the light given off by the return of the relatively more stable excited electrons to their unexcited state, once the electron beam excitation is removed.
  - Most of the phosphors relax back to the ground state by emitting a photon of light which is called ***fluorescence***

- **Persistence**

  A phosphor's persistence is the time for the emitted light to decay to 10 % of the initial intensity. The persistence may be varied with different phosphors. The phosphors used for graphics display usually have persistence of 10 to 60 microseconds. A phosphor with low persistence is useful for animation; a high persistence phosphor is useful for high complex static picture.

- **Refresh Rate**

  The refresh rate is the number of times per second the image is redrawn to give a feeling of un-flickering pictures and is usually 50 per second. The refresh rate above which a picture stops flickering and fuses into a steady image is called the ***critical fusion frequency (CFF)***. The factors affecting the CFF are persistence, image intensity, room light, wave length of light & observer.

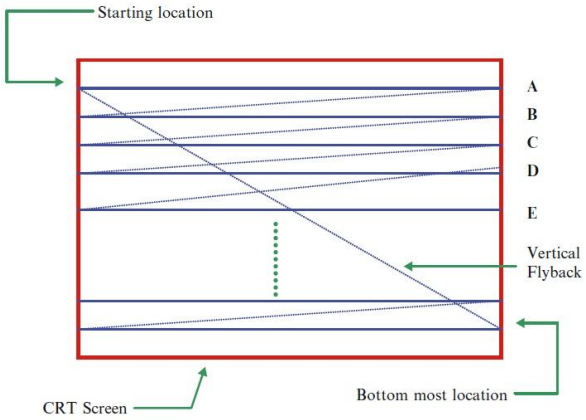
- **Resolution**

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

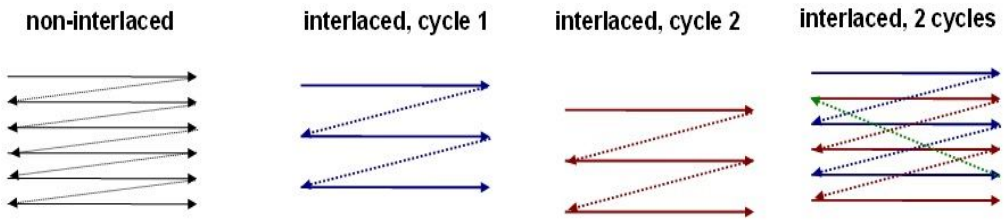
- **Aspect ratio**

  The ratio of vertical points to horizontal point necessary to produce equal length line in both directions on screen (as monitor is rectangular not the square) is called aspect ratio. An aspect ratio of ¾ means that a vertical line is plotted with 3 points has the same length as horizontal line plotted with 4 points. Generally, the aspect ratio is not one.

- **Retrace procedure**
  At the end of each scan line in raster scan display, the electron beam returns to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refreshing each scan line is called the ***horizontal retrace*** of the electron beam. And at the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame which is called ***vertical retrace***.



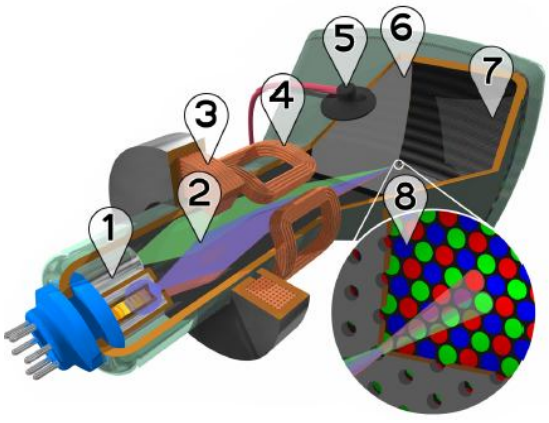
- **Interlaced refresh procedure**
  On some raster scan systems each frame is displayed in two passes using an *interlaced refresh procedure* so that the whole picture should displaced in half time. Here, the first scan does the ***even lines*** 0, 2, 4,... then the second scan does the ***odd lines*** 1, 3, 5,.... Interlacing is primarily used with slower refreshing rates to avoid the flickering. To show the animation, we have to move 24-frame per second (fps).



A. Color CRT

The **cathode ray tube (CRT)** technology was first used for computer displays, video monitors, televisions, radar displays and oscilloscopes. The CRT is an Evacuated gas tube that uses the filament to produce the electron beams which are focused in proper position of phosphorous coated screen by the help of magnetic focusing & deflection coils. Phosphors are ***organic compounds*** characterized by their persistence and their color (blue, red, green). A cathode ray tube (CRT) contains these basic parts:

1. Electron guns
2. Electron beams
3. Focusing coils
4. Deflection coils
5. Anode connection
6. Mask for separating beams for red, green, and blue part of displayed image
7. Phosphor layer with red, green, and blue zones
8. Close-up of the phosphor-coated inner side of the screen



* CRTs, or ***video monitors***, are the most common output device on computers today that uses the phosphors coated screen. Phosphors are characterized by color (RGB) and persistence.
* After the generation of electrons of weaker negative charge by heating the filament, it is focused to accelerate it by the repulsion of the inner cylinder walls in just the way that water is speeds up when its flow though a smaller diameter pipe.

* Then, the electron beam is deflected at proper position by the help of two set of plates of opposite charge, one positive the other negative. The first set displaces the beam up and down, and the second displaces the beam left and right. This helps us to focus the electron beam in proper position.
* The user can vary the voltage on the control grid to attenuate the electron flow. The electron beam causes the phosphor's atoms to move into higher energy state. The atoms give off energy as light when they return to their stable state i.e. glow & decay operation occurs & hence we need to refresh the screen continuously.
* A refresh rate of 50 - 60 Hz is usually sufficient to prevent flicker, but some systems refresh at even higher rates such as 72-76 Hz.

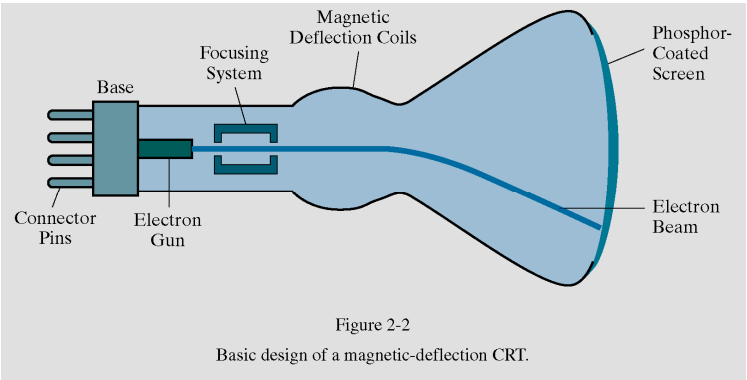
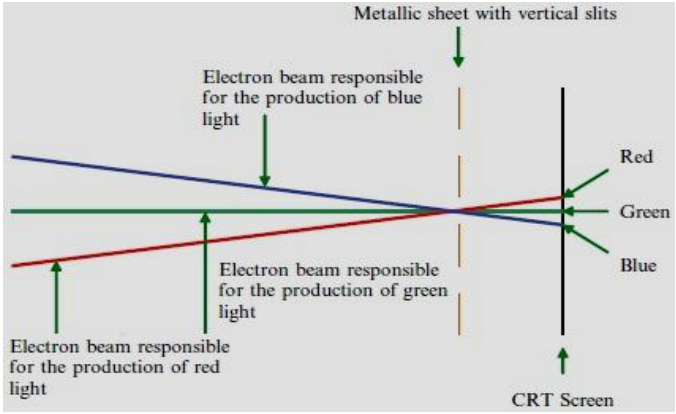


Figure 2-2
Basic design of a magnetic-deflection CRT.



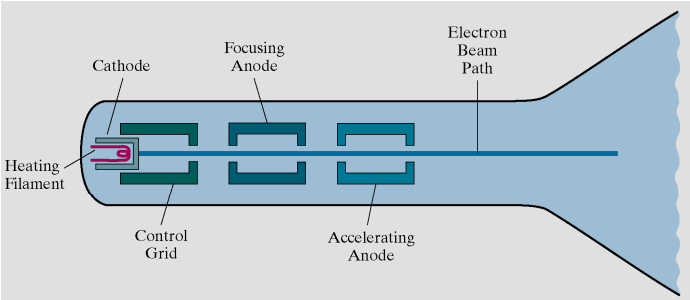


Figure 2-3
Operation of an electron gun with an accelerating anode.

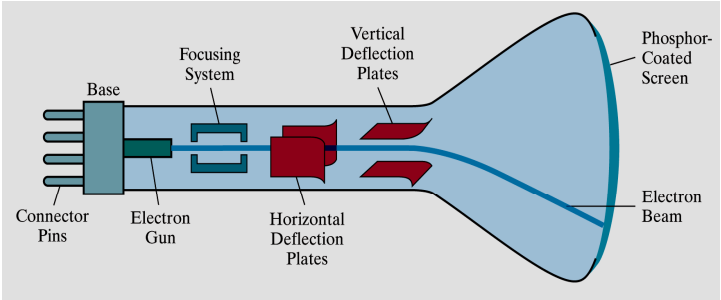


Figure 2-4
Electrostatic deflection of the electron beam in a CRT.

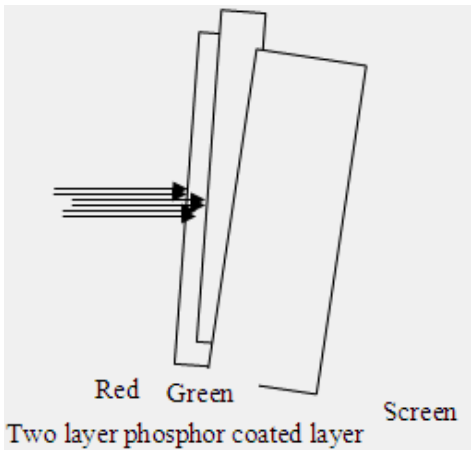
Methods for Color CRT

Two Methods are used in Color CRT for the raster display method

a. Beam Penetration Method

The beam penetration method is for the random scan monitor display where two different layers of phosphor coating are used, Red (outer) and Green (inner) coated on the CRT screen. The display of color depends on the depth of penetration of the electron beam into the phosphor layers. Screen color is controlled by the beam acceleration voltage. In this method, only four colors possible and hence the poor picture quality

- A beam of slow electrons excites only the outer ***red*** layer
- A beam of very fast electrons penetrates thru the red phosphor and excites the inner ***green*** layer.
- Intermediate is a combination of red and green so two additional colors orange and yellow color.
  - When quantity of red is more than green then color appears as ***orange***
  - When quantity of green is more than red then color appears as ***yellow***

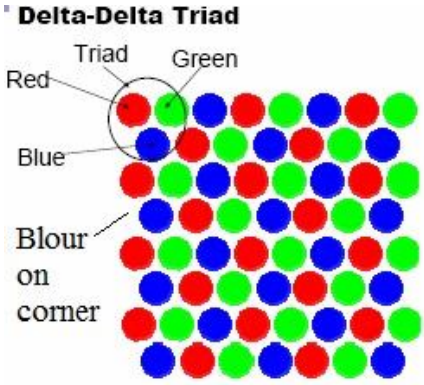


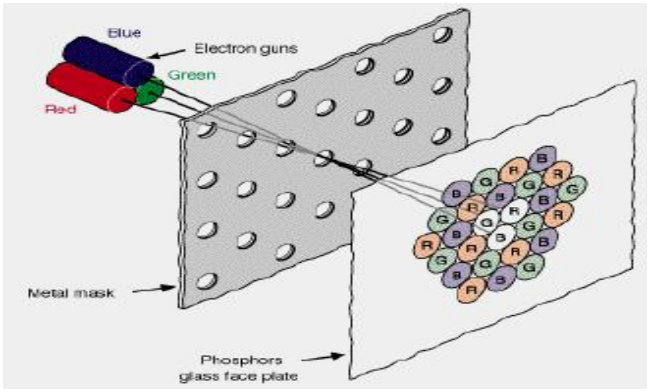
b. Shadow Mask Method

Shadow is a perforated *metal sheet* that ensures that the electron beam hits only the correctly colored phosphor dots and does not illuminate more than one dot. Essentially, the shadow mask "masks" the electron beam, thereby forming a smaller and more rounded point that can hit individual phosphor dots. The shadow mask absorbs electrons that are directed at the wrong color phosphor.

i. Shadow Mask (Delta-Delta CRT)

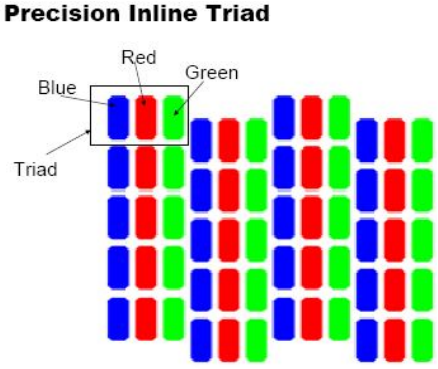
- Normally for Raster Scan System
- Inner side of viewing has several groups of Electron gun closely spaced red, green and blue phosphor dots called ***triad*** in delta fashion
- Thin metal plate is perforated with many holes near to inner surface called ***shadow mask***
- Shadow mask mounted in such a way that *each hole is aligned with respective triad*
- Triad so small that, it is perceived as a mixture of color.

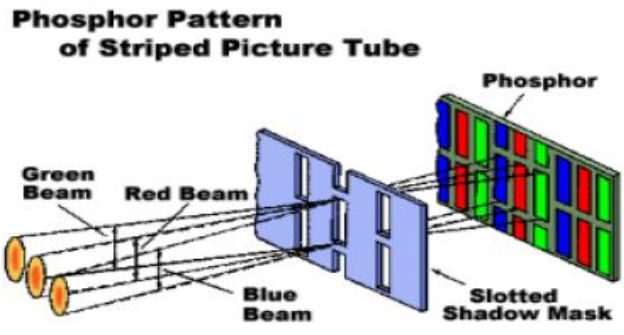




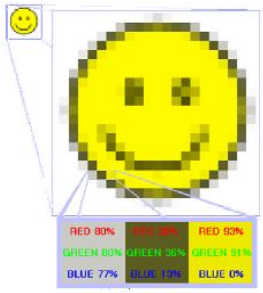
ii. Shadow Mask (Precision Inline CRT)

- Eliminates drawback of delta-delta CRT at the cost of slight reduction of images sharpness
- Normally 1000 scan lines
- Distance between centers of adjacent ***triads*** is called ***pitch.***
- In very high resolution pitch=0.21mm (0.61mm for home TV)
- Diameter of a electron beam is set at 1.75 * pitch (at which 50% of max)
- But small pitch is difficult to manufacture due to small triads too many holes in the shadow mask
- Also shadow mask decreases brightness only 20% electron beam hit the phosphorous
- No. of electron increased by increasing beam current (but focusing difficult, generate more heat & mask wrapping)



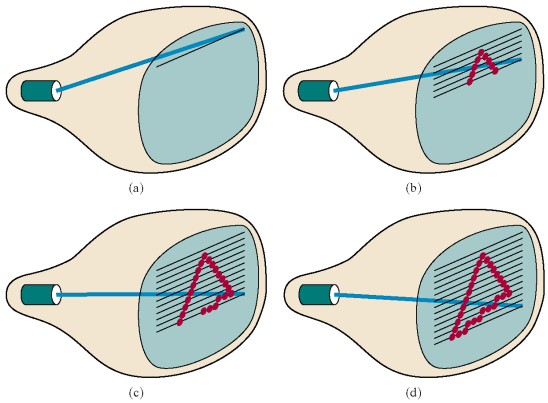


2.3 Raster Graphics



* A **raster graphics** image, **digital image**, or **bitmap**, is a data file or structure representing a generally rectangular grid of pixels, or points of color, on a computer monitor, paper, or other display device.
* The color of *each pixel is individually defined*; images in the RGB color space, for instance, often consist of colored pixels defined by three bytes—one byte each for red, green and blue.
* Here, the images are loaded on memory called the ***frame buffer*** before they display on the monitor.
* Less colorful images require less information per pixel; for example, an image with only black and white pixels requires only a single bit for each pixel.
* Raster graphics are distinguished from vector graphics in that vector graphics represent an image through the use of geometric objects such as curves and polygons.

2.3.1 Raster Display



* A raster display device is a large matrix of discrete cells or dots, each of which can be made bright to show the image on the screen.
* Our Computer Televisions uses raster display where the scanning of the pixel is done one row (scan line or raster line) at a time from the top-left of the screen to the bottom-right, even if there is change on the single pixel, in regular time interval as shown in figure.
* Here, we add a large continuous piece of special memory, called the ***frame buffer***, to store the intensity values (define shading & coloring) of each pixel and this is mapped on the screen using DAC.
* Frame buffer is a digital device but raster CRT is an analog device so DAC (Digital to Analog Converter) is required for reading from frame buffer and displaying on raster CRT.
* The ***bit plane*** is the minimum amount of memory for the pixel representation. In this memory, the bits are placed in continuous fashion as array to place the bits in matrix order. When there is bit-1, the electron gun strikes for one pixel & when bit-0, then nothing will happen. The different intensity level can be described by the number of bit planes.
* The process of digitizing the picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer is called ***scan conversion.***

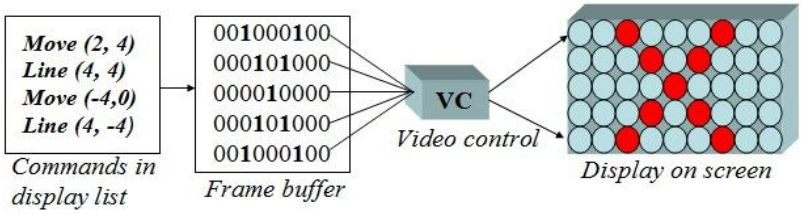


Fig. Raster display method on CRT monitor

* The ***display processor*** produces the raster image in the frame buffer from the commands, called scan conversation.
* The ***video controller*** moves the beam row wise across the pixels setting it on and off according to the content of the frame buffer
* The display must be refreshed to avoid flickering (raster image redisplayed 30 to 60 times per second)

2.3.2 Raster Display Technology

* When a particular command is called by the application program the graphics subroutine package sets the appropriate pixels in the *frame buffer*.
* The *video controller* then cycles thru the frame buffer, one scan line at a time, typically 50 times per second. It brings a value of each pixel contained in the buffer and uses it to control the intensity of the CRT electron beam.

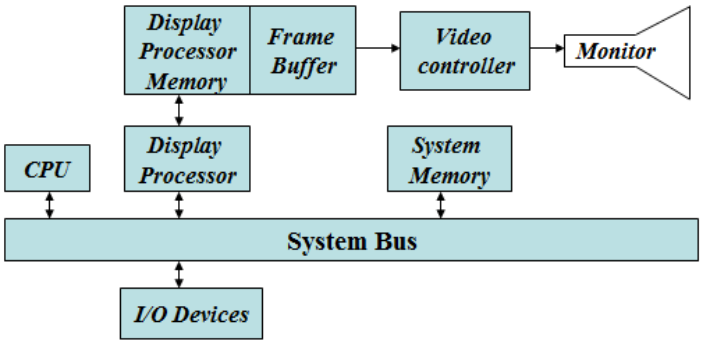


*Fig. **Architecture of Raster display system with display processor***

* So there exists a one to one relationship between the pixel in the frame buffer and that on the CRT screen
* 640 pixels by 480 lines is an example of *medium resolution* raster display & 1600 by 1200 is a *high resolution* one.

Advantages

- Ability to fill areas with solid color or patter
- Refreshing independent of the complexity of the image
- If interlaced refresh procedure is used, it draw picture quickly
- Can be used for intensity calculation and support large number of color combination.
- Can be used in any resolution, i.e., aspect ratio can be maintained easily.

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

Limitations

- Require special algorithm to move a pixels
- The "***stair case effect***" is unavoidable. Since the image are stores on the basis of pixel on the grid like structure so on making the small image more large, we seems the stair like image boundary of pixel. This is happen because we cannot draw the image on the half area of grid.
- To refresh a single pixel, it has to scan all frame buffers from top to bottom so is time consuming compared to the vector scan technology.
- Require more memory space (to increase frame buffer size)
- Works only with high speed display processor.

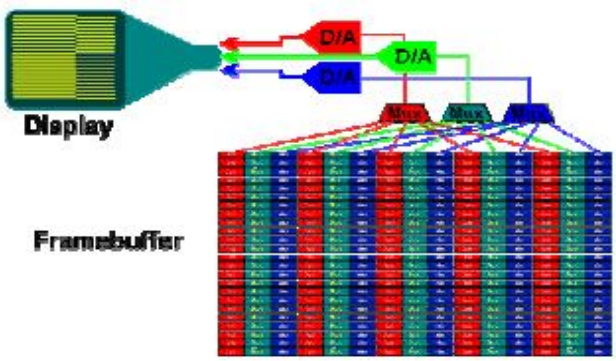
2.3.3 Frame Buffer

* Frame buffer (bit map) is a large continuous piece of memory that stores the color values of each pixel & hence the video controller connected to it, map these pixel intensity value one by one on the screen.
* Color values are commonly stored in 1-bit monochrome, 4-bit palletized, 8-bit palletized, 16-bit high color and 24-bit true color formats.
* An additional alpha channel is sometimes used to retain information about pixel transparency.

* The total amount of the memory required to drive the frame buffer is dependent on the resolution of the output signal, as well as the color depth and palette size.

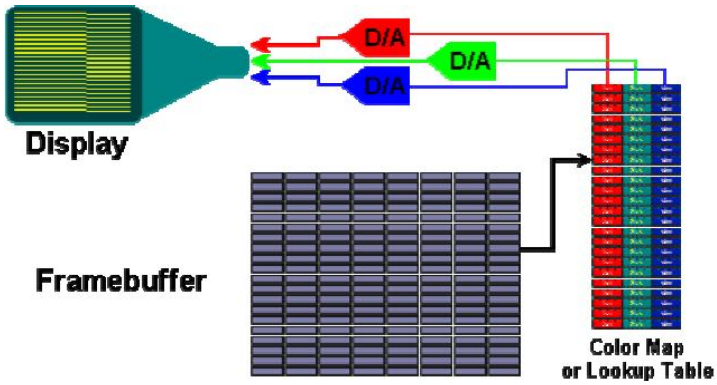
Frame Buffer Architecture of Raster Displays

- Each pixel requires at least 3 bytes. One byte for each primary color.
- Sometimes combined with a look-up table per primary
- Each pixel can be one of 2^{24} colors = 16777216 colors.



Frame Buffer Architecture of Indexed-Color

- Each pixel uses one byte
- Each byte is an index into a color map
- If the color map is not updated synchronously then *Color-map flashing* may occur.
- Color-map Animations
- Each pixel may be one of 2^{24} colors, but only 256 color be displayed at a time



Simple color frame buffer

- Three bit planes required one for each primary color. Each bit plane drives individual color gun for each 3 colors. These 3 colors are combined at CRT to yield 8 different colors.
- Each RGB electron beam being able to be set at one of 256 intensity levels.
- Total color depth of 24 bits per pixel, referred to as a true color system.
- Color and intensity are big topics in CRT design.

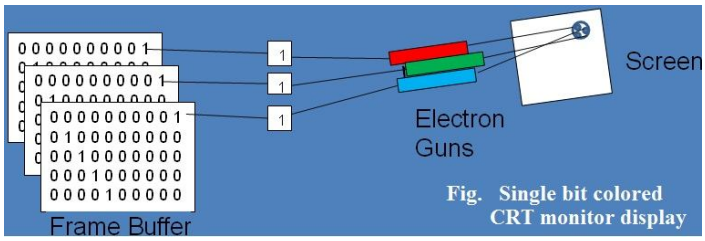
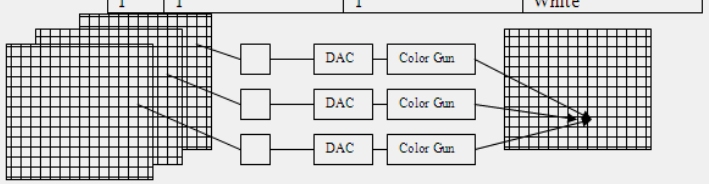


Fig. Single bit colored CRT monitor display

RED	GREEN	BLUE	Color
0	0	0	Black
1	0	0	Red
0	1	0	Green
0	0	1	Blue
1	1	0	Yellow
0	1	1	Cyan
1	0	1	Magenta
1	1	1	White



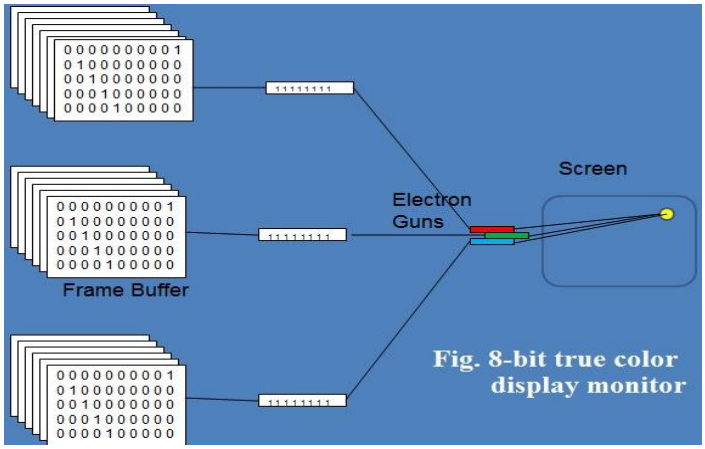


Fig. 8-bit true color display monitor

- **Numerical-1**: If a pixel is accessed from the frame buffer with an average access time of 300ns then will this rate produce an un-flickering effect for the screen size: 640 x 480.

  Solution

  The size of the screen = 640 x 480
  The average access time of one pixel = 300ns
  Thus, the time required for total pixel to show the image on the full screen = 640 x 480 x 300ns
  = 640 x 480 x 300 x10^{-9} sec = 0.09216 sec

  Now, The frequency of cycle for image = 1/ 0.09216 sec = 10.86 frame / sec (since, f = 1/t)

  As we know that the minimum number of frame on monitor must be more than 60fps for the un-flicker image display. Hence, we conclude that this monitor has fickleness.

- **Numerical-2:** If the total intensity achievable for a pixel is 256 and the screen resolution is 640 x 480. What will be the size of the frame buffer?

  Solution

  There is only one frame buffer with pixel intensity 256 = 2^8
  Thus, the number of bits required for the screen of size 640 x 480 x 8 = 2457600 bits = 300 KB.

- **Numerical-3:** What is the time required to display a pixel on the monitor of size 1024 X 768 with refresh rate 60 Hz.

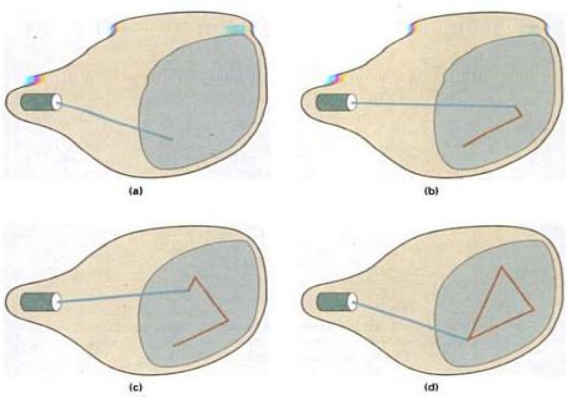
  Solution

  To refresh a display of 1024 * 768 pixels at a moderate refresh rate of 60 Hz requires

  Now, the time required to display a pixel on monitor is = 1/(1024 * 768 * 60) seconds = 21 ns.

2.4 Random Graphics or Vector Graphics

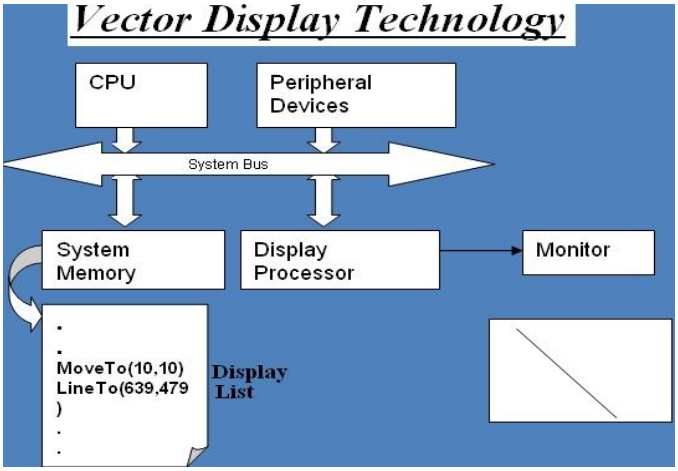
* Vector graphics (also called ***geometric modeling*** or ***object oriented graphics***) is the use of geometrical primitives such as points, lines, curves, and polygons, which are all based upon mathematical equations to represent images in computer graphics. It has complexity on drawing the complex images.
* The vector graphic system is seems on oscillators, medical diagnosis monitors etc.
* All modern current computer video displays *translate vector representations* of an image to a raster format. The raster image, containing a value for every pixel on the screen, is stored in memory.



2.4.1 Vector Display

* Developed in 60's also called ***random scan***, a ***stroke***, a ***line drawing*** or a ***calligraphic display***.
* This technique uses the geometrical shapes for constructing the image & hence if we require changing some portion of image in screen then it is not necessary to scan the whole screen as the raster system, we simply perform the change in that area only. Thus, it is more faster technique.
* In this technology we have to change the whole basic geometric shape in same manner so the special effects like shadowing are not possible.
* The image on the CRT's face must be constantly redrawn, refreshed, or updated.
* The two primary problems with vector displays are that they required constant updates to avoid fading, thus limiting the drawn scene's complexity, and they only drew wire frames.

2.4.2 Vector Display Technology



- The architecture of vector display technology consists of a central processing unit, display processor, a monitor, system memory and peripheral devices such as mouse and key board.
- A display processor is also called a *display processing unit* or *graphics controller*, which totally responsible for picture draw on the screen according to the command line stored on system memory.
- The application program and *graphics subroutine package* both reside in the system memory. A graphics subroutine package creates a *display list.* A portion of the system memory where display list resides is called a *refresh buffer.*
- A display list contains point and line plotting commands with end point coordinates as well as character plotting commands.

Advantages

- Can produce smooth output with high resolution & better time interval.
- No problem of stair case effect like raster scan display method because the random or vector display method use direct line drawing primitive or algorithms not the frame buffer.
- Better than raster for animation, requires only end point information.

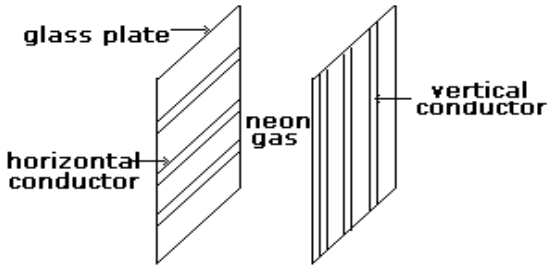
Limitations

- Can't fill area with patterns and manipulate bits
- Refreshing an image depends upon its complexity (more lines takes, longer time), flicker if complex image.

2.5 Flat Panel Display

* The term flat panel display refers to a class of video display devices that have reduced volume, weight, and power requirements compared to CRT.
* Thinner flat panel display can able to hang on a wall or able to wear on wrist.
* We can write on 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)

2.5.1 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-discharge display***, where region between two glass plates is filled with a mixture of gases such as neon, xenon.
- Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh pixel positions 60 times per second.
- 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 gray scale.
- The intermediately filling gas is a light emitter as it converts the supplied voltage energy into the light energy.

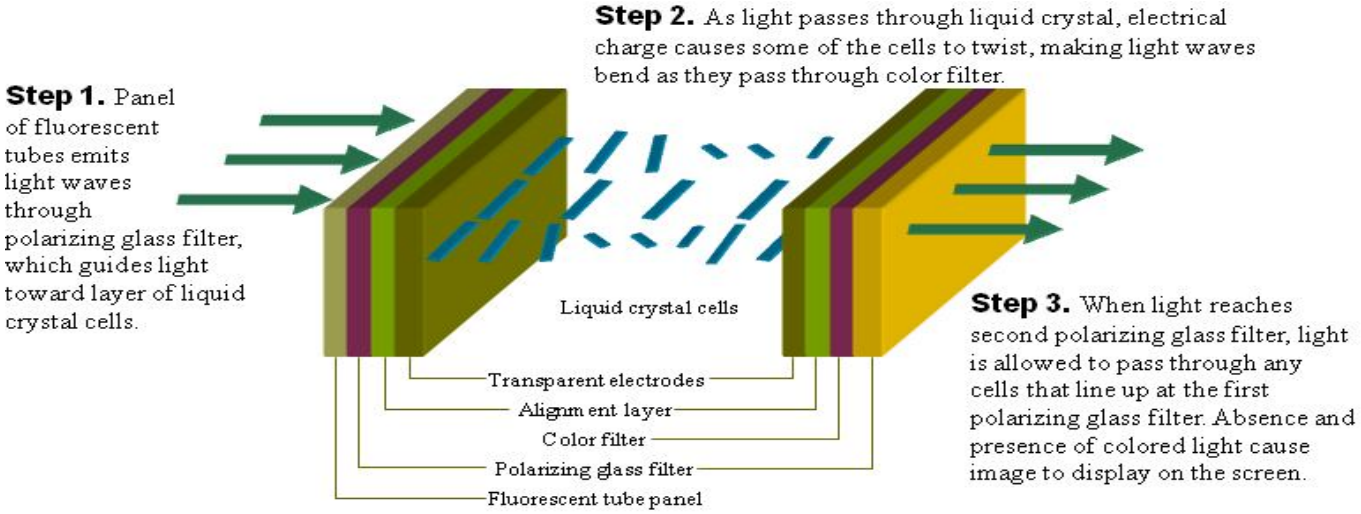
2.5.2 Non-Emissive Display

- Non emissive displays (or non-emitters but the orientation of supplied light is changed to provide the picture pattern on display monitor) 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 (LCD).

2.6 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.
* 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.

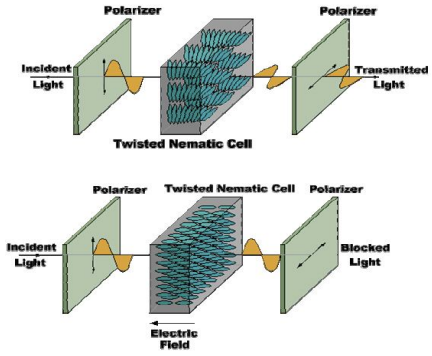
* Picture definitions are stored in refresh buffer, and the screen is refreshed at the rate of 60 frames per second.
* Back lighting is also commonly applied using solid state electronic devices, so that the system is no completely dependent on out light sources.
* Colors can be displayed by using different materials or dyes and by placing a triad of color pixels at each screen location.



2.6.1 LCD operation

* 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.*
* LCD displays have a native resolution.

LCD Off and LCD On state



CHAPTER – 3

Two Dimensional Algorithms

In case of the two dimensional algorithm, we mostly deal with the line, circle, ellipse etc as they are the basic output primitives.

3.1 Direct and Incremental Line Drawing Algorithms

- To draw a straight line we have to fixed two points: starting and ending point then we required to find out the intermediate points

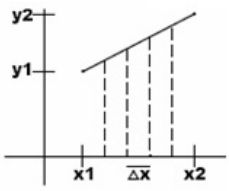
$$y = mx + b \quad (1)$$

- That is, we required to find out coordinate of pixels along the line path
- The location/coordinate of pixel along the line path can be calculated by the reference of equation of straight line

$$m = \frac{y2 - y1}{x2 - x1} \quad (2)$$

- Where **m** is a slope and **b** is y-intercept.
- Assume that two end points of line segment are say (x1,y1) and (x2,y2)

$$b = y1 - mx1 \quad (3)$$



- Them we evaluate slope **m** and **y** intercept **b** as
- Since line is a sequence of points we have to put many pixels between two end points
- In general while value of x coordinate change then y coordinate also change
- That is if Δx is an increment in x, we have to determine, increment in y that it Δy on the reference of slope m
- Assume that Δx be increment in x and Δy be the increment in Δy then

$$y = mx + b$$
$$\text{or, } y + \Delta y = m(x + \Delta x) + b$$
$$\text{or, } y + \Delta y = mx + b + m\Delta x$$
$$\text{or, } y + \Delta y = y + m\Delta x$$
$$\therefore \Delta y = m\Delta x \quad (4)$$

- This implies that is x is the increment by Δx then y is increment with mΔx. That is

$$y_{k+1} = y_k + m\Delta x \quad (5)$$

- On the reference of eqn 4 , we can obtain the x-interval Δx corresponding to the specified y-interval Δy as

$$\Delta x = \frac{\Delta y}{m} \quad (6)$$

- This implies that, if y is increment by Δy then x increment by
  That is

$$x_{k+1} = x_k + \frac{1}{m}\Delta y \quad (7)$$

3.1.1 Mathematical Analysis (Mathematical Interpretation)

- If |m|<1 (i.e. slope is less than 45°) then it gives small vertical deflection but significant horizontal deflection
- If |m|>1 (i.e. slope is greater than 45°) then it gives small horizontal deflection but significant vertical deflection
- If |m|=1 (i.e. slope is 45°) then it gives equal horizontal and vertical deflection
- If |m|=0(i.e. slope is 0°) then it only gives horizontal deflection. That is, line is parallel to x-axis
- If |m|=∞ (i.e. slope is 90°) then it only gives vertical deflection. That is line is parallel to y-axis.
- This is fundamental algorithm for drawing line

3.1.2 Drawbacks

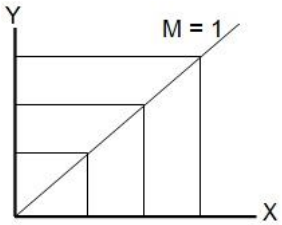
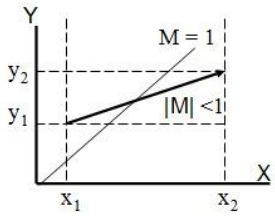
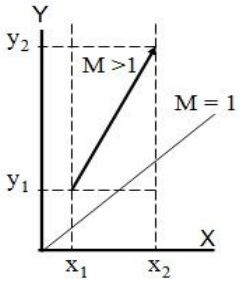
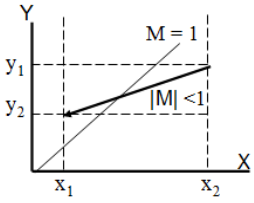
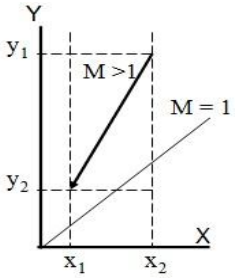
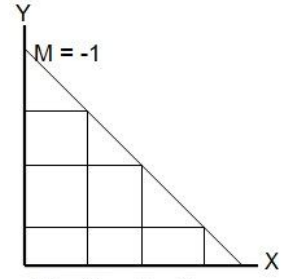
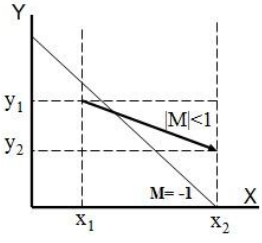
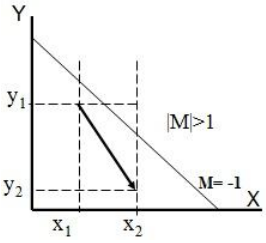
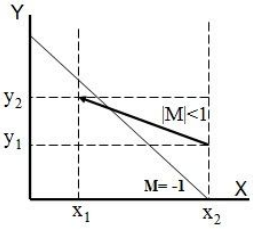
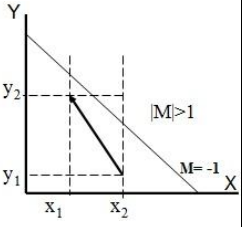
The main drawbacks of this algorithm are

- It required floating point addition and multiplication at each step while evaluating new position of pixel to put
- In each computational step, coordinate value of new pixel position required to round off which leads staircase effect.

Note: - In this method we calculate the value of 'm' in each increment or move but in case of the DDA algorithm we calculate 'm' in only one time. In this direct method, we draw the line only interpreting the nature of equation but not analyze the case of $\Delta x > \Delta y$ or $\Delta x < \Delta y$ as DDA.

3.2 Digital Differential Analyzer (DDA) Algorithm

In the DDA algorithm, we have to approximate the value of each pixel in whole number if any value of x-coordinate or y-coordinate is in fractional form by the addition or subtraction of the slope (m). Here, we approximate the value by the round-off or by counting the selling or floor value. This is the main disadvantage of the DDA algorithm of deviate the line from its exact position.

	Moving Left to Right		**Moving Right to Left**	
	Slope (m) < 1	**Slope (m) > 1**	**Slope (m) < 1**	**Slope (m) > 1**
Positive Slope  *Fig. Positive Slope*	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
Negative Slope  *Fig. Negative Slope*	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

Note: The DDA algorithm is a faster method for calculating the pixel position than that the use of direct method of line drawing method by using y = mx + c. The DDA method is seems good as it do not need to calculate the slope (m), in each iteration like the direct method.

Advantages of DDA algorithm

- DDA algorithm is faster method for calculating pixel position than simple line drawing algorithm
- It avoids directly deal of equation of straight line y= mx + c
- If avoids the floating point addition and subtraction in the simple algorithm

Limitations in DDA Algorithm

- The slope "m" is usually stored in floating point number
- There could be round off error
- The line will move away from the true line path, especially when it is long due to successive round of error.

Example-1: Digitize the line with end points (2, 1) and (8, 3) using DDA.

Solution

Here,

Starting point of line = (x1, y1) = (2, 1) and
Ending point of line = (x2, y2) = (8, 3)
Thus, slope of line, m = Δy / Δx = *y2-y1 / x2-x1* = (3-1) / (8-2) = 1/3 =0.3333
As the given points, it is clear that the line is moving left to right with the slope
m = 1/3 <1

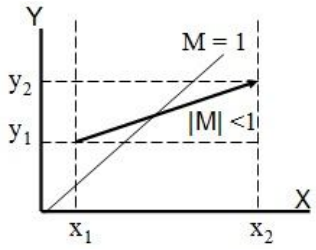
Thus,

Δx=1

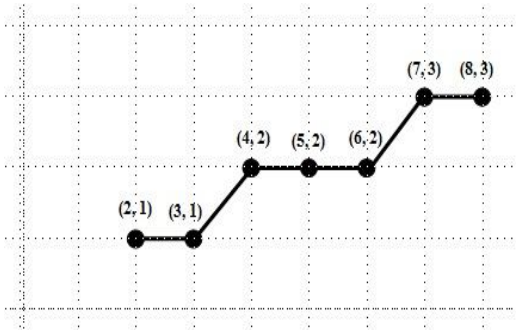
Δy=m

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

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



X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
2	1	(2,1)
2+1 = 3	1+0.3333 = 1.333 ≈ 1	(3, 1)
3+1 = 4	1+2 *0.3333 = 1.666 ≈ 2	(4, 2)
4+1 = 5	1+3*0.3333 = 1.999 ≈ 2	(5, 2)
5+1 = 6	1+4*0.3333 = 2.333 ≈ 2	(6, 2)
6+1 = 7	1+5*0.3333 = 2.666 ≈ 3	(7, 3)
7+1 = 8	1+6*0.3333 = 2.999 ≈ 3	(8, 3)



Example-2: Digitize the line with end points (3, 7) and (8, 3) using DDA.

Solution

Here,

Starting point of line = (x1, y1) = (3, 7) and
Ending point of line = (x2, y2) = (8, 3)
Thus, slope of line, m = Δy / Δx = *y2-y1 / x2-x1* = (3-7) / (8-3) = - 4/5 = - 0.8

As the given points, it is clear that the line is moving left to right with the negative slope, |m| = 0.8 <1

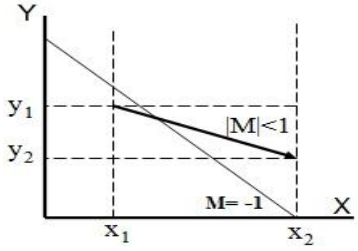
Thus,

Δx=1

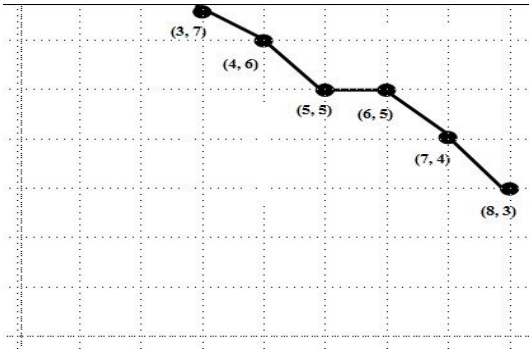
Δy= - m

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

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



X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
3	7	(3, 7)
3+1 = 4	7 - 0.8 = 6.2 ≈ 6	(4, 6)
4+1 = 5	7 – 2*0.8 = 5.4≈ 5	(5, 5)
5+1 = 6	7 – 3*0.8 = 4.6 ≈ 5	(6, 5)
6+1 = 7	7 – 4*0.8 = 3.8 ≈ 4	(7, 4)
7+1 = 8	7 – 5*0.8 = 3	(8, 3)



3.3 Bresenham's Line Algorithm (BLA)

The BLA is a more efficient method used to plot pixel position along a straight-line path.

3.3.1 Advantage of BLA over DDA

- In DDA algorithm each successive point is computed in ***floating point***, so it requires ***more time*** and ***more memory space***. While in BLA each successive point is calculated in ***integer value*** or ***whole number***. So it requires less time and less memory space.
- In DDA, since the calculated point value is floating point number, it should be rounded at the end of calculation but in BLA it does not need to round, so there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while in BLA line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse and other curves.

3.3.2 Bresenham's Line drawing Algorithm for slope, $|m| \leq 1$

Let us assume that pixel (x_k, y_k) is already plotted assuming that the ***sampling direction*** is along X-axis i.e. (x_k+1, y_k) or (x_k+1, y_k+1). Thus, the common equation of the line is $y = m(x_k+1) + c$.

Now,

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

$d_2 = y_{k+1} - y = y_k+1 - \{m(x_k+1) + c\}$

So, $d_1 - d_2 = [m(x_k+1) + c - y_k] - [y_k+1 - \{m(x_k+1) + c\}]$

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

Since, slope of line $(m) = \Delta y / \Delta x$, we have

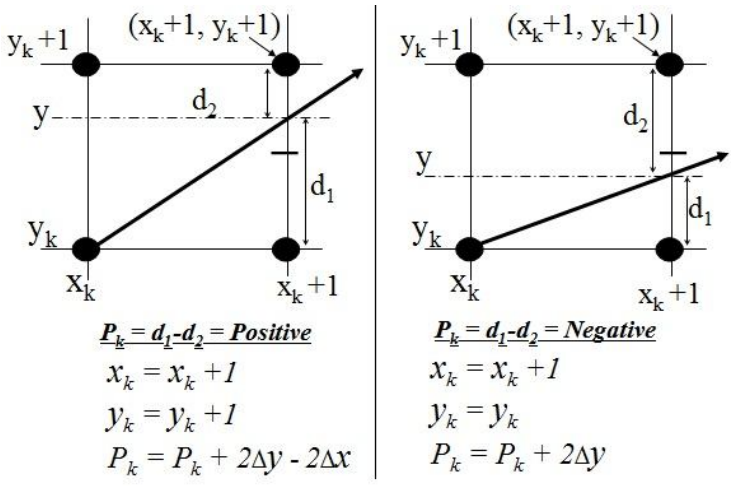


Fig. Bresenham's Line Algorithm for $|m| \leq 1$

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

$P_k = 2\Delta y\, x_k - 2\Delta x\, y_k + \{2\Delta y + 2c\,\Delta x - \Delta x\}$

$P_k = 2\Delta y\, x_k - 2\Delta x\, y_k +$ **b, *equation – I*** and $(K+1)^{th}$ step, we have

$P_{k+1} = 2\Delta y\, x_{k+1} - 2\Delta x\, y_{k+1} +$ **b, *equation – II***

Subtracting equation I & II, we have

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

As we are sampling in x-direction, increase by 1-pixel position in left to right i.e. $x_{k+1} = x_k+1$.

We have,

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

$P_{k+1} = P_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k)$, ***equation – III***

Now,

- If $P_k > 0$, (*i.e.* $d_1 - d_2$ = *Positive* or $y_{k+1} - y_k = 1$), now equation – III reduces as:
  $P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k+1$
- Else
  If $P_k < 0$, (*i.e.* $d_1 - d_2$ = *Negative* or $y_{k+1} - y_k = 0$), now equation – III reduces as:
  $P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k+1$

Note: If the starting point is (x_0, y_0), then what will be the value of initial decision parameter (P_0) when $|m| \leq 1$?

We have $P_k = \Delta x (d_1 - d_2)$

$P_k = \Delta x (2m (x_k+1) + 2c - 2y_k -1)$ [since, $d_1 - d_2 = 2m (x_k+1) + 2c - 2y_k -1$]

$P_k = \Delta x (2mx_k+2m + 2c - 2y_k -1)$

$P_k = \Delta x \{2(mx_k+ c - y_k)+2m -1)$

The line $y = mx + c$ must passes through the initial point ***(x_0, y_0)*** and hence $(mx_k+ c - y_k) =0$, we have

Thus, $P_0 = \Delta x (2m -1)$

$P_0 = \Delta x (2\Delta y / \Delta x -1)$ [since, $m = \Delta y / \Delta x$]

$P_0 = 2\Delta y - \Delta x$, which is the initial decision parameter.

Conclusion: Bresenham's Line drawing Algorithm for slope, $|m| \leq 1$

a) Calculate the starting value of the decision parameter: $P_0 = 2\Delta y - \Delta x$.
b) At each x_k along the line, starting at k = 0, perform the following test.
   - If $P_k > 0$,
     $P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k+1$
   - Else If $P_k < 0$,
     $P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k+1$

3.3.3 Bresenham's Line drawing Algorithm for slope, |m| > 1

Let us assume that pixel (x_k, y_k) is already plotted assuming that the ***sampling direction*** is along y-axis i.e. (x_k, y_k+1) or (x_k+1, y_k+1). Thus, the common equation of the line is $y_k+1 = m x + c$ or $x = \{y_k+1 – c\}/m$

Now,

$d_1 = x - x_k = \{y_k +1 – c\}/m - x_k$ and

$d_2 = x_{k+1} - x = x_k+1– \{y_k+1 – c\}/m$

So,

$d_1 - d_2 = [\{y_k +1 – c\}/m - x_k] - [x_k+1– \{\{y_k+1 – c\}/m \}]$

Or, $d_1 - d_2 = 2/m* (y_k+1) - 2c/m – 2x_k -1$

Since, slope of line (m) = $\Delta y / \Delta x$, we have

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

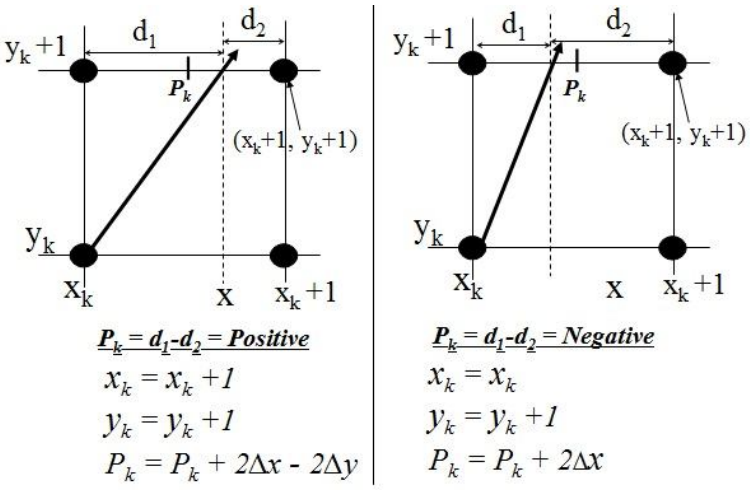


Fig. Bresenham's Line Algorithm for |m| > 1

$P_k = 2\Delta x\ y_k - 2\Delta y\ x_k + \{2\Delta x - 2c\ \Delta x - \Delta y\}$
$P_k = 2\Delta x\ y_k - 2\Delta y\ x_k +$ **b, *equation – I*** and $(K+1)^{th}$ step, we have
$P_{k+1} = 2\Delta x\ y_{k+1} - 2\Delta y\ x_{k+1} +$ **b, *equation – II***

Subtracting equation I & II, we have
$P_{k+1} - P_k = 2\Delta x\ (y_{k+1} - y_k) - 2\Delta y\ (x_{k+1} - x_k)$

As we are sampling in y-direction, increase by 1-pixel position in lower to upper position i.e. $y_{k+1} = y_k + 1$.
We have,
$P_{k+1} = P_k + 2\Delta x\ (y_k + 1 - y_k) - 2\Delta y\ (x_{k+1} - x_k)$
$P_{k+1} = P_k + 2\Delta x - 2\Delta y\ (x_{k+1} - x_k)$, ***equation – III***

Now,

- If $P_k > 0$, (*i.e.* $d_1 - d_2$ = *Positive* or $x_{k+1} - x_k = 1$), now equation – III reduces as:
  $P_{k+1} = P_k + 2\Delta x - 2\Delta y$, and $x_{k+1} = x_k + 1$ & $y_{k+1} = y_k + 1$
- Else
  If $P_k < 0$, (*i.e.* $d_1 - d_2$ = *Negative* or $x_{k+1} - x_k = 0$), now equation – III reduces as:
  $P_{k+1} = P_k + 2\Delta x$, and $x_{k+1} = x_k$ & $y_{k+1} = y_k + 1$

Note: If the starting point is (x_0, y_0), then what will be the value of initial decision parameter (P_0) when $|m| > 1$?

We have
$P_k = \Delta y\ (d_1 - d_2)$
$P_k = \Delta y\ \{2/m^*\ (y_k + 1) - 2c/m - 2x_k - 1\}$ [since, $d_1 - d_2 = 2/m^*\ (y_k + 1) - 2c/m - 2x_k - 1$]
$P_k = \Delta y\ \{2y_k/m + 2/m - 2c/m - 2x_k - 1\}$
$P_k = \Delta y\ [2/m\ \{y_k - c - mx_k\} + 2/m - 1]$
The line $y = mx + c$ must passes through the initial point ***(x_0, y_0)*** and hence $y_k - c - mx_k = 0$, we have
Thus,
$P_0 = \Delta y\ (2/m - 1)$
$P_0 = \Delta y\ (2\Delta x/\Delta y - 1)$ [since, $m = \Delta y / \Delta x$]
$P_0 = 2\Delta x - \Delta y$, which is the initial decision parameter.

Conclusion: Bresenham's Line drawing Algorithm for slope, $|m| > 1$

c) Calculate the starting value of the decision parameter: $P_0 = 2\Delta y - \Delta x$.
d) At each y_k along the line, starting at $k = 0$, perform the following test.
   - If $P_k > 0$,
     $P_{k+1} = P_k + 2\Delta x - 2\Delta y$, and $x_{k+1} = x_k + 1$ & $y_{k+1} = y_k + 1$
   - Else If $P_k < 0$,
     $P_{k+1} = P_k + 2\Delta x$, and $x_{k+1} = x_k$ & $y_{k+1} = y_k + 1$

Example-1: Digitize the line with end points (20, 10) and (30, 18) using BLA.
Solution
Here, Starting point of line = (x1, y1) = (20, 10) and
Ending point of line = (x2, y2) = (30, 18)
Thus, slope of line, $m = \Delta y / \Delta x = y2\text{-}y1 / x2\text{-}x1 = (18\text{-}10) / (30\text{-}20) = 8/10$
As the given points, it is clear that the line is moving left to right with the negative slope,
$|m| = 0.8 < 1$

Thus,

The initial decision parameter (P_0) = $2\Delta y - \Delta x = 2*8 - 10 = 6$

Since, for the Bresenham's Line drawing Algorithm of slope, $|m| \leq 1$, we have

- If $P_k > 0$,
  $P_{k+1} = P_k + 2\Delta y - 2\Delta x$, and $y_{k+1} = y_k + 1$ & $x_{k+1} = x_k + 1$
- Else
  If $P_k < 0$,
  $P_{k+1} = P_k + 2\Delta y$, and $y_{k+1} = y_k$ & $x_{k+1} = x_k + 1$

Thus,

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	6	20+1 = 21	11	(21, 11)
1.	6 + 2*8 -2*10 = 2	21+1 = 22	12	(22, 12)
2.	2 + 2*8 -2*10 = -2	22+1 = 23	12	(23, 12)
3.	-2 + 2*8 = 14	23+1 = 24	13	(24, 13)
4.	14 + 2*8 -2*10 = 10	24+1 = 25	14	(25, 14)
5.	10 + 2*8 -2*10 = 6	25+1 = 26	15	(26, 15)
6.	6 + 2*8 -2*10 = 2	26+1 = 27	16	(27, 16)
7.	2 + 2*8 -2*10 = -2	27+1 = 28	16	(28, 16)
8.	-2 + 2*8 = 14	28+1 = 29	17	(29, 17)
9.	14 + 2*8 -2*10 = 10	29+1 = 30	18	(30, 18)

Example-2: Digitize the line with end points (1, 0) and (3, 3) using BLA.

Solution

Here, Starting point of line = (x1, y1) = (1, 0) and

Ending point of line = (x2, y2) = (3, 3)

Thus, slope of line, $m = \Delta y / \Delta x = y2-y1 / x2-x1 = (3-0) / (3-1) = 3/2$

As the given points, it is clear that the line is moving left to right with the negative slope, $|m| = 1.5 > 1$

Thus, the initial decision parameter (P_0) = $2\Delta x - \Delta y = 2*2 - 3 = 1$

Since, for the Bresenham's Line drawing Algorithm of slope, $|m| > 1$, we have

- If $P_k > 0$,
  $P_{k+1} = P_k + 2\Delta x - 2\Delta y$, and $x_{k+1} = x_k + 1$ & $y_{k+1} = y_k + 1$
- Else If $P_k < 0$,
  $P_{k+1} = P_k + 2\Delta x$, and $x_{k+1} = x_k$ & $y_{k+1} = y_k + 1$

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	1	2	1	(2, 1)
1.	1 + 2*2 -2*3 = -1	2	1+1 = 2	(2, 2)
2.	-1 + 2*2 = 3	3	2+1 = 3	(3, 3)

3.4 Midpoint Circle Algorithm

- Let us consider a circle centered at origin. Assume that (x, y) is any point on the circle we can trivially compute other seven points on the circle as shown in figure.
- The slope of the circular curve is less than zero inside the circle, equal to zero on the circle boundary and greater than zero on outside the circle.

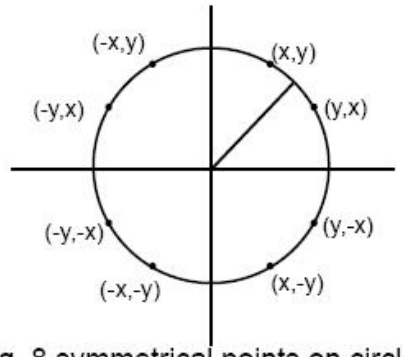


Fig. 8 symmetrical points on circle

- We can generate all pixel positions around a circle by calculating only the points within the first octants & positions in the other seven octants are obtained by symmetry.
- Here, we have to calculate all the pixel position from (0, r) up to both the co-ordinate either equal or x-coordinate is greater than that of the y-coordinate but the final value cannot be counted in the ***octant*** coordinate (i.e. only for checking) because this point is the starting pixel for next octant.

To apply the midpoint method, we define a circle function:

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

The relative position of any point (x, y) can be determined by checking the sign of the circle function as: -

i. $f_{circle}(x, y) < 0$, the point (x, y) is inside the circle boundary.
ii. $f_{circle}(x, y) = 0$, the point (x, y) is on the circle boundary.
iii. $f_{circle}(x, y) > 0$, the point (x, y) is outside the circle boundary.

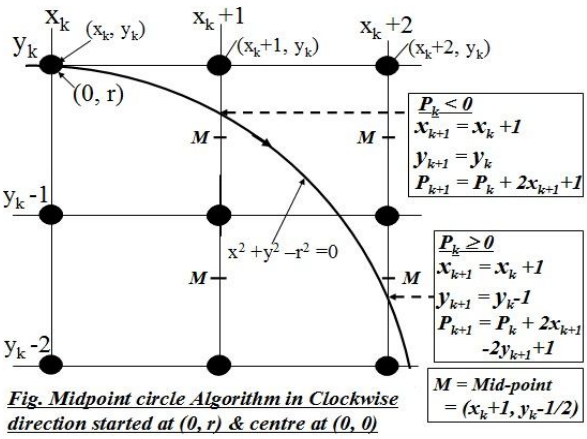


Fig. Midpoint circle Algorithm in Clockwise direction started at (0, r) & centre at (0, 0)

Suppose, (x_k, y_k) is the pixel plotted, then the next pixel (x_{k+1}, y_{k+1}) will be either (x_{k+1}, y_k) or (x_{k+1}, y_{k-1}) as shown in figure.

Here, the mid-point (m) = $(x_k+1, y_k-1/2)$.

Thus, our decision parameter $(P_k) = f_{circle}(x_k+1, y_k-1/2) = (x_k+1)^2 + (y_k-1/2)^2 - r^2$ → I, is the circle function evaluated at midpoint.

Also, the successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluation the circle function at sampling position $x_{k+1}+1 = x_k+2$. Thus, the next pixel to plot will either be $(x_{k+1}+1, y_{k+1})$ or $(x_{k+1}+1, y_{k+1}-1)$.

Also, the midpoint here is $(x_{k+1}+1, y_{k+1}-1/2)$ and thus, our decision parameter $(P_{k+1}) = f_{circle}(x_{k+1}+1, y_{k+1}-1/2)$

$P_{k+1} = (x_{k+1}+1)^2 + (y_{k+1}-1/2)^2 - r^2$
$P_{k+1} = \{(x_k+1)+1\}^2 + (y_{k+1}-1/2)^2 - r^2$
$P_{k+1} = (x_k+1)^2 + 2(x_k+1) + 1 + (y_{k+1}-1/2)^2 - r^2$ → II

Now, subtracting I from II, we have

$$P_{k+1} - P_k = 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

If $P_k < 0$, the midpoint is inside the circle and the pixel on the scan line y_k closer to the circle boundary. Otherwise, the midpoint is outside or on the circle boundary, and we select the pixel on scan line y_k-1.

Case – I: $P_k < 0$

$x_{k+1} = x_k + 1$
$y_{k+1} = y_k$
$P_{k+1} = P_k + 2(x_k+1) + 1$
$P_{k+1} = P_k + 2x_{k+1} + 1$

Case – I: $P_k \geq 0$

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

$P_{k+1} = P_k + 2(x_k+1) + [(y_k - 1)^2 - y_k^2)] - (y_k - 1 - y_k) + 1$
$= P_k + 2(x_k+1) + [(y_k^2 - 2y_k + 1 - y_k^2)] - (y_k - 1 - y_k) + 1$

$= P_k+2(x_k+1) - 2y_k +1 +1 + 1$

$= P_k+2(x_k+1) – 2(y_k +1) +1$

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

3.4.1 Initial decision parameter (P_0) for circle

The initial decision parameter is obtained by evaluating the circle function at the start position $(x_0, y_0) = (0, r)$. Here, the next pixel will either be (1, r) or (1, r-1) where the midpoint is (1, r -1/2). Thus, the initial decision parameter is given by:

$P_0 = f_{circle}(1, r-1/2) = 1^2 + (r -1/2)^2 – r^2 = 5/4 – r \approx 1 – r$

Thus, $P_0 = 1 – r$

Conclusion

1. Calculate the initial decision parameter $(P_0) = 1 – r$
2. If $P < 0$

Plot $(x_k +1, y_k)$

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

Else $(P \geq 0)$

Plot $(x_k +1, y_k - 1)$

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

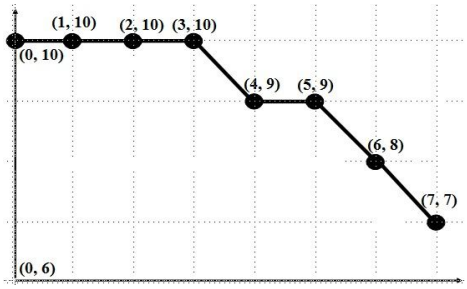
Note : If the center of the circle is not at origin then first calculate the octant points of the circle in the same way as the center at origin & then add the given circle center on each calculated pixels.

Example: Digitize a circle with radius 10.

Solution

Here, the initial decision parameter $(P_0) = 1 – r = 1-10 = - 9$

Since, for the Midpoint Circle Algorithm of initial point (0, r) & center at origin (0, 0) rotating at clockwise direction, we have



If $P < 0$

Plot $(x_k +1, y_k)$

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

Else $(P \geq 0)$

Plot $(x_k +1, y_k - 1)$

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

Thus,

k	P_k	X_{k+1}	Y_{k+1}	(X_{k+1}, Y_{k+1})
0.	-9	0+1 = 1	10	(1, 10
1.	-9 + 2*1 +1= -6	1+1 = 2	10	(2, 10)
2.	-6 + 2*2 +1= -1	2+1 = 3	10	(3, 10)
3.	-1 + 2*3 +1= 6	3+1 = 4	9	(4, 9)
4.	6 + 2*4 - 2*9 +1= -3	4+1 = 5	9	(5, 9)
5.	-3 + 2*5 +1= 8	5+1 = 6	8	(6, 8)
6.	8 + 2*6 - 2*8 +1= 5	6+1 = 7	7	(7, 7) (Halt)

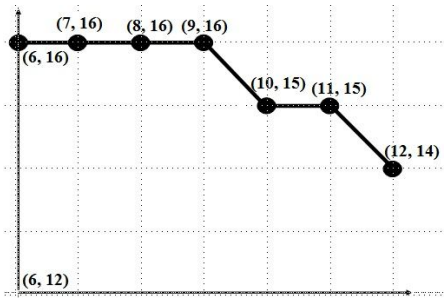
Example: Digitize a circle with radius 9 and center at (6, 7).

Solution

Here, the initial decision parameter (P_0)

$P_0 = 1 - r = 1 - 9 = -8$

Since, for the Midpoint Circle Algorithm of starting point (0, r) & centre at origin (0, 0) rotating at clockwise direction, we have



If $P < 0$

Plot $(x_k + 1, y_k)$

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

Else $(P \geq 0)$

Plot $(x_k + 1, y_k - 1)$

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

Thus,

k	**P_k**	**X_{k+1}**	**Y_{k+1}**	**$(X_{k+1}, Y_{k+1})_{At\ (0, 0)}$**	**$(X_{k+1}, Y_{k+1})_{At\ (6, 7)}$**
0.	-8	0+1 = 1	9	(1, 9)	(1+6, 9+7) = (7, 16)
1.	-8 + 2*1 +1= -5	1+1 = 2	9	(2, 9)	(2+6, 9+7) = (8, 16)
2.	-5 + 2*2 +1= 0	2+1 = 3	8	(3, 8)	(3+6, 8+7) = (9, 15)
3.	0 + 2*3 - 2*8 +1= -9	3+1 = 4	8	(4, 8)	(4+6, 8+7) = (10, 15)
4.	-9 + 2*4 +1= 0	4+1 = 5	7	(5, 7)	(5+6, 8+7) = (11, 15)
5.	0 + 2*5 - 2*7 +1= -3	5+1 = 6	7	(6, 7) (Halt)	(6+6, 7+7) = (12, 14)
6.	-3 + 2*6 +1= 10	6+1 = 7	6	(7, 6) (Only for checking)	

3.5 Midpoint Ellipse Algorithm

Our approach here is similar to that used in displaying a raster circle but the ellipse has 4-way symmetry. The midpoint ellipse method is applied throughout the first quadrant in two parts or region as shown in figure. The region-1 just behaves as the circular property and the region-2 is slightly straight curve.

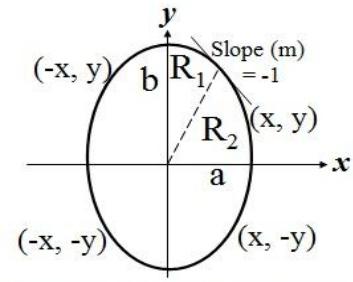


Fig. Ellipse with 4-symmetry & two region in each quadrant

We have,

The equation of ellipse, whose centre at (0, 0) is

$$x^2/a^2 + y^2/b^2 = 1$$

Hence, we define the ellipse function for centre at origin (0, 0) as:

$F_{ellipse}(x, y) = x^2b^2 + y^2a^2 - a^2b^2$

This has the following properties:

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

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

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

Starting at **(0, b),** we take unit steps in the ***x*** direction until we reach the boundary between region 1 and region 2. Then we switch to unit steps in the **y** direction over the remainder of the

curve in the first quadrant. At each step, we need to test the value of the slope of the curve. The ellipse slope is calculated by differentiating the ellipse function as:

$2xb^2 + 2ya^2 * dy/dx = 0$ Or $dy/dx = - 2xb^2 / 2ya^2$

At the boundary between region 1 and region 2, **dy/dx** = - 1 and $2xb^2 = 2ya^2$. Therefore, we move out of region **1** whenever $2xb^2 \geq 2ya^2$.

3.5.1 For Region – 1: Condition ($2xb^2 \geq 2ya^2$)

Assuming that the position (x_k, y_k) has been plotted, we determine next position (x_{k+1}, y_{k+1}) as either (x_{k+1}, y_k) or (x_{k+1}, y_{k-1}) by evaluating the decision parameter $P1_k$ as:

$P1_k = F_{ellipse}(x_k+1, y_k-1/2)$

$= b^2(x_k+1)^2 + a^2(y_k-1/2)^2 - a^2 b^2$ --- I

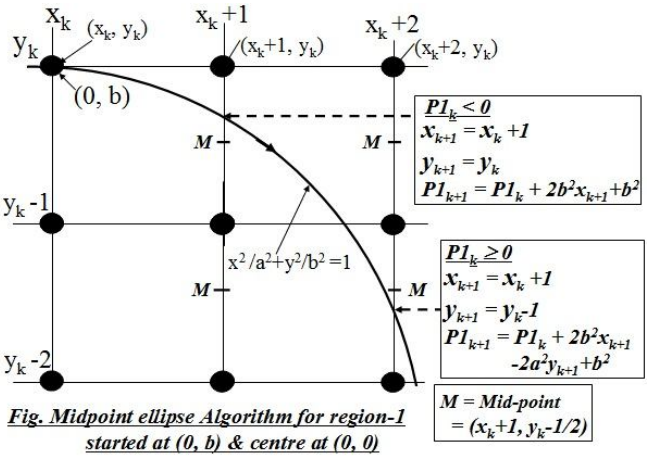


Fig. Midpoint ellipse Algorithm for region-1 started at (0, b) & centre at (0, 0)

At next sampling position, the decision parameter will be

$P1_{k+1} = F_{ellipse}(x_{k+1}+1, y_{k+1}-1/2)$

$= b^2(x_{k+1}+1)^2 + a^2(y_{k+1}-1/2)^2 - a^2 b^2$

$= b^2\{(x_k+1)+1\}^2 + a^2(y_{k+1}-1/2)^2 - a^2 b^2$

$= b^2\{(x_k+1)^2 + 2(x_k+1) + 1\} + a^2(y_{k+1}-1/2)^2 - a^2 b^2$ --- II

Now, subtracting I from II, we have

$$P_{k+1} - P_k = 2b^2(x_k+1) + a^2[(y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)] + b^2$$

If $P_k < 0$, the midpoint is inside the ellipse and the pixel on the scan line y_k closer to the ellipse boundary. Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line y_k-1.

Case – I: $\mathbf{P1_k < 0}$

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

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

$\mathbf{P_{k+1} = P_k + 2b^2(x_k+1) + b^2}$

$\mathbf{P_{k+1} = P_k + 2b^2x_{k+1} + b^2}$

Case – I: $\mathbf{P_k \geq 0}$

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

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

$\mathbf{P_{k+1} = P_k + 2b^2(x_k+1) + a^2[(y_k - 1)^2 - y_k^2)] - a^2(y_k - 1 - y_k) + b^2}$

$\mathbf{= P_k + 2b^2(x_k+1) + a^2[(y_k^2 - 2y_k + 1 - y_k^2)] - a^2(y_k - 1 - y_k) + b^2}$

$\mathbf{= P_k + 2b^2(x_k+1) - 2a^2y_k + a^2 + a^2 + b^2}$

$\mathbf{= P_k + 2b^2(x_k+1) - 2a^2(y_k + 1) + b^2}$

$\mathbf{P_{k+1} = P_k + 2b^2x_{k+1} - 2a^2y_{k+1} + b^2}$

Initial decision parameter ($P1_0$) for region-1 of ellipse

The initial decision parameter is obtained by evaluating the ellipse function at the start position $(x_0, y_0) = (0, b)$. Here, the next pixel will either be (1, b) or (1, b-1) where the midpoint is (1, b -1/2). Thus, the initial decision parameter is given by:

$P1_0 = F_{ellipse}(1, b-1/2) = b^2 + a^2(b - 1/2)^2 - a^2b^2$

$= b^2 - a^2b^2 + a^2 * 1/4$

Thus, $P1_0 = b^2 - a^2b^2 + a^2/4$

3.5.2 For Region – 2: Condition ($2xb^2 < 2ya^2$)

Assuming that the position (x_k, y_k) has been plotted, we determine next position (x_{k+1}, y_{k+1}) as either (x_{k+1}, y_{k-1}) or (x_k, y_{k-1}) by evaluating the decision parameter $P2_k$ as:

$P2_k = F_{ellipse}(x_k+1/2, y_k-1)$

$= b^2 (x_k+1/2)^2 + a^2 (y_k-1)^2 - a^2 b^2$ --- I

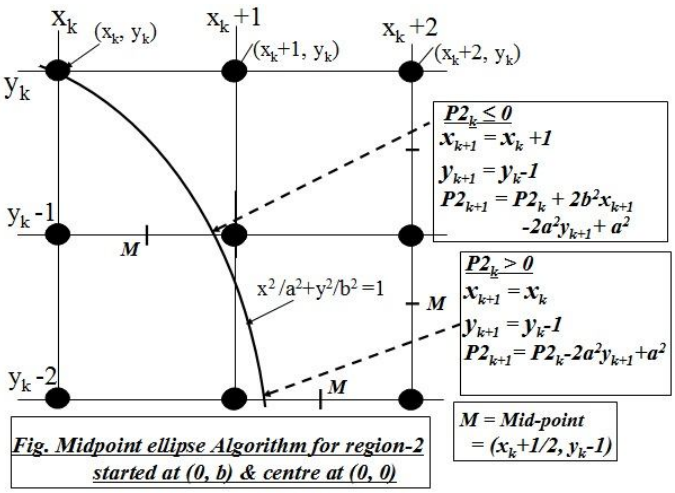


Fig. Midpoint ellipse Algorithm for region-2 started at (0, b) & centre at (0, 0)

At next sampling position, the decision parameter will be

$P2_{k+1} = F_{ellipse}(x_{k+1}+1/2, y_{k+1}-1)$

$= b^2 (x_{k+1}+1/2)^2 + a^2 (y_{k+1}-1)^2 - a^2 b^2$

$= b^2 (x_{k+1}+1/2)^2 + a^2 \{(y_k-1) - 1\}^2 - a^2 b^2$

$= b^2 (x_{k+1}+1/2)^2 + a^2 \{(y_k-1)^2 - 2(y_k-1) + 1\} - a^2 b^2$

--- II

Now, subtracting I from II, we have

$$P2_{k+1} - P2_k = b^2 [(x_{k+1}^2 - x_k^2) + (x_{k+1} - x_k)] - 2a^2 (y_k-1) + a^2$$

If $P_k < 0$, the midpoint is inside the ellipse and the pixel on the scan line x_k is closer to the ellipse boundary. Otherwise, the midpoint is outside or on the ellipse boundary, and we select the pixel on scan line x_k+1.

Case – I: $P2_k \le 0$

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

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

$P2_{k+1} = P2_k + b^2 [\{(x_k+1)^2 - x_k^2\} + (x_k+1 - x_k)] - 2a^2 (y_k-1) + a^2$

$= P2_k + b^2 [\{x_k^2 + 2x_k + 1 - x_k^2\} + (x_k+1 - x_k)] - 2a^2 (y_k-1) + a^2$

$= P2_k + b^2 [2x_k+1+1] - 2a^2 (y_k-1) + a^2$

$= P2_k + 2b^2 (x_k+1) - 2a^2 (y_k-1) + a^2$

$P2_{k+1} = P2_k + 2b^2 x_{k+1} - 2a^2 y_{k+1} + a^2$

Case – I: $P_k \ge 0$

$x_{k+1} = x_k$

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

$P2_{k+1} = P2_k - 2a^2 (y_k-1) + a^2$

$P_{k+1} = P2_k - 2a^2 y_{k+1} + a^2$

Initial decision parameter ($P2_0$) for region-2 of ellipse

When we enter region 2, the initial position ***(xo, y_0)*** is taken as the last position selected in region 1 and the initial derision parameter in region *2* is given by:

$P2_0 = F_{ellipse}(x_0+1/2, y_0-1)$

$P2_0 = b^2 (x_0+1/2)^2 + a^2 (y_0-1)^2 - a^2 b^2$

Note: After the calculation of all the points in one quadrant, determine the symmetry points in the other three quadrants as shown in figure.

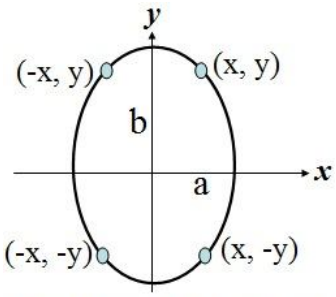


Fig. Ellipse with 4-symmetry

Example: Given input ellipse parameters r_x = a = 8 and r_y = b = 6, we illustrate the steps in the midpoint ellipse algorithm by determining raster positions along the ellipse path in the first quadrant.

$2b^2x = 0$ (with increment $2b^2 = 72$)

$2a^2y = 2a^2b$ (withincrement-$2a^2$=-128)

For region-1

The initial point for the ellipse centered on the origin is $(x_0, y_0) = (0, 6)$, and the initial decision parameter value is: $P1_0 = b^2 - a^2b^2 + a^2/4 = -332$

Successive decision parameter values and positions along the ellipse path are calculated using the midpoint method as:

k	$P1_k$	$(X_{k+1}, Y_{k+1})_{At\ (0,0)}$	$2b^2X_{k+1}$	$2a^2Y_{k+1}$
0.	-332	(1, 6)	72	768
1.	-224	(2, 6)	144	768
2.	-44	(3, 6)	216	768
3.	208	(4, 5)	288	640
4.	-108	(5, 5)	360	640
5.	288	(6, 4)	432	512
6.	244	(7, 3)	540	384

We now move out of region 1, since $2b^2X > 2a^2Y$

For region-2

The initial point is $(x_0, y_0) = (7, 3)$ and the initial decision parameter is:

$P2_0 = b^2 (x_0+1/2)^2 + a^2 (y_0-1)^2 - a^2 b^2 = -151$

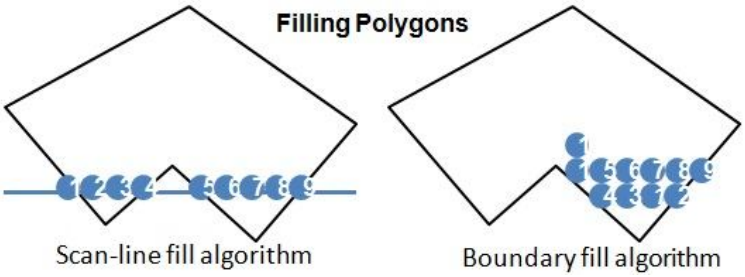
The remaining positions along the ellipse path in the first quadrant are then calculated as:

k	$P1_k$	$(X_{k+1}, Y_{k+1})_{At\ (0,0)}$	$2b^2X_{k+1}$	$2a^2Y_{k+1}$
0.	-151	(8, 2)	576	256
1.	233	(8, 1)	576	128
2.	748	(8, 0)	-	-

3.6 Filled Area Primitives

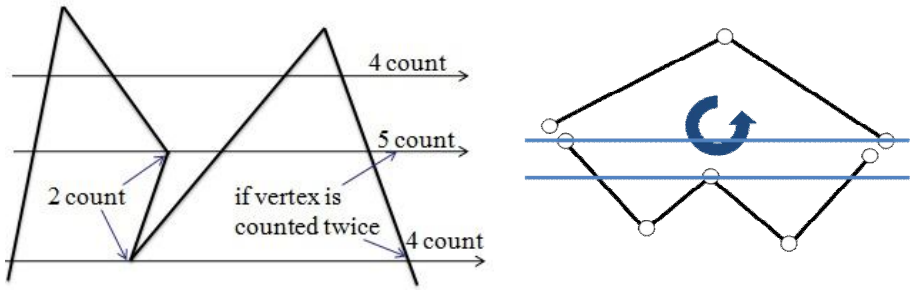
In graphics system we fill ***polygons*** or ***hardwired structure*** constructed using points, straight-line segments, curves etc and create the ***surfaces***. The main idea behind the 2D or 3D object filling procedure is that it provides us more realism on the object of interest. There are two basic approaches to area filling on raster systems.

* Determine the overlap intervals for scan lines that cross the area. Typically useful for filling polygons, circles, ellipses.
* Start from a given interior position and paint outwards from this point until we encounter the specified boundary condition useful for filling more complex boundaries, interactive painting system.

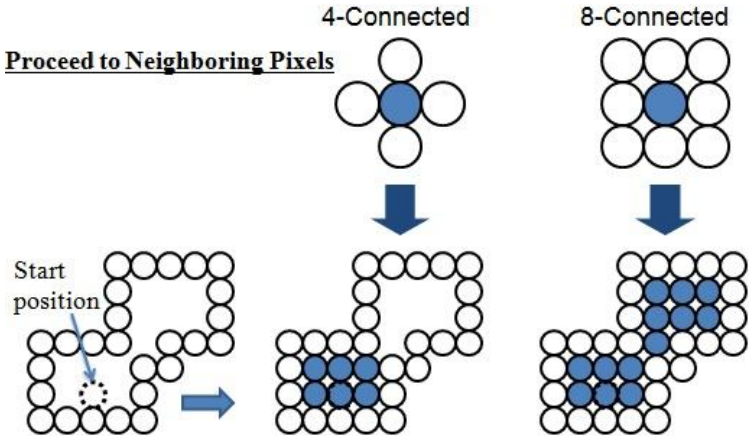


A. Scan-Line Polygon Fill Algorithm

For each scan line crossing a polygon, the area-fill algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color. In this method, the scan line must be even. At two edge connecting point called the ***vertex***, we count the scan line two to handling the problem for scan line passing through vertex, but this consideration also may creates problem for some instant as shown in figure. To handle such problem shown by count-5, we keep the vertex blank and hence the count becme-1so that overall count in that scan line become even as shown in figure.



B. Boundary Fill Algorithm



Another approach to area filling is to start at a point inside a region called the ***seed pixel*** and paint the interior outward toward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered. This method, called the boundary-till algorithm, is particularly useful in interactive painting packages, where interior points are easily selected. In boundary fill algorithm, we use ***recursion*** to fill area either in 4-connected or 8-connected way. The 4-connected method has some demerits which can solve by the 8-connected method.

Pseudo-code for boundary fill algorithm

```
Void boundaryFill (int x, y, fill_color, boundary_color)
{
  int Current;
  Current = getpixel (x, y);
   If (Current != boundary_color) && (Current != fill_color)
{
        setpixel (x, y);
        setColor (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)
    }
}
```

3.7 Flood-Fill Algorithm

Sometimes we want to fill in an area that is not defined within a single color boundary. We can paint such areas by replacing a specified interior color instead of searching for a boundary color value. This approach is called a flood-fill algorithm. We start from a specified interior point (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color. If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color. Using either a 4 connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

```
Void floodFill (int x, y, fill_color, old_color)
{
    If (getpixel (x, y) ==old_color)
{
        setpixel (x, y);
     setColor (fill_color);
        floodFill (x+1, y, fill_color, old_color)
        floodFill (x, y+1, fill_color, old_color)
        floodFill (x-1, y, fill_color, old_color)
        floodFill (x, y-1, fill_color, old_color)
    }
}
```

Note: To display the operation of boundary fill algorithm in raster display screen, the each pixel filling procedure is left to right in downward manner but not exactly as the algorithm defined because even if the frame buffer set the value according as the algorithm but display it raster display procedure.

- *What is the difference between the boundary fill & flood-fill algorithm.*
- *Write down the pseudo code procedure to show how a flood fill algorithm would fill the region using four connected definition for region pixel.*

CHAPTER – 4

2D Geometric Transformation

4.1 Basic Transformations

The orientation, size, and shape of the output primitives are accomplished with geometric transformations that alter the coordinate descriptions of objects. The basic geometric transformations are translation, rotation, and scaling. Other transformations that are often applied to objects include reflection and shear. In these all cases we consider the reference point is origin so if we have to do these transformations about any point then we have to shift these point to the origin first and then perform required operation and then again shift to that position.

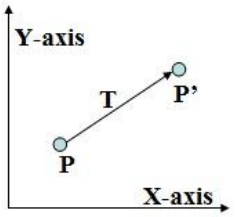
A. Translation

A translation is applied to an object by repositioning it along a straight-line path from one coordinate location to another. We translate a two-dimensional point by adding translation distances, t_x, and t_y, to the original coordinate position (x, y) to move the point to a new position (x ' , y').

$\mathbf{x' = x + t_x}$ $\mathbf{y' = y + t_y}$, where the pair (t_x, t_y) is called the ***translation vector*** or ***shift vector***.

We can write equation as a single matrix equation by using column vectors to represent coordinate points and translation vectors. Thus,

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



*Translating a point from position **P** to position **P'** With translation vector **T**.*

So we can write

$P' = P + T$

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

Translation is a ***rigid-body transformation*** that moves objects without deformation. That is, every point on the object is translated by the same amount

B. Rotation

A two-dimensional rotation is applied to an object by repositioning it along a circular path in the **xy** plane. To generate a rotation, we specify a rotation angle θ and the position (x_r, y_r,) of the rotation point (or pivot point) about which the object is to be rotated.

+ Value for 'θ' define *counter-clockwise* rotation about a point

- Value for 'θ' defines *clockwise* rotation about a point

Let (x, y) is the original point, 'r' the constant distance from origin, & 'Φ' the original angular displacement from x-axis. Now the point (x, y) is rotated through angle 'θ' in a counter clock wise direction

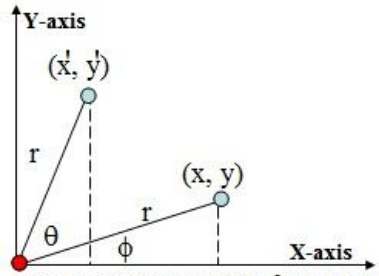


Fig. Rotating a point from position (x, y) to position (x', y ') through an angle θ about rotation point (0, 0). The original angular displacement of the point from the x axis is ϕ.

Express the transformed coordinates in terms of 'ϕ' and 'θ' as

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

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

We know that original coordinates of point in polar coordinates are

$x = r\cos\phi$

$y = r\sin\phi$

Substituting these values in (i) and (ii), we get,

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

So using column vector representation for coordinate points the matrix form would be

$P' = R \cdot P$

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

Rotation of a point about an arbitrary pivot position can be seen in the figure.

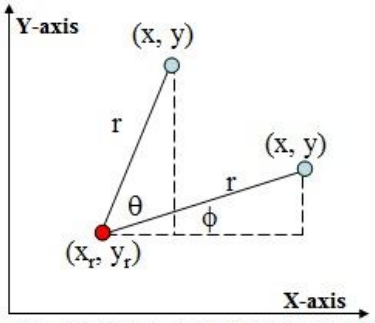


Fig. Rotating a point from position (x, y) to position (x', y ') through an angle θ about rotation point (x_r, y_r).

Here,

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

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

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1-\cos\theta & -\sin\theta \\ -\sin\theta & 1-\cos\theta \end{bmatrix} \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

Note: - This can also be achieved by translating the arbitrary point into the origin and then apply the rotation and finally perform the reverse translation.

C. Scaling

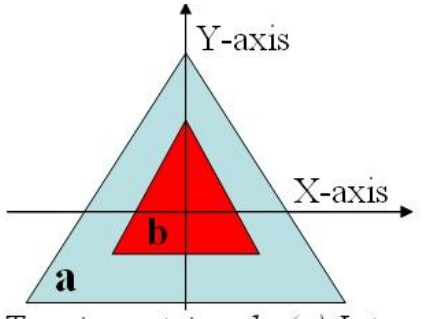
A scaling transformation alters the size of an object. This operation can be carried out for polygons by multiplying the coordinate values (x, y) of each vertex by scaling factors s_x and s_y to produce the transformed coordinates (x', y').

- s_x scales object in 'x' direction
- s_y scales object in 'y' direction

Thus, for equation form, $x' = x.\ s_x$ and $y' = y.\ s_y$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Values greater than 1 for s_x s_y produce ***enlargement***
- Values less than 1 for s_x s_y *reduce* size of object
- $s_x = s_x = 1$ leaves the size of the object ***unchanged***
- When s_x, s_y are assigned the same value $s_x = s_x =$ 3 or 4 etc then a ***Uniform Scaling*** is produced



Turning a triangle (a) Into a triangle (b) with scaling factors $s_x = -2$ and $s_y = -2$

4.2 Other Transformations

Basic transformations such as translation, rotation, and scaling are included in most graphics packages. Some packages provide a few additional transformations that are useful in certain applications. Two such transformations are reflection and shear.

A. Shearing

It distorts the shape of object in either 'x' or 'y' or both direction. In case of single directional shearing (e.g. in 'x' direction can be viewed as an object made up of very thin layer and slid over each other with the *base* remaining where it is). Shearing is a ***non-rigid-body transformation*** that moves objects with deformation.

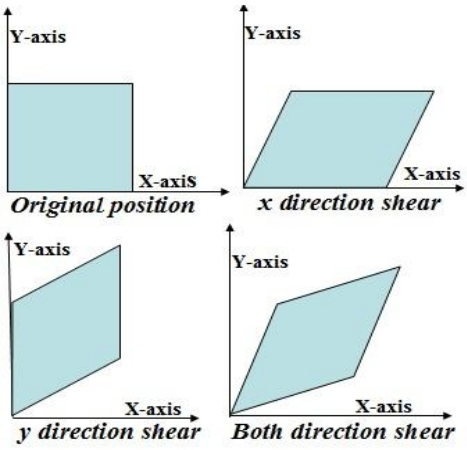


Fig. Two Dimensional shearing

In 'x' direction,

$x' = x + s_{hx} . y$

$y' = y$

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

In 'y' direction,

$x' = x$

$y' = y + s_{hy} . x$

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

In both directions,

$x' = x + s_{hx} . y$

$y' = y + s_{hy} . x$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s_{hx} \\ s_{hy} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

B. Reflection

A reflection is a transformation that produces a mirror image of an object. The mirror image for a 2D reflection is generated relative to an axis of reflection by rotating the object 180" about the reflection axis. We can choose an axis of reflection in the **xy**-plane or perpendicular to the xy plane. When the reflection axis is a line in the xy plane, the rotation path about this axis is in a plane perpendicular to the **xy**-plane. For reflection axes that are perpendicular to the **xy**-plane, the rotation path is in the xy plane.

(i) Reflection about x axis or about line y = 0

Keeps 'x' value same but flips y value of coordinate points

So $x' = x$

$y' = -y$

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

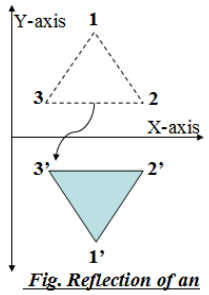


Fig. Reflection of an object about the x axis.

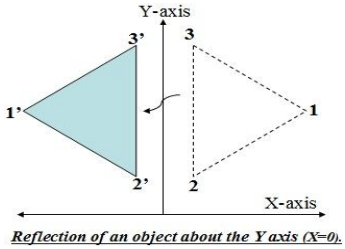
(ii) Reflection about y axis or about line x = 0

Keeps 'y' value same but flips x value of coordinate points

So $x' = -x$

$y' = y$

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



Reflection of an object about the Y axis (X=0).

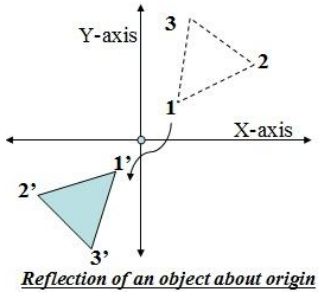
(iii) Reflection about origin

Flip both 'x' and 'y' coordinates of a point

So $x' = -x$

$y' = -y$

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



Reflection of an object about origin

(iv) Reflection about line y = x

Steps required:

- Rotate about origin in clockwise direction by 45 degree which rotates line y = x to x-axis
- Take reflection against x-axis
- Rotate in anti-clockwise direction by same angle

$$R = \begin{pmatrix} \text{Cos}\,\theta & \text{Sin}\,\theta \\ -\text{Sin}\,\theta & \text{Cos}\,\theta \end{pmatrix} \;\&\; R' = \begin{pmatrix} \text{Cos}\,\theta & -\text{Sin}\,\theta \\ \text{Sin}\,\theta & \text{Cos}\,\theta \end{pmatrix}$$

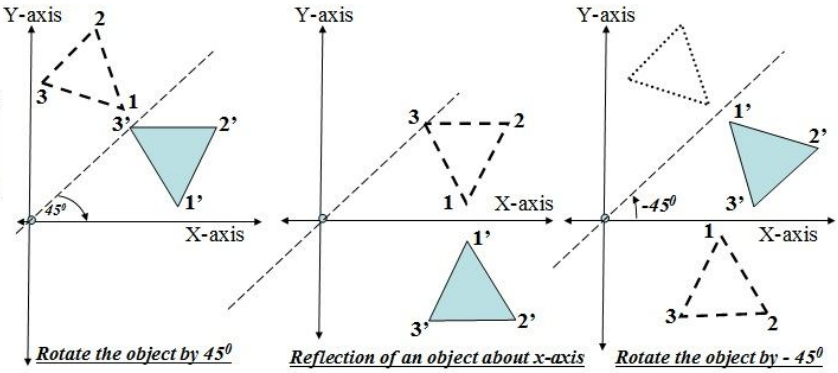
Reflection about in anticlockwise direction (θ =45)

Thus, reflection against x=y-axis (i.e. θ =45)

$$R_{x=y} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Hence

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



Composite matrix = $R_{\theta=-45}\, R_{fx}\, R_{\theta=45}$

Note: If there is more than one transformation should be performed for any task then it is called the composite transformation.

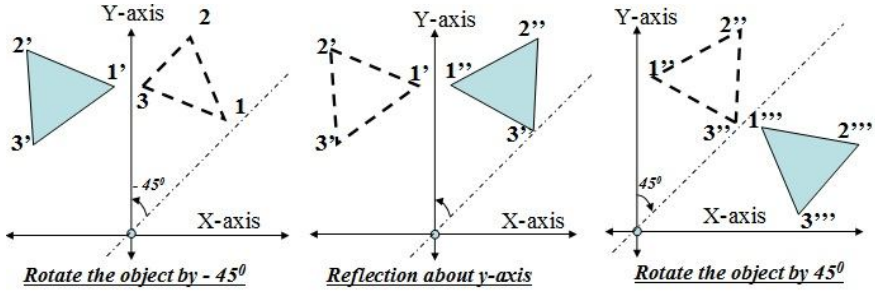
Reflection about in clockwise direction (θ = -45)

Thus, reflection against x=y-axis (i.e. θ = -45)

$$R_{x=y} = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$$

Hence

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



Composite matrix = $R_{\theta=45}\, R_{fy}\, R_{\theta=-45}$

(v) Reflection about line y = -x

Steps required:

i. Rotate about origin in clockwise direction by 45
ii. Take reflection against y-axis
iii. Rotate in anti-clockwise direction by same angle

$$R = \begin{pmatrix} \text{Cos}\,\theta & -\text{Sin}\,\theta \\ \text{Sin}\,\theta & \text{Cos}\,\theta \end{pmatrix} \;\&\; R = \begin{pmatrix} \text{Cos}\,\theta & \text{Sin}\,\theta \\ -\text{Sin}\,\theta & \text{Cos}\,\theta \end{pmatrix}$$

Thus, reflection against x=y-axis
in anti-clockwise direction (i.e. θ = 45)

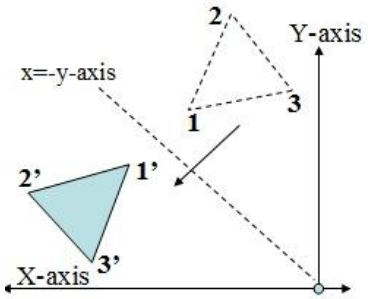
Hence

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \textit{Where} \quad R_{x=-y} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$$

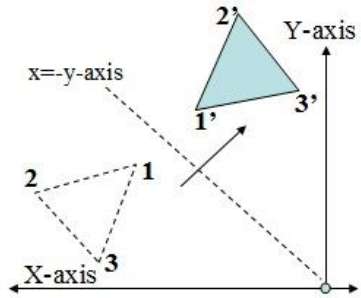
Thus, reflection against x=y-axis
in clockwise direction (i.e. θ = - 45)

Hence

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \textit{Where} \quad R_{x=-y} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



Reflection of an object about x= -y in anti-clockwise direction



Reflection of an object about x= -y in clockwise direction

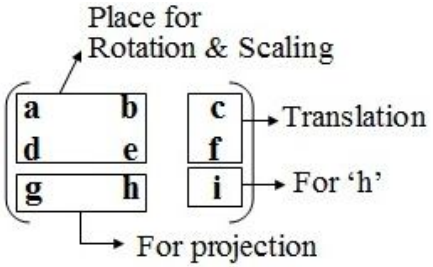
4.3 Homogenous Coordinates

The matrix representations for translation, scaling and rotation are respectively:

Translation: **P' = T + P** (*Addition*)
Scaling: **P' = S . P** (*Multiplication*)
Rotation: **P' = R . P** (*Multiplication*)

Since, the composite transformation such as ***Com* = $R_f T R_\theta T'$** include many sequence of translation, rotation etc and hence the many naturally differ addition & multiplication sequence have to perform by the graphics allocation. Hence, the applications will take more time for rendering. Thus, we need to treat all three transformations in a consistent way so they can be combined easily & compute with one mathematical operation. If points are expressed in homogenous coordinates, all geometrical transformation equations can be represented as matrix multiplications.

Here, in case of homogenous coordinates we add a third coordinate '*h*' to a point (x, y) so that each point is represented by (*h*x, *h*y, *h*). The '*h*' is normally set to 1. If the value of 'h' is more the one value then all the co-ordinate values are scaled by this value.



Coordinates of a point are represented as three element column vectors, transformation operations are written as 3 x 3 matrices.

For translation

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

With T(t_x , t_y) as translation matrix, inverse of this translation matrix is obtained by representing t_x, t_y with $-t_x$, $-t_y$.

For rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Cos}\theta & -\text{Sin}\,\theta & 0 \\ \text{Sin}\theta & \text{Cos}\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \textbf{(a)}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \text{Cos}\theta & \text{Sin}\,\theta & 0 \\ -\text{Sin}\theta & \text{Cos}\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \textbf{(b)}$$

Here, figure-a shows the Counter Clockwise (CCW) rotation & figure-b shows the Clockwise (CW) rotation.

For scaling

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} S_{hx} & 0 & 0 \\ 0 & S_{hy} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Inverse scaling matrix is obtained with $1 / s_{hx}$ and $1 / s_{hx}$.

For Reflection

- Reflection about x-axis

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

- Reflection about y-axis

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

- Reflection about y=x-axis

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

- Reflection about y=-x-axis

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

- Reflection about any line y=mx+c

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{-(m^2-1)}{(m^2+1)} & \frac{2m}{(m^2+1)} & \frac{2mc}{(m^2+1)} \\ \frac{2m}{(m^2+1)} & \frac{(m^2-1)}{(m^2+1)} & \frac{2c}{(m^2+1)} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

4.4 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. Forming products of transformation matrices is often referred to as a **concatenation,** or **composition,** of matrices. For column matrix representation of coordinate positions we form composite transformation by multiplying matrices in order from right to left.

- *What do you mean by composite transformation? What is its significance in computer graphics?*

Exercise-I

- *Rotate the triangle (5, 5), (7, 3), (3, 3) about fixed point (5, 4) in counter clockwise (CCW) by 90 degree.*

Solution

Here, the required steps are:

- Translate the fixed point to origin.
- Rotate about the origin by specified angle θ.
- Reverse the translation as performed earlier.

Thus, the composite matrix is given by

Com = $T_{(xf, yf)}\ R_\theta\ T_{(-xf, -yf)}$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} Cos90 & -Sin90 & 0 \\ Sin90 & Cos90 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -5 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 5 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 4 \\ 1 & 0 & -5 \\ 0 & 0 & 1 \end{pmatrix}$$

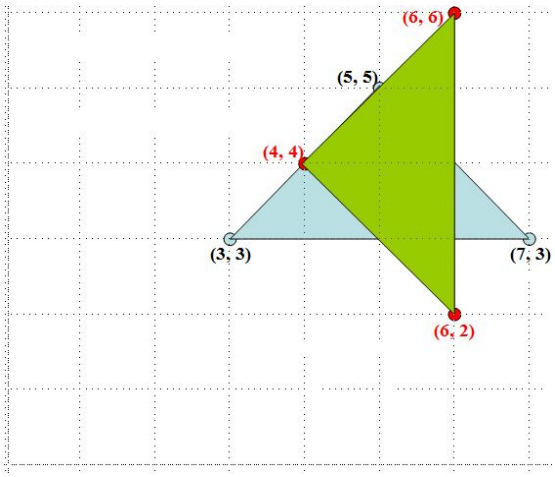
$$= \begin{pmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

Now, the required co-ordinate can be calculated as:

P' = Com x P

$$= \begin{pmatrix} 0 & -1 & 9 \\ 1 & 0 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 & 7 & 3 \\ 5 & 3 & 3 \\ 1 & 1 & 1 \end{pmatrix}$$

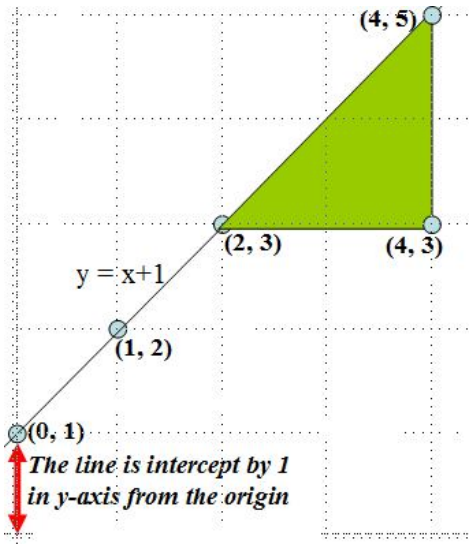
$$= \begin{pmatrix} 4 & 6 & 6 \\ 4 & 6 & 2 \\ 1 & 1 & 1 \end{pmatrix}$$



Hence, the new required coordinate points are (4, 4), (6, 6) & (6, 2).

Exercise-II

- *Reflect an object (2, 3), (4, 3), (4, 5) about line y = x +1.*



Solution

Here,

The given line is $y = x + 1$.

Thus,

When x = 0, y=1

When x = 1, y=2

When x = 2, y=3

Also,

The slope of the line (m) = 1

Thus, the rotation angle (θ) = $Tan^{-1}(m) = Tan^{-1}(1) = 45^0$

Here, the required steps are:

- Translate the line to origin by decreasing the y-intercept with one.
- Rotate the line by angle 45^0 in clockwise direction so that the given line must overlap x-axis.
- Reflect the object about the x-axis.
- Reverse rotate the line by angle -45^0 in counter-clockwise direction.
- Reverse translate the line to original position by adding the y-intercept with one.

Thus, the composite matrix is given by: Com = $\boldsymbol{T' R_\theta' R_{fx} R_\theta T}$

Thus, composite matrix is given by

$$= \underset{\text{Addition y-intercept}}{\begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}} \underset{\text{CCW Rotation}}{\begin{pmatrix}\text{Cos45}&-\text{Sin45}&0\\\text{Sin45}&\text{Cos45}&0\\0&0&1\end{pmatrix}} \underset{\text{Reflection about x-axis}}{\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}} \underset{\text{CW Rotation}}{\begin{pmatrix}\text{Cos45}&\text{Sin45}&0\\-\text{Sin45}&\text{Cos45}&0\\0&0&1\end{pmatrix}} \underset{\text{Reduce y-intercept}}{\begin{pmatrix}1&0&0\\0&1&-1\\0&0&1\end{pmatrix}}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&0\\-1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&1&-1\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\-1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&-1/\sqrt{2}\\1/\sqrt{2}&-1/\sqrt{2}&1/\sqrt{2}\\0&0&1\end{pmatrix}$$

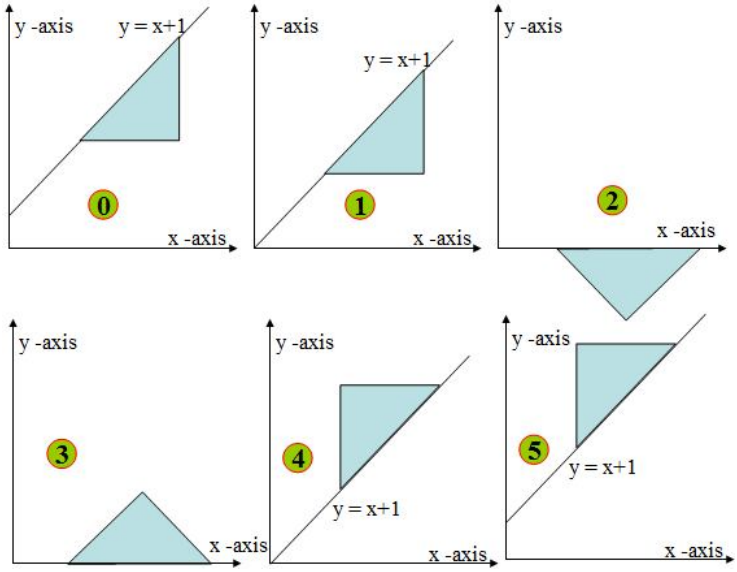
$$= \begin{pmatrix}1&0&0\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}0&1&-1\\1&0&0\\0&0&1\end{pmatrix} = \begin{pmatrix}0&1&-1\\1&0&1\\0&0&1\end{pmatrix}$$

Also, the composite matrix can be calculated as:

$$= \begin{pmatrix}\frac{-(m^2-1)}{(m^2+1)} & \frac{2m}{(m^2+1)} & \frac{2mc}{(m^2+1)}\\ \frac{2m}{(m^2+1)} & \frac{(m^2-1)}{(m^2+1)} & \frac{2c}{(m^2+1)}\\ 0 & 0 & 1\end{pmatrix}$$

Where, m =1, c=1

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



Now, the required co-ordinate can be calculated as:

P' = Com x P

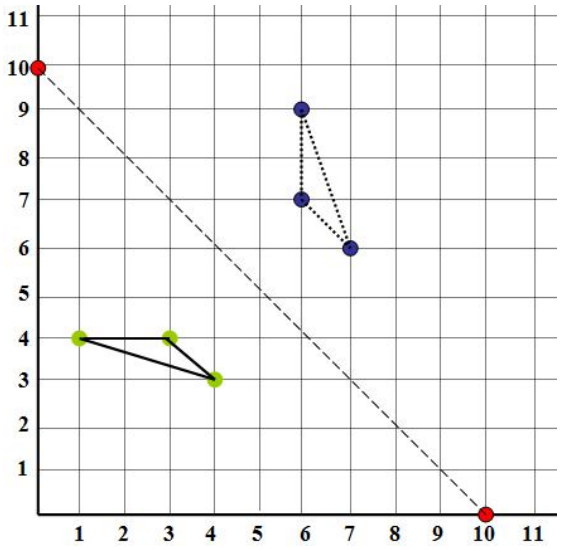
$$= \begin{pmatrix}0&1&-1\\1&0&1\\0&0&1\end{pmatrix}\begin{pmatrix}2&4&4\\3&3&5\\1&1&1\end{pmatrix}$$

$$= \begin{pmatrix}2&2&4\\3&5&5\\1&1&1\end{pmatrix}$$

Hence, the final coordinates are (2, 3), (2, 5) & (4, 5).

Exercise-III (Imp)

- *A mirror is placed such that it passes through (0, 10), (10, 0). Fin the mirror image of an object (6, 7), (7, 6), (6, 9).*



Solution

Here,

The given mirror or line is passing through the points *(0, 10) & (10, 0).*

Now, the slope of the line (m) = $(y_2-y_1) / (x_2-x_1)$ = (0 - 10) / (10 - 0) = -1

Thus, the rotation angle (θ) = $Tan^{-1}(m) = Tan^{-1}(-1) = -45^0$

The composite matrix is given by:

Com = $\boldsymbol{T}_{(0,\,10)\ or\ (10,\,0)}\ \boldsymbol{R}_{\theta\ in\ CW}\ \boldsymbol{R}_{fx}\ \boldsymbol{R}_{\theta\ in\ CCW}\ \boldsymbol{T}_{(0,\,-10)\ or\ (-10,\,0)}$

Here, we can either shift the x-coordinate of mirror to origin or the y-coordinate but not both during the translation. Let us more the x-intercept to origin.

Thus, composite matrix is given by

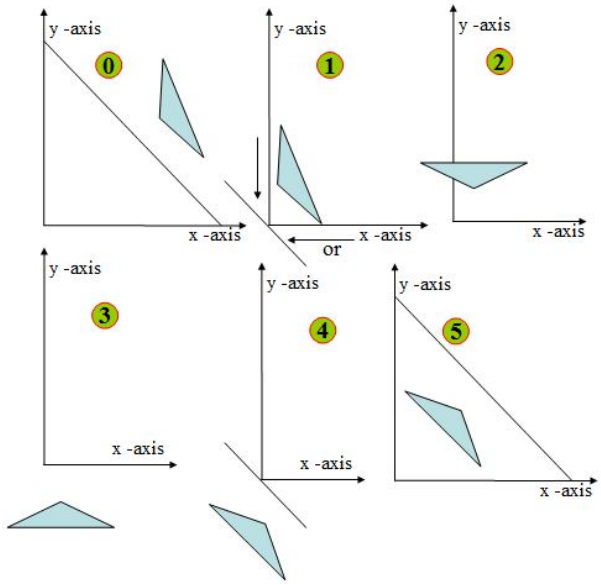
$$= \underset{\text{Addition x-intercept}}{\begin{pmatrix}1&0&10\\0&1&0\\0&0&1\end{pmatrix}} \underset{\text{CW Rotation}}{\begin{pmatrix}Cos45&Sin45&0\\-Sin45&Cos45&0\\0&0&1\end{pmatrix}} \underset{\text{Reflection about x-axis}}{\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}} \underset{\text{CCW Rotation}}{\begin{pmatrix}Cos45&-Sin45&0\\Sin45&Cos45&0\\0&0&1\end{pmatrix}} \underset{\text{Reduce x-intercept}}{\begin{pmatrix}1&0&-10\\0&1&0\\0&0&1\end{pmatrix}}$$

$$= \begin{pmatrix}1&0&10\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&0\\-1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&0\\1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&-10\\0&1&0\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&10\\0&1&1\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&0\\-1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&0&0\\0&-1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&-10/\sqrt{2}\\1/\sqrt{2}&1/\sqrt{2}&-10/\sqrt{2}\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&10\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&1/\sqrt{2}&0\\-1/\sqrt{2}&1/\sqrt{2}&0\\0&0&1\end{pmatrix}\begin{pmatrix}1/\sqrt{2}&-1/\sqrt{2}&-10/\sqrt{2}\\-1/\sqrt{2}&-1/\sqrt{2}&10/\sqrt{2}\\0&0&1\end{pmatrix}$$

$$= \begin{pmatrix}1&0&10\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}0&-1&0\\-1&0&10\\0&0&1\end{pmatrix} = \begin{pmatrix}0&-1&10\\-1&0&10\\0&0&1\end{pmatrix}$$



Now, the required co-ordinate can be calculated as:

P' = Com x P

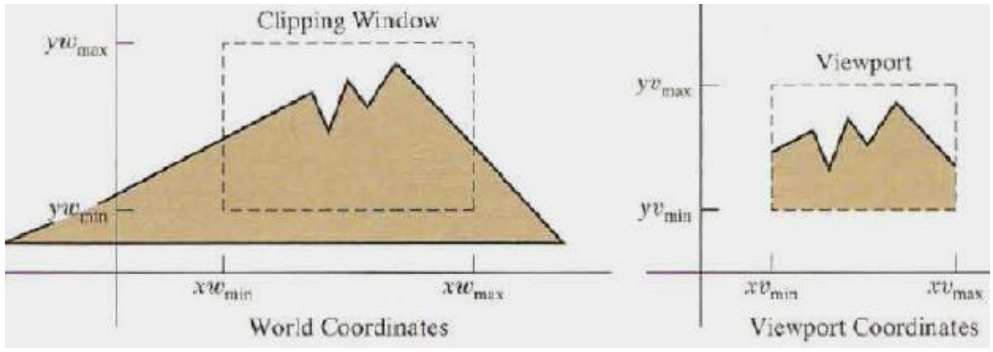
$$= \begin{pmatrix}0&-1&10\\-1&0&10\\0&0&1\end{pmatrix}\begin{pmatrix}6&7&6\\7&6&9\\1&1&1\end{pmatrix}$$

$$= \begin{pmatrix}3&4&1\\4&3&4\\1&1&1\end{pmatrix}$$

Hence, the final coordinates are (3, 4), (4, 3) & (1, 4).

4.5 Window to View port Transformation

A world-coordinate area selected for display is called a ***window***. An area on a display device to which a window is mapped is called a ***viewport***. The window defines *what* is to be viewed; the viewport defines *where* it is to be displayed.



Often, windows and viewports are rectangles in standard position (*sometime polygon shapes and circles but these takes longer process)*), with the rectangle edges parallel to the coordinate axes. In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation. Sometimes the 2D viewing transformation is simply referred to as the *window-to-viewport transformation* or the *windowing transformation.*

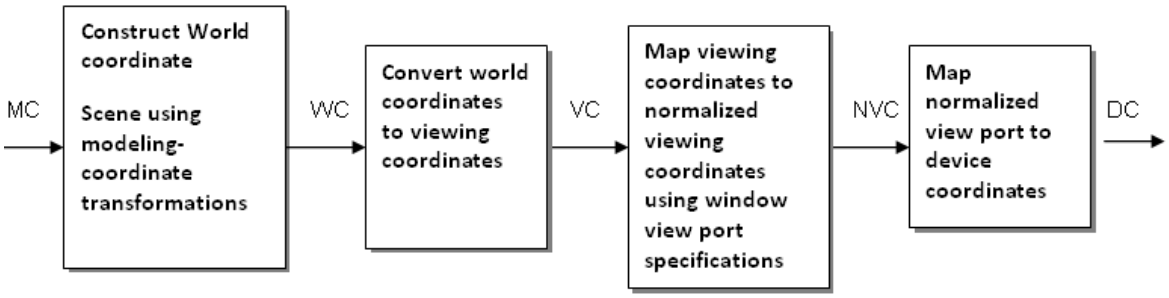
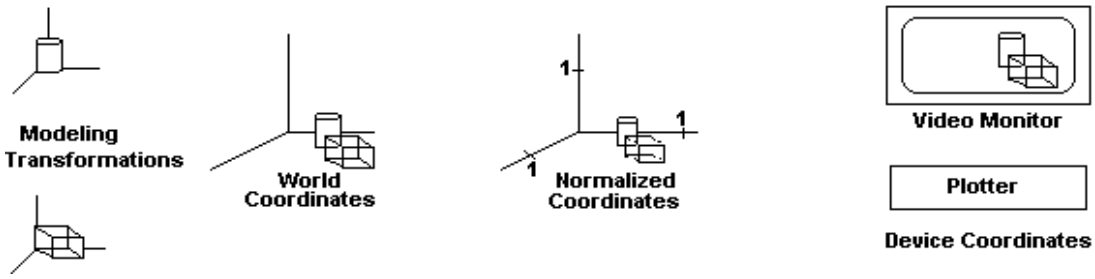


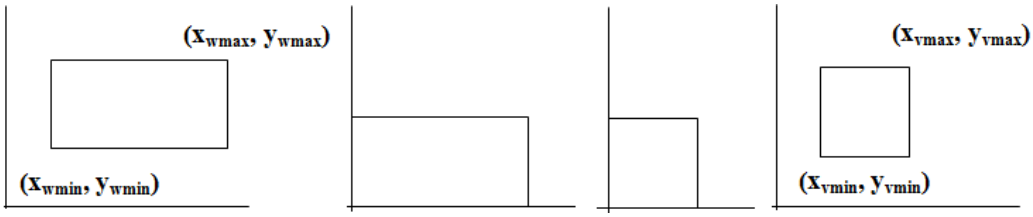
Fig. 2-D Viewing Transformation Pipeline



To transform a window to the view port we have to perform the following steps:

The overall transformation which performs these three steps called the viewing transformation. Let the window coordinates be (x_{wmin}, y_{wmin}) and (x_{wmax}, y_{wmax}) where as the view port coordinates be (x_{vmin}, y_{vmin}) and (x_{vmax}, y_{vmax}).

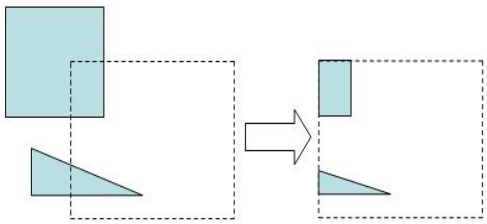
- **Step1:** The object together with its window is translated until the lower left corner of the window is at the origin
- **Step2:** The object and window are scaled until the window has the dimensions of the view port
- **Step3:** Again translate to move the view port to its correct position on the screen



4.6 Clipping in Raster World

Generally, any procedure that identifies those portions of a picture that are either *inside* or *outside* of a specified region of space is referred to as a ***clipping algorithm***, or simply ***clipping***. The region against which an object is to clip is called a clip ***window.*** In other word, the clipping defines how much we show or hide the object on the window. Here, our perception must be like as a machine to detect the visible surface. An application of clipping includes:

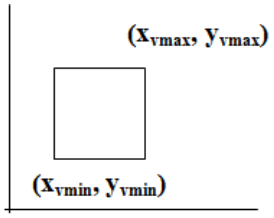
- Extracting parts of defined scene for viewing
- Identifying visible surfaces in three dimension views
- Drawing, painting operations that allow parts of picture to be selected for copying, moving, erasing or duplicating etc.



The clipping primitives are:

- Point Clipping
- Line Clipping (straight-line segments)
- Area Clipping (polygons)
- Curve Clipping
- Text Clipping

A) Point clipping



Assuming that the clip window is a rectangle in standard position, we save a point P = ***(x, y)*** for display if the following inequalities are satisfied:

- $x_{min} \leq x \leq x_{max}$
- $y_{min} \leq y \leq y_{max}$

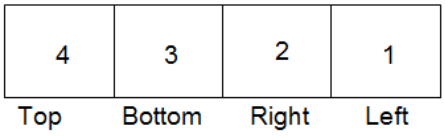
Here the edges of the clip window (x_{min}, x_{max}, y_{min}, y_{max}) can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

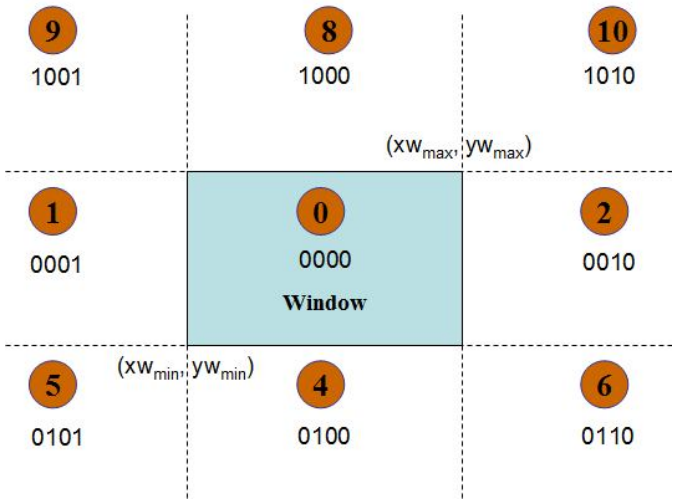
B) Line clipping

Cohen-Sutherland Line Clipping Algorithm

- Divide 2D space into 3x3 = 9-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 we deal for bits 2, 3 and 4.

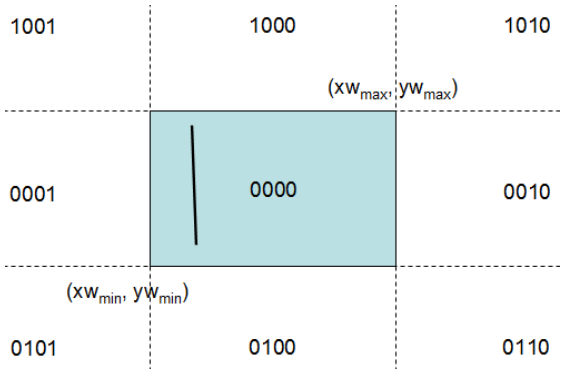




- If the left = 1 then right positional region code must be zero & similarly, if top = 1 then bottom = 0.
- To clip a line, find out which regions its two endpoints lie in.

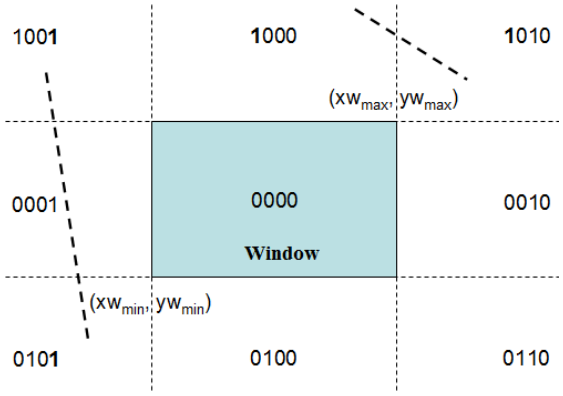
Case I

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



Case II

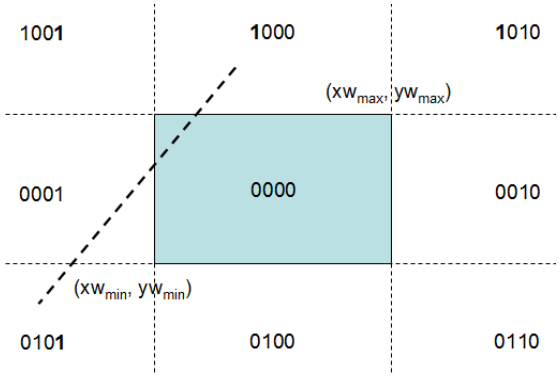
- If the two region numbers both have a 1 in the same bit position, the line is **completely out**.



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
- Here, to find the visible surface, the intersection points on the boundary of window can be determined as:
  - $y-y_1 = m\,(x-x_1)$
  - where $m = (y_2-y_1)/(x_2-x_1)$

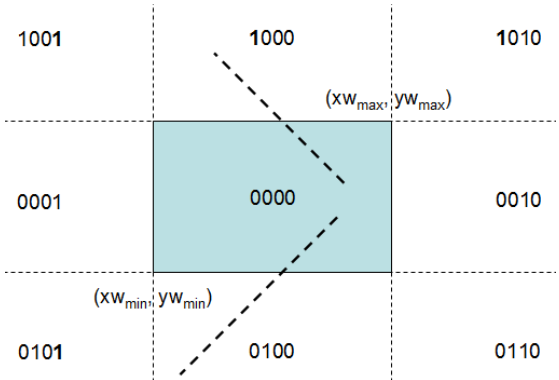


Condition-1

- For a line with end point coordinates (x1,y1) and (x2, y2) the y coordinate of the intersection point with a horizontal boundary is

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

  Where x is set to either xw_{min} or xw_{max}

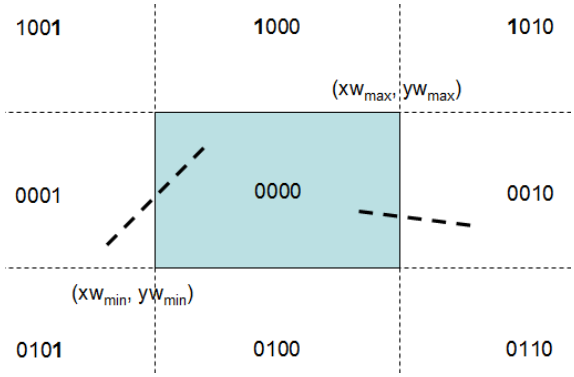


Condition-2

- 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}



Example-1

- *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)*

Solution

Finding the region code

For A (70, 90)	*For B (60, 85)*
▪ $x - xw_{min} = 70 - 50$ = Positive → 0 (Left) ▪ $xw_{max} - x = 90 - 70$ = Positive → 0 (Right) ▪ $y - yw_{min} = 90 - 80$ = Positive → 0 (Bottom) ▪ $yw_{max} - y = 120 - 90$ = Positive → 0 (Top) ***Hence the region code = 0000***	▪ $x - xw_{min} = 60 - 50$ = Positive → 0 (Left) ▪ $xw_{max} - x = 90 - 60$ = Positive → 0 (Right) ▪ $y - yw_{min} = 85 - 80$ = Positive → 0 (Bottom) ▪ $yw_{max} - y = 120 - 85$ = Positive → 0 (Top) ***Hence the region code = 0000***

Here, both the end point of the line has the region code 0000, hence both the end point are in the clip-window. This means the line is totally inside the window & thus totally visible.

Example-2

- *Use Cohen Sutherland line clipping algorithm to clip a line with end point coordinates A(70,90), B(100, 130) against a clip window with its lower left corner at (50,80) and upper right corner at (90,120)*

Solution

1. **Finding the region code**

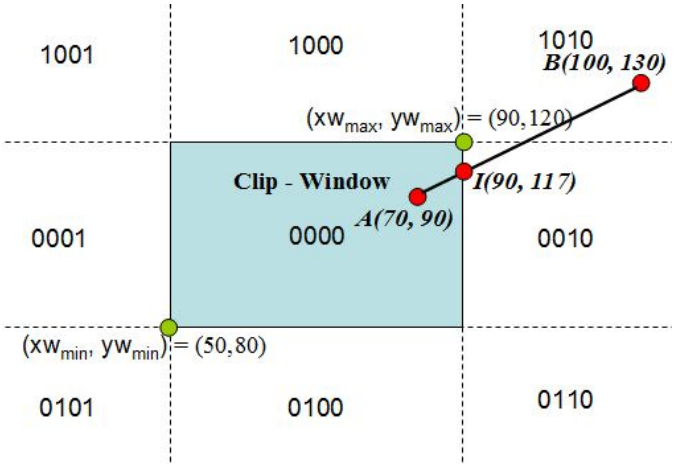
For A (70, 90)	*For B (100, 130)*
▪ $x - xw_{min} = 70 - 50$ = Positive → 0 (Left) ▪ $xw_{max} - x = 90 - 70$ = Positive → 0 (Right) ▪ $y - yw_{min} = 90 - 80$ = Positive → 0 (Bottom) ▪ $yw_{max} - y = 120 - 90$ = Positive → 0 (Top) ***Hence the region code = 0000***	▪ $x - xw_{min} = 100 - 50$ = Positive → 0 (Left) ▪ $xw_{max} - x = 90 - 100$ = Negative → 1 (Right) ▪ $y - yw_{min} = 130 - 80$ = Positive → 0 (Bottom) ▪ $yw_{max} - y = 120 - 130$ = Negative → 1 (Top) ***Hence the region code = 1010***

2. **Finding the position of line & intersection point on the clip window**

Here,
Slope of the line with end point *A(70, 90) & B(100, 130)* is given by
Slope (m) = $(y_2 - y_1)/(x_2 - x_1)$
Or m = (130 – 90) / (100-70) = 4/3
Now,
For x = 90, the value of 'y' can be calculated by using the following equation:

$$\mathbf{y = y_1 + m\,(x - x_1)}$$

Let us take the point B (100, 130) = (x_1, y_1).
Thus,
y = 130 + 4/3*(90-100) = 116.666 = 117
Hence, the intersection point is I (90, 117).



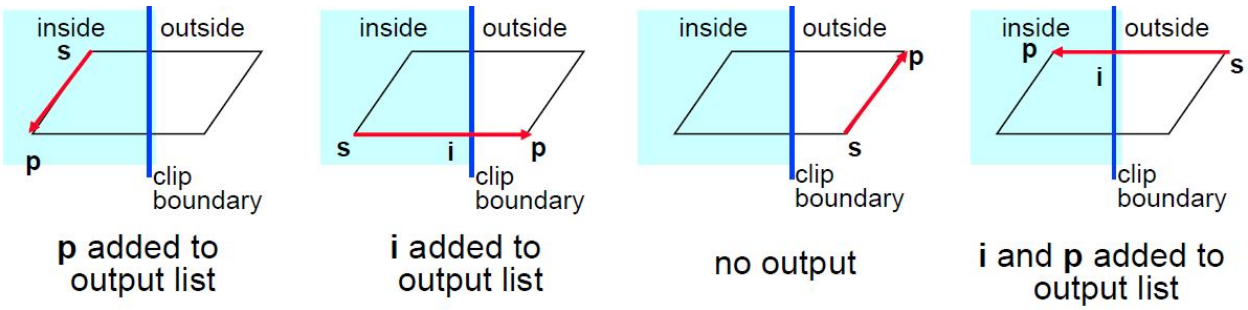
Now, we can conclude that the visible portion of the line has the end points A (70, 90) & I (90, 117).

C) Polygon Clipping: Sutherland - Hodgeman

Any procedure which identifies that portion of a picture which is either inside or outside a region is referred to as a *clipping algorithm* or *clipping.* The region against which an object is to be clipped is called clipping window. The *Sutherland–Hodgeman algorithm* is used for clipping polygons. A single polygon can actually be split into multiple polygons. The algorithm clips a polygon against all edges of the clipping region in turn. This algorithm is actually quite general — the clip region can be any convex polygon in 2D, or any convex polyhedron in 3D.

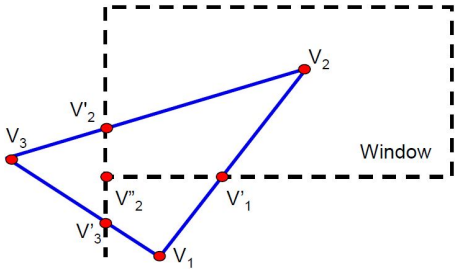
Table 1: Rules for Sutherland-Hodgman clipping.

Case	1st vertex	2nd vertex	output
1	inside	inside	2nd vertex
2	inside	outside	intersection
3	outside	outside	none
4	outside	inside	2nd and intersection

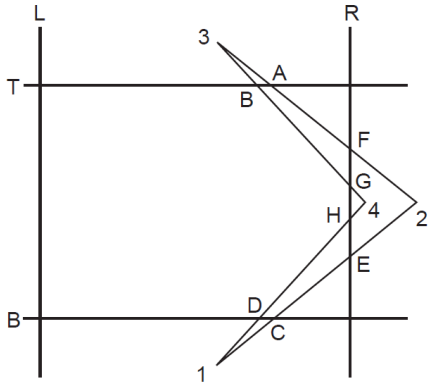


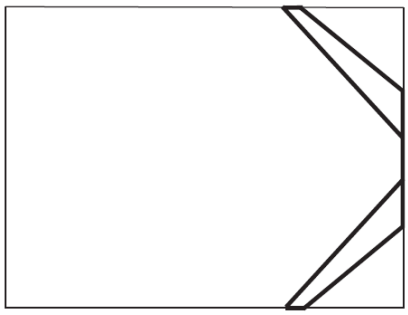
For example:

From	To	1st point	2nd point	Case	Output list
V_1	V_2	Outside	Inside	4	V'_1 and V_2
V_2	V_3	Inside	Outside	2	V'_2
V_3	V_1	Outside	Outside	3	



Assignment:





CHAPTER – 5

3D Graphics System

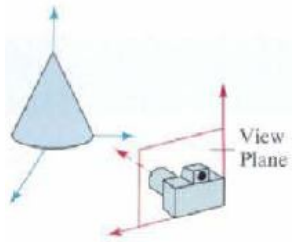
5.1 Three-Dimensional Concepts

- *What are the issue in 3D that makes it more complex than 2D?*

  → When we model and display a three-dimensional scene, there are many more considerations we must take into account besides just including coordinate values as 2D, some of them are:

  * Relatively more co-ordinate points are necessary.
  * Object boundaries can be constructed with various combinations of plane and curved surfaces.
  * Consideration of projection (dimension change with distance) and transparency.
  * Many consideration on visible surface detection and remove the hidden surfaces. Sometime we have to represent the hidden surfaces by dashes on the hardwired structure.
  * Lightning effect and intensity distribution with reference to the light source.

3D display method



* Three-dimensional viewing coordinates must be transformed onto two dimensional device coordinates to view the scene.
* To obtain a display of a three-dimensional scene that has been modeled in world coordinates, we must first set up a coordinate reference for the "camera". This coordinate reference defines the position and orientation for the plane of the camera film, which is the plane we want to use to display a view of the objects in the scene.
* Object descriptions are then transferred to the camera reference coordinates and projected onto the selected display plane.
* We can then apply lighting and surface-rendering techniques to shade the visible surfaces.

5.2 3D Geometric Transformation

Methods for geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the ***z*** coordinate. We now translate an object by specifying a three-dimensional translation vector, which determines how much the object is to be moved in each of the three coordinate directions.

2D transformations can be represented by 3 x 3 matrices using homogenous coordinates, so 3D transformations can be represented by 4 x 4 matrices, providing we use homogeneous coordinate representations of points in 2 spaces as well. Thus instead of representing a point as (x, y, z), we represent if as (x, y, z, H), where two these quadruples represent the same point it one is a non-zero multiple of the other the quadruple (0, 0, 0, 0) is not allowed. A standard representation of a point (x, y, z, H) with H not zero is given by (x/H, y/H, z/H, 1). Transforming the point to this form is called homogenizing.

A. Translation

A point is translated from position P=(x, y, z) to position P' = (x', y', z') with the matrix operation as:

P' = T.P

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

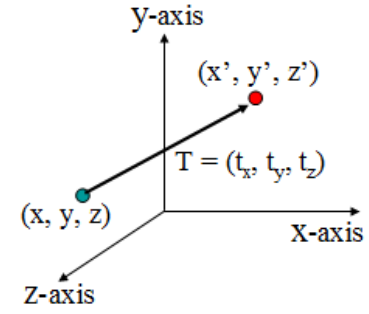


Fig. Translating a point or an object with translating vector T = (t_x, t_y, t_z)

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: $x' = x + t_x$

$y' = y + t_y$

$z' = z + t_z$

B. 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 :

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

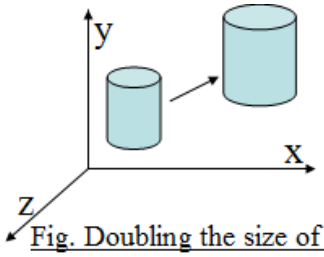


Fig. Doubling the size of an object with transformation

Scaling with respect to a selected fixed point (x_f, y_f, z_f) can be represented with:

- Translate fixed point to the origin
- Scale object relative to the coordinate origin
- Translate fixed point back to its original position

Or $T_{(x, y, z)} \cdot S_{(x, y, z)} \cdot T_{(-x, -y, z)}$

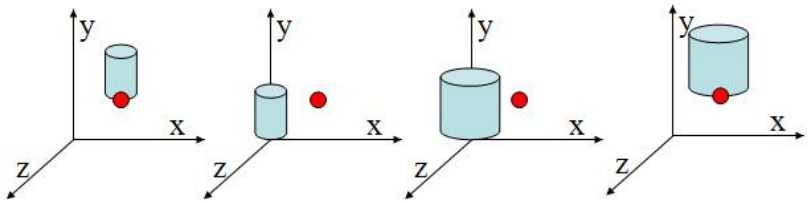


Fig. Scaling an object relative to a selected fixed point without transformation to that point

C. Reflection

A three-dimensional reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane. In general, three-dimensional reflection matrices are set up similarly to those for two dimensions. Reflections relative to a given axis are equivalent to 180 degree rotations about that axis. Reflections with respect to a plane are equivalent to 180 degree rotations in four-dimensional space.

- ***The matrix representation for the reflection of a point relative to XY-plane is given by***

$$RFz = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

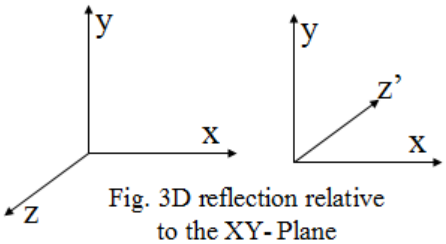


Fig. 3D reflection relative to the XY- Plane

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

- ***The matrix representation for the reflection of a point relative to XZ-plane is given by***

$$RFy = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

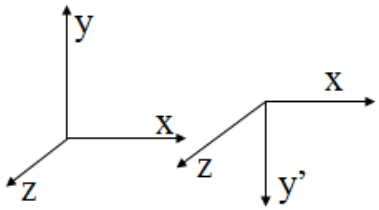


Fig. 3D reflection relative to the ZX- Plane

- ***The matrix representation for the reflection of a point relative to XZ-plane is given by***

$$RFx = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

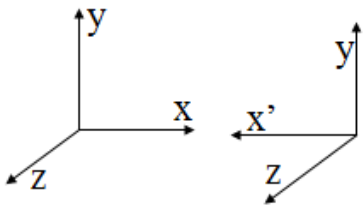


Fig. 3D reflection relative to the YZ- Plane

- *How do you reflect an arbitrary plane characterized by normal vector 'N' in 3D plane?*

D. Shearing

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

$$SHz = \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

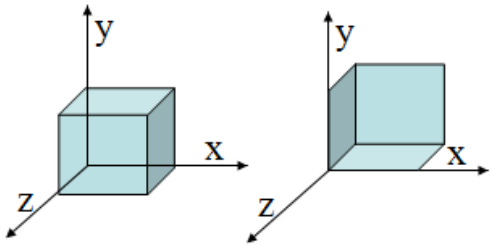


Fig. A unit cube (a) is sheared (b) by transformation matrix with a = b = 1

- The parameter 'a' & 'b' are assuming 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.

E. Rotation

To generate a rotation transformation for an object in 3D space, we require the following:

- Angle of rotation.
- Pivot point
- Direction of rotation
- Axis of rotation

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

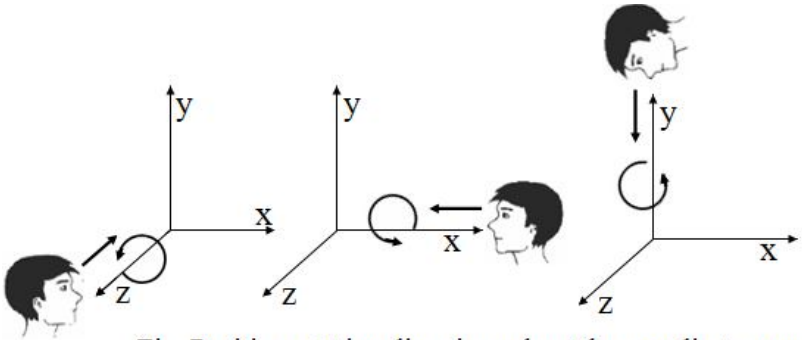
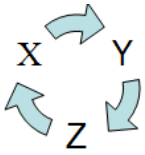


Fig. Positive rotation directions about the coordinate axes are Counterclockwise, when looking toward the origin from a positive coordinate position on each axis.

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

x → y → z →



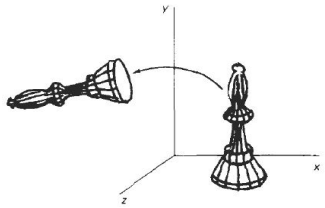
- Taking origin as the centre of rotation, when a point P(x, y, z) is rotated through an angle θ about any one of the axes to get the transformed point P'(x', y', z'), we have the following equation for each.

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

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

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

P' = Rz (θ) . P

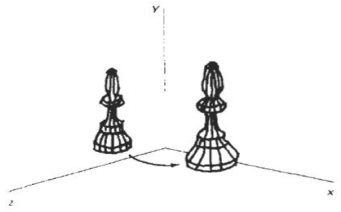


- 3D y-axis rotation equations are expressed in homogenous coordinate form as

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

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

P' = Ry (θ) . P

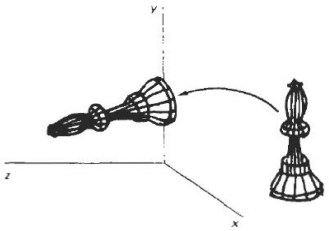


- 3D x-axis rotation equations are expressed in homogenous coordinate form as

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

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

P' = Rx (θ) . 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 its original position.

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

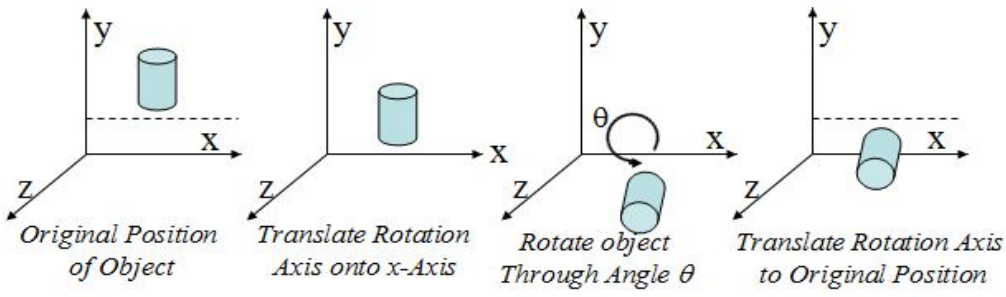


Fig. Sequence of transformations for rotating an object about an axis that is parallel to the *x* axis

Rotation about any arbitrary axis in 3D Space

Here,

Composite matrix = $T'R_y'R_x'R_{FZ}R_yR_xT$

Step-1: Translate the arbitrary axis so that it passes thru origin.

Here, the translation matrix is given by:

$$T = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step-2: Align the arbitrary axis on any major co-ordinate axis (z-axis)

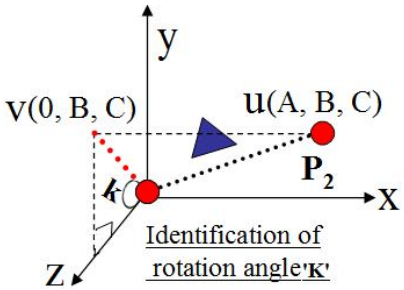
Here, at first to find the angle of rotation, project the arbitrary axis on yz-plane so that the x-coordinate will be eliminated. Here, we can find angle 'k' for rotation about x-axis by projection & 'α' for rotation about y-axis directly. After these two rotations, the arbitrary axis will align with the z-axis.

2.a: Rotation about x-axis by angle 'k' in clockwise direction so that the arbitrary axis lies on xz-plane.

Here,

Sin k = B/V &

Cos k = C/V



Now, the translation matrix is given by:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \text{Cos k} & -\text{Sin k} & 0 \\ 0 & \text{Sin k} & \text{Cos k} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & -B/V & 0 \\ 0 & B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

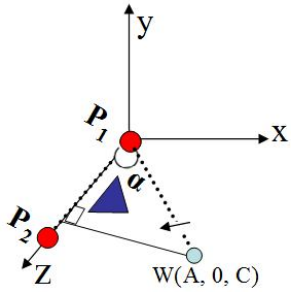
2.b: Rotation about y-axis by angle 'α' in clockwise direction so that the arbitrary axis align with z-axis.

Here,

Sinα= A/W &

Cosα= C/W

Now, the translation matrix is given by:



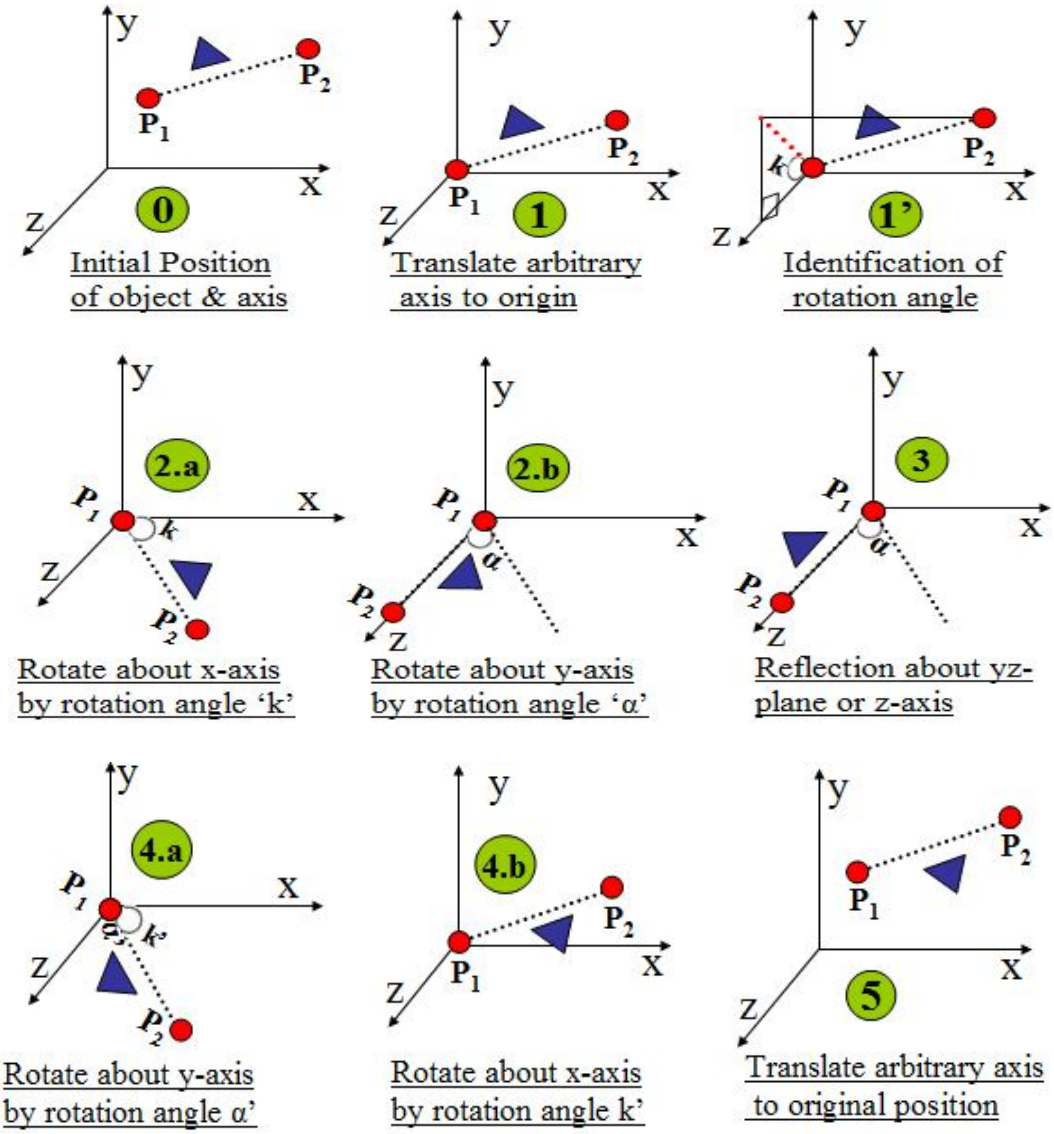
$$R_y = \begin{pmatrix} \text{Cos}\alpha & 0 & -\text{Sin}\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \text{Sin}\alpha & 0 & \text{Cos}\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} C/W & 0 & -A/W & 0 \\ 0 & 1 & 0 & 0 \\ A/W & 0 & C/W & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step-3: Reflect the object about yz-plane or z-axis

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

Step-4: Perform inverse rotation about y-axis & then x-axis

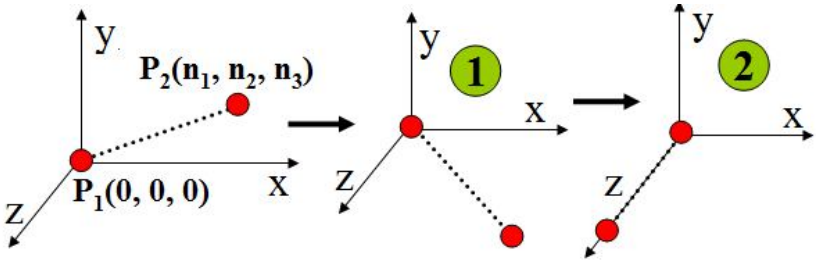
Step-5: Perform inverse translation



 # Find the alignment transformation matrix (AN) that aligns a given vector N = $n_1I + n_2J + n_3K$ to positive z-axis.

Solution

Here,
The given vector is passing through the origin with end points (n_1, n_2, n_3) in 3D space.



Thus, we have to only perform rotation about x-axis & then about y-axis to align the vector on the z-axis.

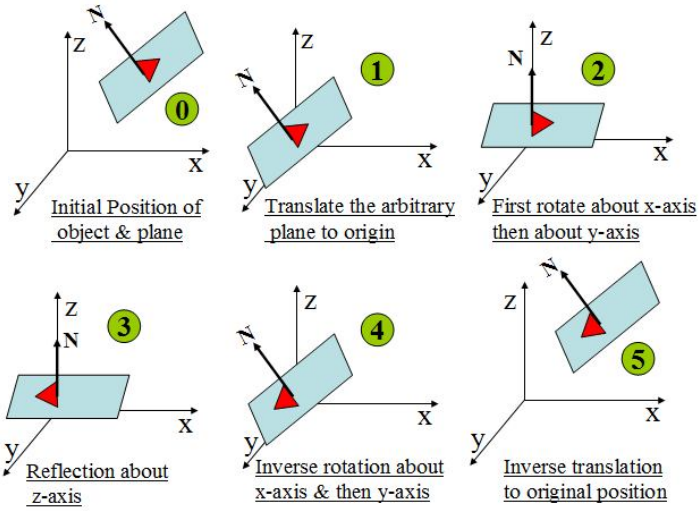
Hence, the composite matrix is given by: Com = $R_y * R_x$

Note: - Here, the angle for rotation about x-axis is identified by the given value (n_1, n_2, n_3).

Rotation about any arbitrary plane in 3D Space

The rotation about any arbitrary plane perform same operation as the rotation about any arbitrary line, the only difference is that we have to characterize the rotation by any normal vector 'N' in that plane.

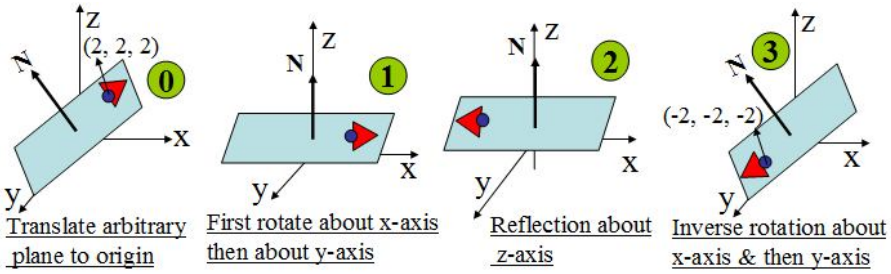
Here,
Composite matrix = $T'R_y'R_x'R_{FZ}R_yR_xT$



 # Find the mirror reflection with respect to a plane passing through point (2, 2, 2) & having normal vector N = I + J + K (Note: if the vector are represented by small letter then we have to put an arrow on their head).

Solution

Here, the composite matrix is given by: Com = $R_y' * R_x' * R_y * R_x$



Note: Here, the angle of rotation about x-axis is 45 degree as the normal passes that angle (according to given vector).

5.2 3D Object Representation

Graphics scenes can contain many different kinds of objects like trees, flowers, clouds, rocks, water etc. these cannot be describe with only one methods but obviously require large precisions such as polygon & quadratic surfaces, spline surfaces, procedural methods, volume rendering, visualization techniques etc. Representation schemes for solid objects are often divided into two broad categories:

* ***Boundary representations (B-reps):*** describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. For examples: polygon surfaces and spline patches.
* ***Space-partitioning representations***: are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes). For example ***octree*** representation.

The visual realism in 3D object can be maintained by maintaining the transparency, projection, lighting & shading effect on each portion. The representation of 3D object include following three stages:

1. Represent the objects with multiple polygons i.e. wired-frame representation.
2. Fill all the polygons either with flat filling or smooth filling.
3. Give the lightning or shading effect and the coloring effect.

5.3 Polygon Surfaces

A set of polygon surfaces are used to represent object boundary that enclose the object interior in 3D graphics. This simplifies and speeds up the surface rendering and display of objects, since all surfaces are described with linear equations of plane: Ax + By + Cz + D = 0. For this reason, polygon descriptions are often referred to as "standard graphics objects." The wireframe representations are common in design & solid modeling application because they can be displayed quickly to of a give a general indication of the surface structure.

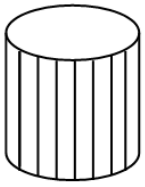


Fig. Wireframe model of a cylinder with back (hidden) lines removed

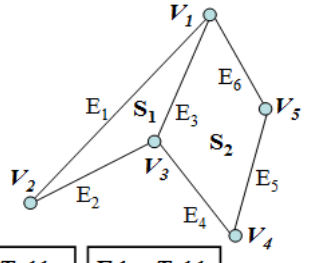
Polygon Tables

To specify a polygon surface, a set of vertex coordinates and associated attribute parameters are placed into tables & that are used in the subsequent processing, display, error checking and manipulation of the objects in a scene. Polygon data tables can be organized into two groups:

a) Geometric tables

Geometric data tables contain vertex coordinates and parameters to identify the spatial orientation of the polygon surfaces. A convenient organization for storing geometric data is to create three lists: a vertex table, an edge table, and a polygon table.

* Coordinate values for each vertex in the object are stored in the ***vertex table***.
* The ***edge table*** contains pointers back into the vertex table to identify the vertices for each polygon edge.
* The ***polygon table*** contains pointers back into the edge table to identify the edges for each polygon.



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

Edge Table
E_1: V_1, V_2
E_2: V_2, V_3
E_3: V_3, V_1
E_4: V_3, V_4
E_5: V_4, V_5
E_6: V_5, V_1

Polygon Table
S_1: E_1, E_2, E_2
S_2: E_3, E_4, E_5, E_6

b) Attribute tables

Attribute information for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics. The above three table also include the polygon attribute according to their pointer information.

- *What is boundary representation? How are polygon tables useful for modeling 3D surfaces?*

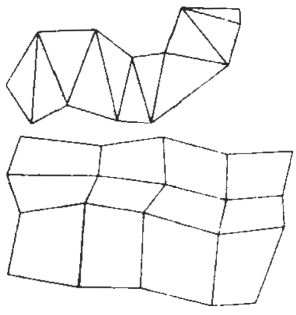
Plane Equations

The equation for a plane surface can be expressed as: Ax + By + Cz + D =0, Where, (x, y, z) is any point on the plane- coefficients ABCD are constants describing the spatial properties of the plane.

* If Ax + By + Cz + D < 0 then the point (x, y, z) is inside the surface
* If Ax + By + Cz + D > 0 then the point (x, y, z) is outside the surface

Polygon Meshes

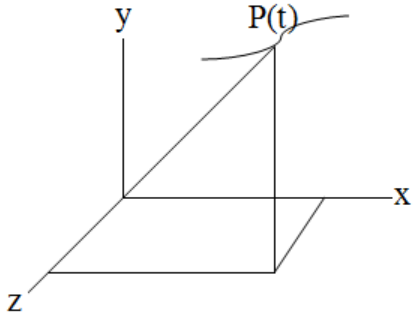
A polygon mesh is a collection of edges, vertices and polygons connected such that each edge is shared by at most two polygons & hence bounded the planer surface. One type of polygon mesh is the ***triangle strip*** that produces n – 2 connected triangles, given the coordinates for n vertices. Another similar functions the ***quadrilateral mesh*** that generates a mesh of (n-1).(m-1) quadrilaterals, given the coordinates for an n by m array of vertices.



5.4 Parametric Cubic Curve

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

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



Expanding equation (i) yield

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

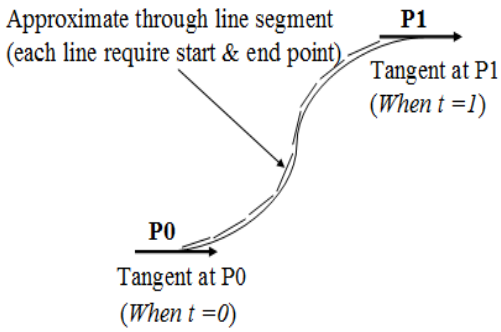
This equation is separated into three components of P (t)

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

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

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

- To be able to solve (iii) the twelve unknown coefficients a_{ij} (algebraic coefficients) must be specified
- From the known end point coordinates of each segment, six of the twelve needed equations are obtained.
- The other six are found by using tangent vectors at the two ends of each segment
- The direction of the tangent vectors establishes the slopes (direction cosines) of the curve at the end points
- This procedure for defining a cubic curve using ***end points*** and ***tangent vector*** is one form of **Hermite interpolation**
- Each cubic curve segment is parameterized from 0 to 1 so that known end points correspond to the limit values of the parametric variable *t*, that is P(0) and P(1)
- Substituting *t* = 0 and *t* = 1 the relationship between two end point vectors and the algebraic coefficients are found



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

- 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*) = a1 $\qquad$ **P' (*1*) = $3a_3 + 2a_2 + a_1$** ---------- (V)

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

a_0= P (*0*) $\qquad$ **a_1= P' (0)**

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

(Note: - The value of a2 & a3 can be determined by solving the equation IV & V)

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

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

Spline curve & Spline surface

* In computer graphics, continuous curve that are formed with polynomial pieces with certain boundary conditions are called ***spline curve*** or simply ***spline***. A ***spline surface*** can be described with two sets of orthogonal spline curves.
* In drafting terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points. Several small weights are distributed along the length of the strip to hold the position of ***spline curve*** as requirement.

* We can mathematically describe such a curve with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections.
* Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or image. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

Control points

We specify a spline curve by giving a set of coordinate positions, called ***control points***, which indicates the general shape of the curve These, control points are then fitted with piecewise continuous parametric polynomial functions in one of two ways.

- ***Interpolation curve***: The polynomial sections are fitted by passing the curve through each control points. Interpolation curves are commonly used to digitize drawings or to specify animation paths.
- ***Approximation curve***: The polynomials are fitted to the general control-point path without necessarily passing through any control point. Approximation curves are primarily used as design tools to structure object surfaces.

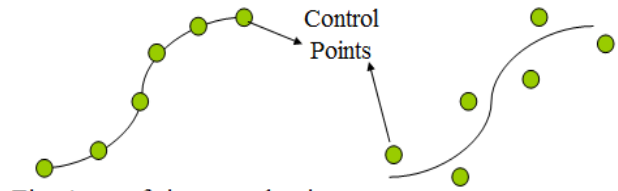


Fig. A set of six control points ***interpolated*** with piecewise continuous polynomial sections

Fig. A set of six control points ***approximated*** with piecewise continuous polynomial sections

A spline curve is defined, modified, and manipulated with operations on the control points. By interactively selecting spatial positions for the control points, a designer can set up an initial curve. After the polynomial fit is displayed for a given set of control points, the designer can then reposition some or all of the control points to restructure the shape of the curve. In addition, the curve can be translated, rotated, or scaled with transformations applied to the control points. CAD packages can also insert extra control points to aid a designer in adjusting the curve shapes.

5.5 Bezier Curve and Surfaces

Bezier splines are highly useful, easy to implement and convenient for curve and surface design so are widely available in various CAD systems, graphics packages, drawing and painting packages.

Bezier curve

In general, a 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 the Bezier polynomial. As with the interpolation splines, a Bezier curve can be specified with boundary conditions, with a characterizing matrix, or with ***blending functions***. The Bezier curve has two important properties:

1. It always passes through the first and last control points.
2. It lies within the convex hull (convex polynomial boundary) of the control points. This follows from the properties of Bezier blending function: they are positive and their sum is always 1, i.e.

$$\sum_{k=0}^{n} \mathbf{z}_k \, BEZ_{k,n}(u) = 1$$

Suppose we are given n + 1 control-point positions: $\mathbf{p}_k = (x_k, y_k, z_k)$, with k varying from 0 to n. These coordinate points can be blended to produce the following position vector P (u), which describes the path of an approximating Bezier polynomial function between p_0 and p_n.

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k \, BEZ_{k,n}(u), \qquad 0 \le u \le 1$$

This, vector equation represents a set of three parametric equations for the individual curve co-ordinates:

$$\mathbf{x}(u) = \sum_{k=0}^{n} \mathbf{x}_k \, BEZ_{k,n}(u)$$

$$\mathbf{y}(u) = \sum_{k=0}^{n} \mathbf{y}_k \, BEZ_{k,n}(u)$$

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

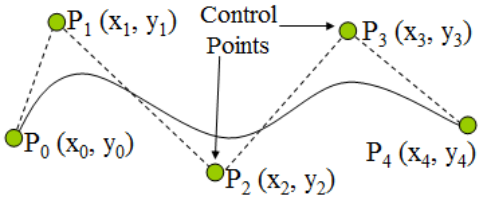
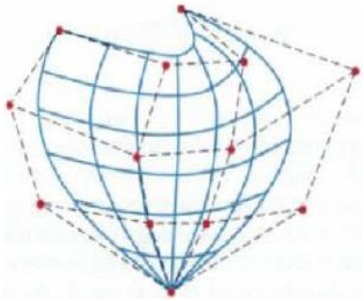


Fig. Bezier curve with 4-control points in xy-plane (z=0)



The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials:

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

Where the ***C (n, k)*** are the binomial coefficients: $C(n, k) = \dfrac{n!}{k!\,(n-k)!}$

Bezier Non-planar Surfaces

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

$$\mathbf{P}(u, v) = \sum_{j=0}^{m} \sum_{k=0}^{n} \mathbf{P}_{j,k} \, BEZ_{j,m}(v) \, BEZ_{k,n}(u)$$

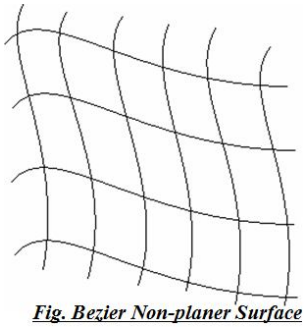
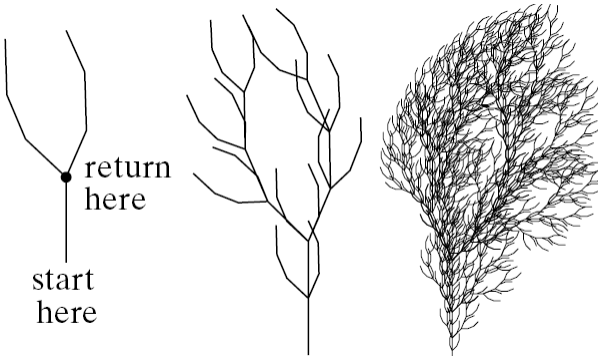


Fig. Bezier Non-planer Surface

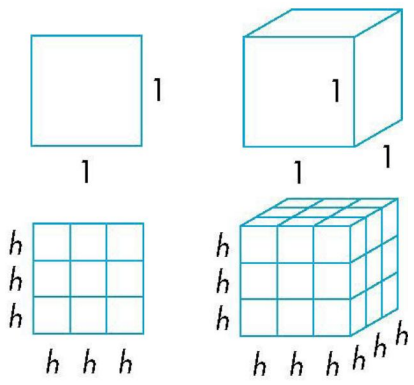
With $\mathbf{p}_{j,k}$ specifying the location of the ***(m* + 1)** by ***(n* + 1)** control points

5.6 Fractal Geometry Method

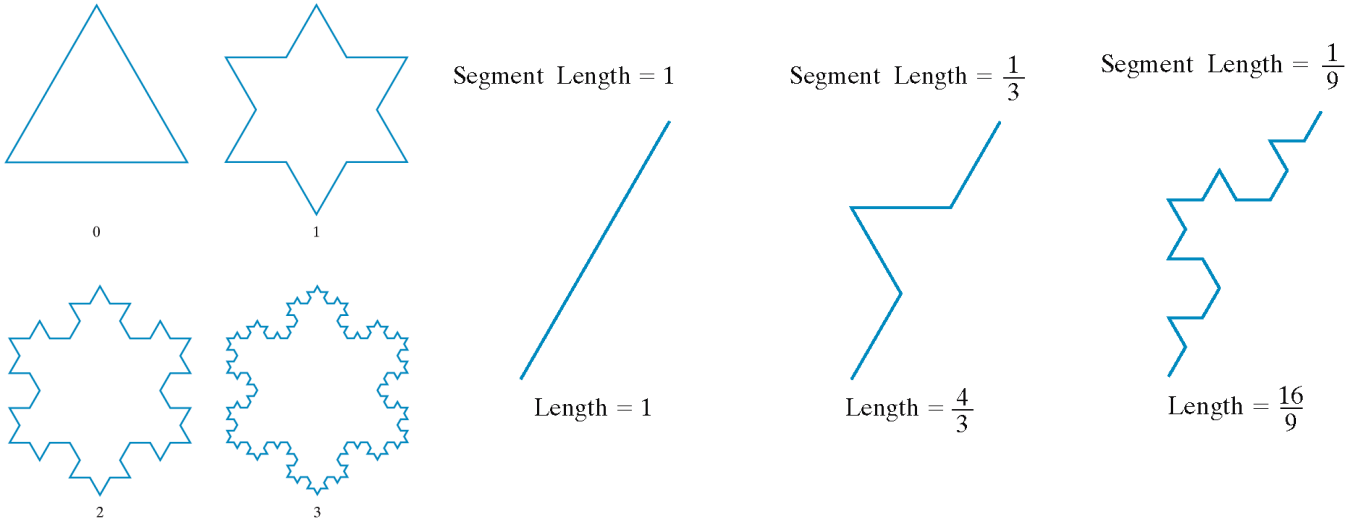
All the previous object representations use Euclidean-geometry methods in which object shapes were described with equations and hence these objects have smooth surfaces and regular shapes. But natural objects, such as mountains and clouds, have irregular or fragmented features, and these are described with fractal-geometry methods. A *fractal* is a mathematical set that typically displays self-similar patterns or recursive operations, which means it is "the same from near as from far".



Fractional object describes and explains the features of natural phenomena with procedures or function in various dimensions that theoretically repeat an infinite number of times but finite number of steps for graphics display. For fractional-Generation procedure, If $P_0 = (x_0, y_0, z_0)$ is a selected initial point, each iteration of a transformation function F generates successive levels of detail with the calculations are: $P_1 = F(P_0)$, $P_2 = F(P_2)$, …



We can describe the amount of variation in the object detail with a number called the ***fractal dimension.*** Unlike the Euclidean dimension, this number is not necessarily an integer. The fractal dimension of an object is sometimes referred to as the ***fractional dimension,*** which is the basis for the name "fractal".



5.7 3D Viewing Transformation

In two-dimensional graphics applications, viewing operations transfer positions from the world-coordinate plane to pixel positions in the plane of the output device. Using the rectangular boundaries for the world-coordinate window and the device viewport, a two-dimensional package maps the world scene to device coordinates and clips the scene against the four boundaries of the viewport. For three-dimensional graphics applications, the situation is a bit more involved, since we now have more choices as to how views are to be generated. First of all, we can view an object from any spatial position: from the front, from above, or from the back. Or we could generate a view of what we would see if we were standing in the middle of a group of objects or inside a single object, such as a building. Additionally, three-dimensional descriptions of objects must be projected onto the flat viewing surface of the output device. And the clipping boundaries now enclose a volume of space, whose shape depends on the type of projection we select.

Viewing Pipeline

The following figure shows general processing steps for modeling and converting a world-coordinate description of a scene to device coordinates. Once the scene has been modeled, world-coordinate positions are converted to viewing coordinates. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of in analogy with the camera film plane. Next, projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device. Objects outside the specified viewing limits are clipped from further consideration, and the remaining objects are processed through visible-surface identification and surface-rendering procedures to produce the display within the device viewport.

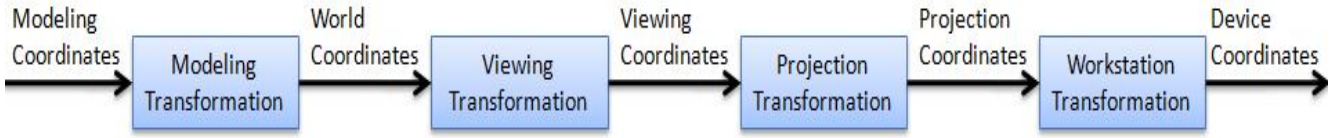


Figure: General three-dimensional transformation pipeline, from modeling coordinates to final device coordinates.

5.8 Projection Methods

Transform points in coordinate system of dimension 'n' into points in a coordinate system of dimension less than 'n'. Projection of 3D object is defined by straight projection rays (projectors) emanating from center of projection, passing thru each point of object and intersecting a projection plane to form projection. These are planar geometric projection as the projection is onto a plane rather than some curved surface and uses straight rather than curved projectors.

- Once the world coordinate descriptions of the objects in a scene are converted to viewing coordinates, we can project the 3-D objects onto the 2-D view plane
- The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.
- The two types of projections are: Parallel projection and Perspective projection

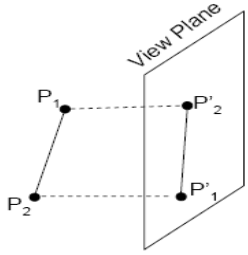


Fig. Parallel Projection of an object to the view plane

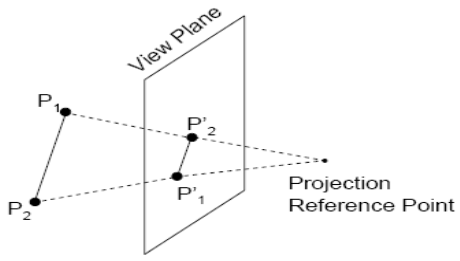


Fig. Perspective Projection of an object to the view plane

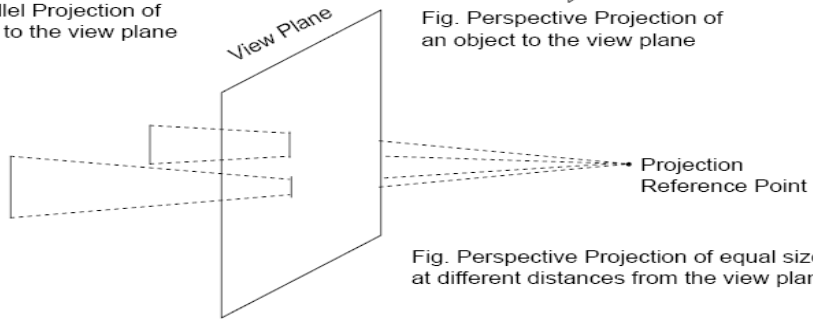
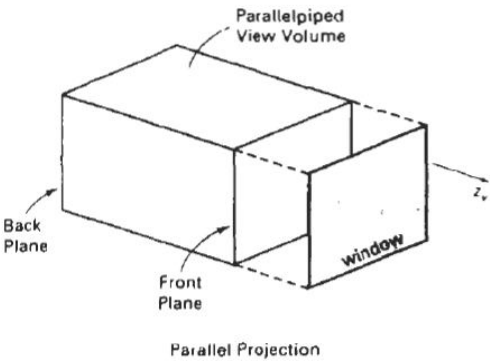


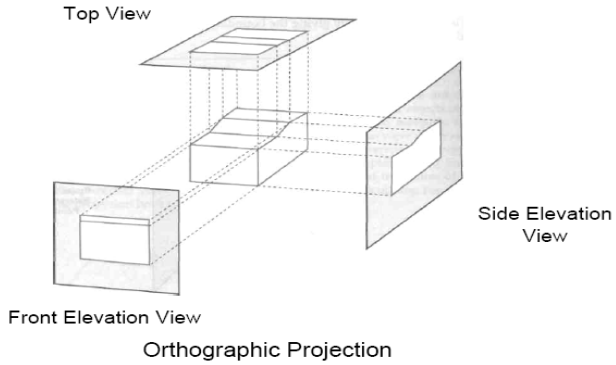
Fig. Perspective Projection of equal sized objects at different distances from the view plane

a) Parallel projection

- The projection is parallel, if the distance is infinite or there is no vanishing point or no converging point of incoming rays from the object.
- The image viewed by the parallel projection is not seemed realistic as they count the exact dimension of the entire surfaces (for realistic view the backend surface must be smaller than that of the front end.



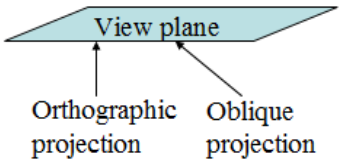
Parallel Projection



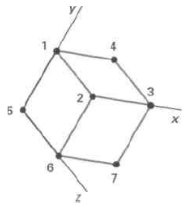
Orthographic Projection

- Coordinate positions are transformed to the view plane along parallel lines.
- Preserves relative proportions of objects so that accurate views of various sides of an object are obtained but doesn't give realistic representation of the 3D object.
- Can be used for exact measurements so parallel lines remain parallel.

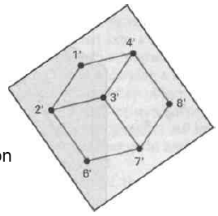
I. Orthographic parallel projection



* When projection is perpendicular to view the plane we have orthographic parallel projection.
* It is used to produce the front, side and top views of an object. Front, side and rear orthographic projections of an object are called ***elevations.***
* Top orthogonal projection is called a ***plane view***.
* This is mainly used in Engineering & architectural drawing.
* Views that display more than one face of an object are called ***axonometric orthographic projections***. Most commonly used axonometric projection is the ***isometric projection***.



Isometric Projection



* *Transformation equations:* If view plane is placed at position z_{vp} along z_v axis then any point (x, y, z) is transformed to projection as: $x_p = x$, $y_p = y$ and z value is preserved for depth information needed (visible surface detection).

II. Oblique parallel projection

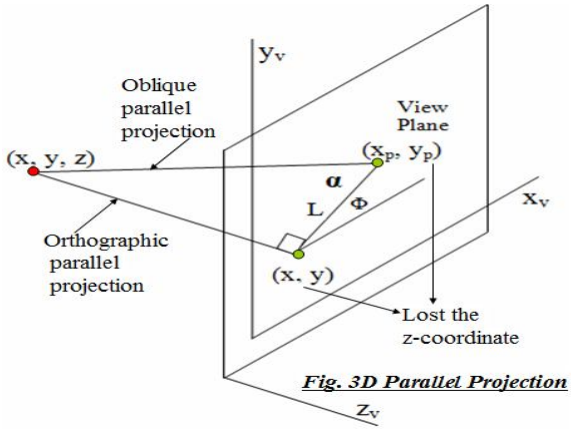


Fig. 3D Parallel Projection

* Obtained by projecting points along parallel lines that are not perpendicular to projection plane.
* Often specified with two angles Φ and α
* Point (x, y, z) is projected to position (x_p, y_p) on the view plane.
* Orthographic projection coordinated on the plane are (x, y).
* Oblique projection line from (x, y, z) to (x_p, y_p) makes an angle with then line on the projection plane that joins (x_p, y_p) and (x , y).
* This line of length L is at an angle Φ with the horizontal direction in the projection plane.
* Expressing projection coordinates in terms of x, y, L and Φ as

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

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

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

Thus,

$$L = z/\tan\alpha = zL_1,$$ where L_1 is the inverse of tanα

So the oblique projection equations are:

$$x_p = x + z(L_1\cos\Phi)$$
$$y_p = y + z(L_1\sin\Phi)$$

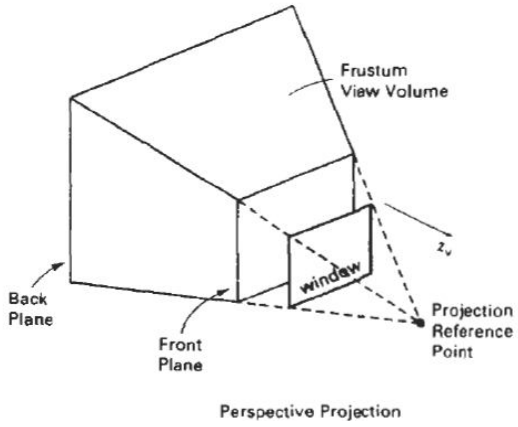
* The transformation matrix for producing any parallel projection onto the x_v y_v-plane can be written as:

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

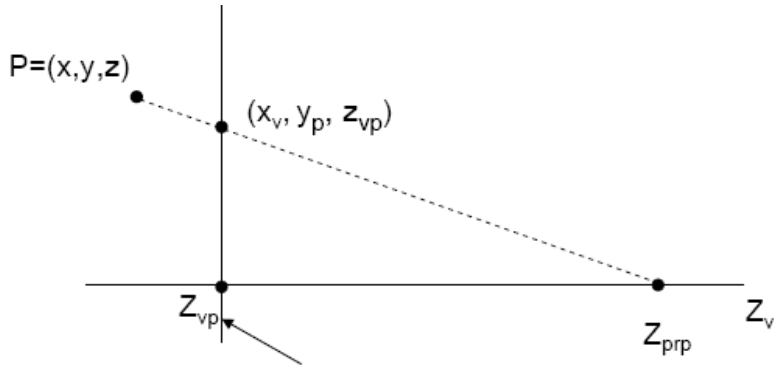
* Orthographic projection is obtained when $L1 = 0$ (occurs at projection angle α of 90 degree)
* Oblique projection is obtained with non-zero values for L_1.

b) Perspective Projection

* The projection is perspective if the distance is finite or has fixed vanishing point or the incoming rays converge at a point. Thus the viewing dimension of object is not exact i.e. seems large or small according as the movement of the view plane from the object.
* Scenes displayed using perspective projections appear more realistic, since this is the way that our eyes and a camera lens form images.
* The perspective projection perform the operation as the scaling (i.e. zoom in & out) but here the object size is only maximize to the size of real present object i.e. only size reduces not enlarge. This operation is possible here only the movement of the view plane towards or far from the object position.



Perspective Projection



View Plane

- To obtain a perspective projection of a 3-D object, we transform points along projection lines that meet at the projection reference point
- Suppose we set the projection reference point at position z_{prp} along the z_v axis, we place the view plane at z_{vp}
- We can write the equations describing coordinate positions along this perspective projection line in a parametric form as

$$x' = x - xu \qquad y' = y - yu \qquad z' = z - (z - z_{prp})\,u$$

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

- When u=0, we are at position P=(x, y, z)
- At the other end of the line, u=1 and we have the projection reference point coordinates (0,0,zprp)
  - On the view plane $z'=z_{vp}$ and we can solve the z' equation for parameter u at this position along the projection line
  - Thus, $z'=z_{vp} = z-(z-z_{prp})\,u$ $\quad u = \dfrac{z_{vp} - z}{z_{prp} - z}$
  - Substituting this value of u into the equations for x' and y', we obtain the perspective transformation equations:

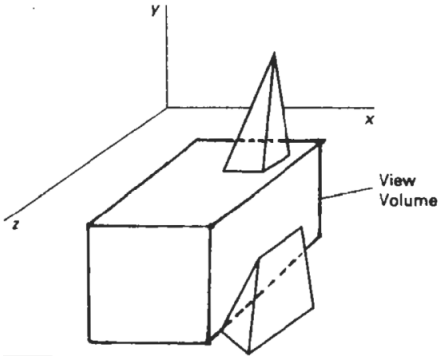
$$x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) = x\left(\frac{d_p}{z_{prp} - z}\right) \quad \text{——— (1)}$$

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

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

5.9 Clipping in 3D

Clipping in three dimensions can be accomplished using extensions of two-dimensional clipping methods. Instead of clipping against straight-line window boundaries, we now clip objects against the boundary planes of the view volume. An algorithm for three-dimensional clipping identifies and saves all surface segments within the view volume for display on the output device. All parts of objects that are outside the view volume are discarded. View-volume clipping boundaries are planes whose orientations depend on the type of projection, the projection window, and the position of the projection reference point.



To clip a polygon surface, we can clip the individual polygon edges. To clip a line segment against the view volume, we would need to test the relative position of the line using the view volume's boundary plane equations. An endpoint (x, y, *z)* of a line segment is

- Outside a boundary plane: $Ax + By + Cz + D > 0$, where *A, B,* C, and *D* are the plane parameters for that boundary.
- Inside the boundary if $Ax + By + Cz + D < 0$

Lines with both endpoints outside a boundary plane are discarded, and those with both endpoints inside all boundary planes are saved. The intersection of a line with a boundary is found using the line equations along with the plane equation. Intersection coordinates (x_1, y_1, z_1) are values that are on the line and that satisfy the plane equation $Ax_1 + By_1 + Cz_1 + D = 0$. To clip a polygon surface, we can clip the individual polygon edges.

As in two-dimensional viewing, the projection operations can take place before the view-volume clipping or after clipping. All objects within the view volume map to the interior of the specified projection window. The last step is to transform the window contents to a two-dimensional viewport, which specifies the location of the display on the output device.

CHAPTER – 6

Visible Surface Detection

6.1 Hidden Surface Removal (Visible Surface Detection) Method

A major consideration in the generation of realistic graphics displays is identifying those parts of a scene that are visible from a chosen viewing position. There are many approaches or algorithms for efficient identification of visible objects but all are differing according as memory requirement, processing time, special application etc. The various algorithms are referred to as ***visible-surface detection methods*** or sometimes ***hidden-surface elimination methods*** but for *wireframe displays*, for example, we may not want to actually eliminate the hidden surfaces, but rather to display them with dashed boundaries or in some other way to retain information about their shape.

Visible-surface detection algorithms are broadly classified as:

- **Object-space methods (OSM):** Deal with object definition
  An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. E.g. *Back-face detection method*

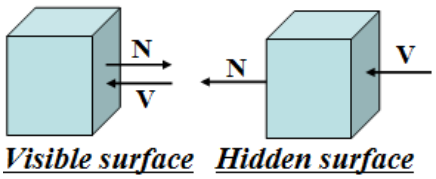
- **Image-space methods(ISM):** Deal with projected image
  In an image-space algorithm, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods. E.g. *Depth-buffer method, Scan-line method, Area-subdivision method*

- **List Priority Algorithms**
  This is a hybrid model that combines both object and image precision operations. Here, depth comparison & object splitting are done with object precision and scan conversion (which relies on ability of graphics device to overwrite pixels of previously drawn objects) is done with image precision. E.g. *Depth-Shorting method, BSP-tree method*

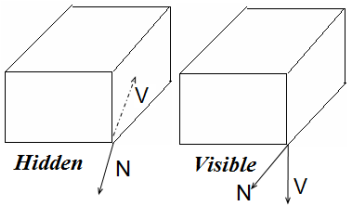
6.2 Back – Face Detection Method

A view vector V is constructed from any point on the surface to the viewpoint, the dot product of this vector and the normal N, indicates visible faces as follows:

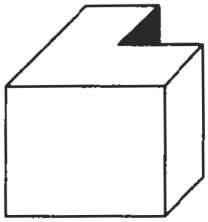
Case-I: If V.N < 0 the face is visible else face is hidden



Case-II: If V.N > 0 the face is visible else face is hidden

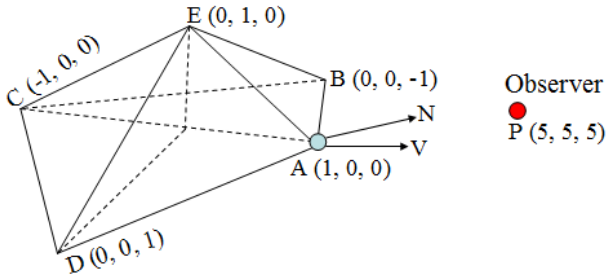


Case-III: For other objects, such as the concave polyhedron in Fig., more tests need to be carried out to determine whether there are additional faces that are totally or partly obscured by other faces.



Numerical

Find the visibility for the surface AED where an observer is at P (5, 5, 5).



Solution

Here,

AE = (0-1)i + (1-0)j + (0-0)k = -i + j

AD= (0-0)i + (1-0)j + (1-0)k = -i + k

Step-1: Normal vector N for AED

Thus, N = AE x AD = $\begin{pmatrix} i & j & k \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix}$ = i (1-0) –j (-1+0) +k (0+1)

= i + j + k

Case-I

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

V = AP = (5-1)i + (5-0)j + (5-0)k = 4i + 5j + 5k

Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

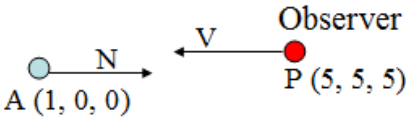
V.N = (4i + 5j + 5k).(i + j + k) = 4+5+5 = 14> 0

This shows that the surface is visible for the observer.

Case-II

Step-2: If observer at P(5, 5, 5) so we can construct the view vector V from surface to view point A(1, 0, 0) as:

V = PA = (1-5)i + (0-5)j + (0-5)k = -4i - 5j - 5k



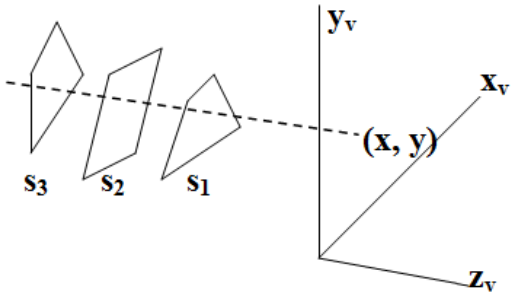
Step-3: To find the visibility of the object, we use dot product of view vector V and normal vector N as:

V.N = (-4i - 5j - 5k).(i + j + k) = -4-5-5 = -14< 0

This shows that the surface is visible for the observer.

6.3 Depth – Buffer (Z – Buffer Method)

It is a commonly used image-space approach which compares surface depths at each pixel position (i.e. one point at a time across the surface) on the projection plane. This procedure is also referred to as the z-buffer method, since object depth is usually measured from the view plane along the z axis of a viewing system. We can implementing the algorithm in normalized coordinates, z value ranges from 0 at the back clipping plane to z_{max} at the front clipping plane.



This is usually for polygon surface, because depth values can be computed very quickly and the method is easy to implement but can apply to non-planar surfaces also.

As the name implies, it require two buffer areas. A ***depth buffer*** is used to store depth values (z-values) for each (x, y) position as surfaces are processed, and the ***refresh buffer*** stores the intensity values for each position. Initially, all positions in the depth buffer are set to **0** (minimum depth), and the refresh buffer is initialized to the background intensity. Each surface listed in the polygon tables is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The

calculated depth is compared to the value previously stored in the depth buffer at that position. If the calculated depth is greater than the value stored in the depth buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same ***xy*** location in the refresh buffer.

A drawback of the depth-buffer method is that it can only find one visible surface for opaque surfaces and cannot accumulate intensity values for transparent surfaces.

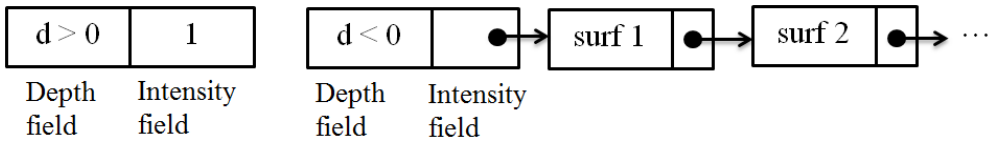
Algorithm

- Initialize the *depth buffer* and *refresh buffer* so that for all buffer position(x, y)
  depth(x, y) = 0 or (z_{min}) and refresh(x, y) = $I_{background}$
- For each position on each polygon surface compare depth values to previously stored values in the depth buffer to determine visibility
  a. Calculate the depth 'z' for each (x, y) position on the polygon
  b. If z > depth(x, y) then set
     depth = z and refresh(x, y) = $I_{surface}$ (x, y)

  Where $I_{background}$ (x, y) is background intensity, $I_{surface}$ (x, y) is projected intensity value for surface at pixel position (x, y).

6.4 A – Buffer Method

The A-buffer (anti-aliased, area-averaged, accumulation buffer) is an extension of the ideas in the depth-buffer method (other end of the alphabet from "z-buffer"). A drawback of the depth-buffer method is that it deals only with opaque surfaces and cannot accumulate intensity values for more than one surface or transparent surfaces. The A-buffer is incorporated into the REYES ("Renders Everything You Ever Saw") 3-D rendering system. The A-buffer method calculates the surface intensity for multiple surfaces at each pixel position, and object edges can be antialiased. Viewing an opaque surface through a transparent surface requires multiple surface-intensity contributions for pixel positions. Each position in the A-buffer has two fields: depth field and intensity field.



If the depth field is positive, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RCB components of the surface color at that point and the percent of pixel coverage, as illustrated first figure.

If the depth field is negative, this indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data, as in second figure. Data for each surface in the linked list includes: RGB intensity components, opacity parameter (percent of transparency), depth, percent of area coverage, surface identifier, other surface-rendering parameters, and pointer to next surface

The A-buffer can be constructed using methods similar to those in the depth-buffer algorithm. Scan lines are processed to determine surface overlaps of pixels across the individual scan lines. Surfaces are subdivided into a polygon mesh and clipped against the pixel boundaries. Using the opacity factors and percent of surface overlaps, we can calculate the intensity of each pixel as an average of the contributions from the over lapping surfaces.

6.5 Scan – Line Method

This image-space method for removing hidden surfaces is an extension of the scan-line algorithm for filling polygon interiors where, we deal with multiple surfaces rather than one. Each scan line is processed with calculating the depth for nearest view for determining the visible surface of intersecting polygon. When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.

To facilitate the search for surfaces crossing a given scan line, we can set up an ***active list*** of edges from information in the edge table that contain only edges that cross the current scan line, sorted in order of increasing ***x.*** In addition, we define a ***flag*** for each surface that is set **on** or **off** to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned on; and at the rightmost boundary, it is turned off.

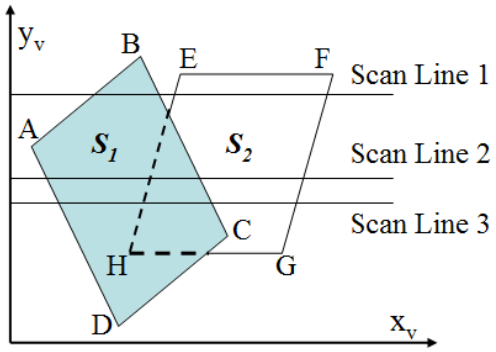


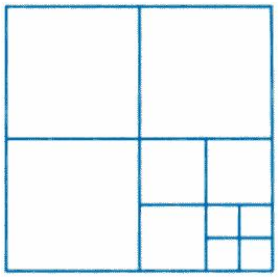
Fig. Scan lines crossing the projection of two surfaces, S_1 and S_2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.

The active list for scan line-1 contains information from the edge table for edges AB, BC, EH, and FG. For positions along this scan line between edges AB and BC, only the flag for surface S_1 is on. Therefore, no depth calculations are necessary, and intensity information for surface S_1, is entered from the polygon table into the refresh buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is on. NO other positions along scan line-1 intersect surfaces, so the intensity values in the other areas are set to the background intensity.

For scan lines 2 and 3, the active edge list contains edges AD, EH, BC, and FG. Along scan line 2 from edge AD to edge EH, only the flag for surface S_1, is on. But between edges EH and BC, the flags for both surfaces are on. In this interval, depth calculations must be made using the plane coefficients for the two surfaces. For this example, the depth of surface S_1 is assumed to be less than that of S_2, so intensities for surface S_1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for surface S_1 goes off, and intensities for surface S_2 are stored until edge FG is passed.

6.6 Area Subdivision Method

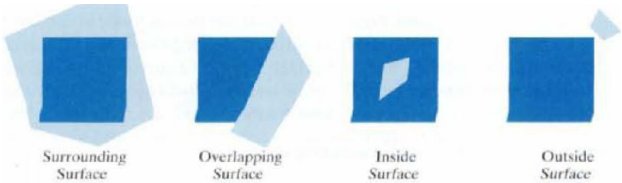
This technique for hidden-surface removal is essentially an image-space method, but object-space operations can be used to accomplish depth ordering of surfaces. The area-subdivision method successively divides the total viewing area into smaller and smaller rectangles until each small area is simple. An easy way to do this is to successively divide the area into four equal parts at each step.



Dividing a square area into equal-sized quadrants at each step.

The procedure to classify each of the surfaces is based on the relations with the area which may be of following type:

- Surrounding surface: a single surface completely encloses the area
- Overlapping surface: a single surface that is partly inside and partly outside the area
- Inside surface: a single surface that is completely inside the area
- Outside surface: a single surface that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true:

1. All surfaces are outside surfaces with respect to the area.
2. Only one inside, overlapping, or surrounding surface is in the area.
3. A surrounding surface obscures all other surfaces within the area boundaries.

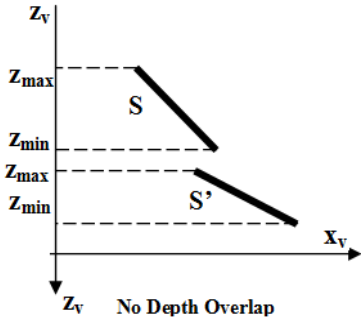
6.7 Depth –Sorting Method

This method for solving the hidden-surface problem by using both image-space and object-space operations in following manner:

- Surfaces are sorted in order of decreasing depth (*using both image and object space*).
- Surfaces are scan converted in order, starting with the surface of greatest depth (*using image space*).

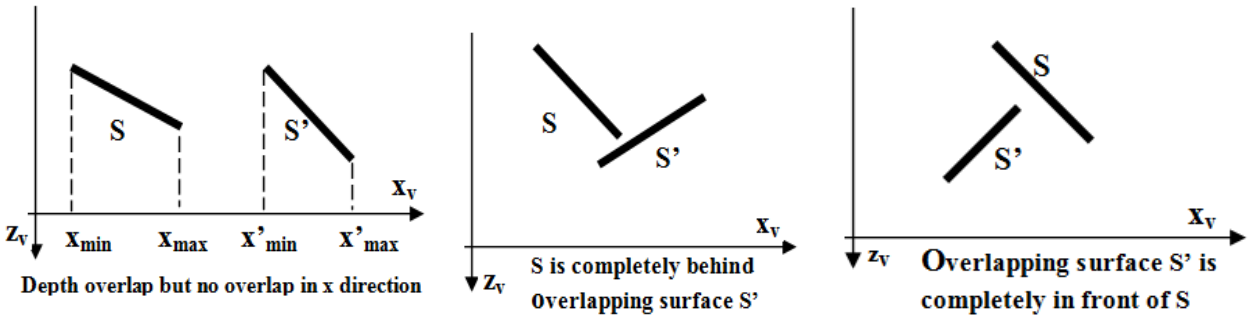
This method is often referred to as the ***painter's algorithm*** because an artist first paints the background colors, then the most distant objects are added, then the nearer objects, and so forth. At the final step, the foreground objects are painted on the canvas over the background and other objects that have been painted on the canvas. Thus, each layer of paint covers up the previous layer.

Similarly, we first sort surfaces according to their distance from the view plane. The intensity values for the farthest surface are then entered into the refresh buffer. Taking each succeeding surface in turn (in decreasing depth order), we "paint" the surface intensities onto the frame buffer over the intensities of the previously processed surfaces.



As long as no overlaps occur, each surface is processed in depth order until all have been scan converted. If a depth overlap is detected at any point in the list, the surfaces should be reordered. Here are some tests for each surface that overlaps with S. If any one of these tests is true, no reordering is necessary for that surface.

a) Bounding rectangle in xy plane for two surfaces don't overlap
b) Surface S is completely behind overlapping surface relative to viewing position
c) Overlapping surface S' is completely in-front of S relative to viewing position
d) Projection of 2 surfaces onto view plane doesn't overlap.



- *Why it is necessary to detect visual surface? Explain depth buffer method.*
- *How would you detect visible surface using back face detection method?*
- *How would you detect visible surface using z-buffer method?*
- *How does scan conversion algorithm detect hidden surface?*

CHAPTER – 7

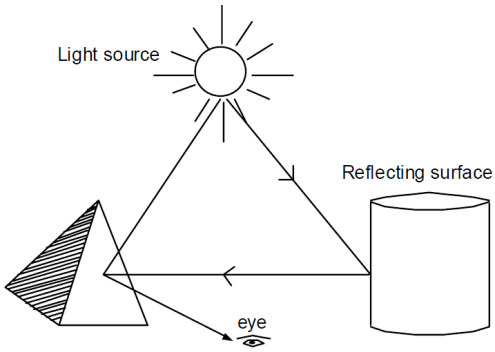
Illumination and Shading

7.1 Illumination Models and Surface Rendering Methods

Realistic displays of a scene are obtained by generating perspective projections of objects and by applying natural lighting effects to the visible surfaces. An *illumination model* or a *lighting model* or a *shading model*, is used to calculate the intensity of light that we should see at a given point on the surface of an object. A surface-rendering algorithm uses the intensity calculations from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene. Surface rendering can be performed by applying the illumination model to every visible surface point.

Light Sources

When we view an opaque non-luminous object, we see reflected light from the surfaces of the object. The total reflected light is the sum of the contributions from a single light source or illuminated nearby objects.



- ***Point source***: Simplest model for a light emitter like tungsten filament bulb
- ***Distributed light source***: The area of the source is not small compared to the surfaces in the scene like fluorescent light on any object in a room
- ***Diffuse reflection:*** Scatter reflected light in all direction by rough or grainy surfaces.
- ***Specular-reflection***: highlights or bright spots created by light source, particularly on shiny surfaces than on dull surfaces.

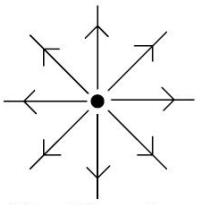


Fig. Diverging ray paths From a point light Source.

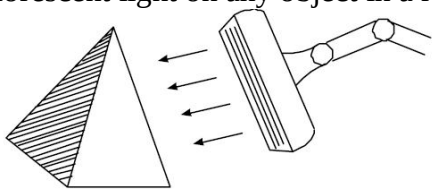


Fig. An object illuminated with a distributed light source.

7.2 Ambient Light and Reflectons

On the basis of the optical properties of surfaces such as glossy, matte, opaque, and transparent, the empirical models are described that provide simple and fast methods for calculating surface intensity at a given point with higher accuracy for most scenes.

Ambient light

In case of ***ambient light***, or ***background light***, the surface of interest is not exposed directly to a light source but reflections from various surfaces to produce a uniform illumination or visibility due to illuminated nearby objects. Ambient light has no spatial or directional characteristics. The amount of ambient light incident on each object is a constant for all surfaces and over all directions.

Diffuse Reflections

Ambient-light reflection is an approximation of global diffuse lighting effects. Diffuse reflections are constant over each surface in a scene, independent of the viewing direction. The ***diffuse-reflection coefficient***, or ***diffuse reflectivity,*** $\mathbf{k_d}$ (varying from 0 to 1) define the fractional amount of the incident light that is diffusely reflected. The parameter k_d (actually function of surface color) depends on the reflecting properties of material so for highly reflective surfaces, the k_d nearly equal to 1. If a surface is exposed only to ambient light, we can express the intensity of the diffuse reflection at any point on the surface as: $\mathbf{I_{ambDiff}} = \mathbf{k_d} \cdot \mathbf{I_a}$, (where I_a = intensity of ambient light)

If N is the unit normal vector to a surface and L is the unit direction vector to the point light source from a position on the surface then cos θ = N.L and the diffuse reflection equation for single point-source illumination is:

$$I_{l,\,diff} = k_d\, I_l \cos\theta = k_d\, I_l\, (N.L)$$

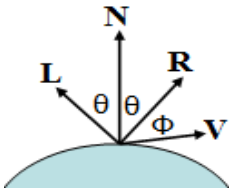
We can combine the ambient and point source intensity calculations to obtain an expression for the total diffuse reflection. Thus, we can write the total diffuse reflection equation as:

$$I_{Diff} = k_d\, .\, I_a + k_d\, I_l\, (N.L)$$

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 directions. This phenomenon, called ***specular reflection***, is the result of total, or near total, reflection of the incident light in a concentrated region around the "specular reflection angle". For an ***ideal reflector*** (perfect mirror), the angle of incident is equal to the angle of reflection or i.e. V and R coincide (Φ = 0).

* 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
* Ø – viewing angle relative to specular reflection direction R



Phong Model

Objects other than ideal reflectors exhibit specular reflections over a finite range of viewing positions around vector R. Shiny surfaces have a narrow specular-reflection range, and dull surfaces have a wider reflection range. An empirical model for calculating the specular-reflection range is called the ***Phong specular-reflection model***, or simply the ***Phong model***, sets the intensity of specular reflection proportional to $\cos^{ns}Ø$. Angle Ø can be assigned values in the range 0 to 90 degree, so that cosØ varies from *0* to 1.

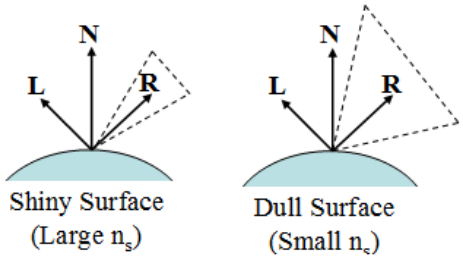


Fig. Modeling specular reflections (shaded area) with parameter n_s.

The value assigned to *specular-reflection parameter* ***n,*** is determined by the type of surface that we want to display. **A** very shiny surface is modeled with a large value for ***n,*** (say, 100 or more), and smaller values (down to 1) are used for duller surfaces. For a perfect reflector, ***n,*** is infinite. For a rough surface, such as chalk **n,** would be assigned a value near 1. Figures show the effect of ***n,*** on the angular range for which we can expect to see specular reflections.

For monochromatic specular reflections intensity variations can be approximated by ***specular-reflection coefficient***, W(θ). The W(θ) tends to increase as θ increases, at θ = 90 degree, W(θ) = 1 and all incident light is reflected. Using ***Fresnal's law of reflection,*** Phong specular reflection model can be describes as: $\mathbf{I_{spec} = W(\theta)\, I_L\, Cos^{ns}\, Ø,}$ Where I_L is intensity of light source, Ø is viewing angle relative to the specular reflection direction R.

Since V and R are unit vectors in the viewing and specular-reflection directions, we can calculate the value of cos Ø with the dot product V R. This expression also can be expressed as:

$$I_{spec} = W(Ø)\, I_L\, (V.R)^{ns}$$

Note: The total intensity of a light is the sum of intensity of ambient light, diffused light & the specular light.

- *Explain what do you mean by illumination model? Explain the Phong model for intensity calculation.*
- *Derive an equation for calculating the total intensity of light at any point on a surface.*
- *Explain the significant of shading algorithm used for bringing about visual realism using Phong shading.*
- *What are the Match bands? Write an algorithm that reduces the match bands.*

7.3 Surface Shading Method

Here, we consider the application of an illumination model to rendering of standard graphics objects formed with polygon surfaces, with the help of scan-line algorithms.

A. Constant-Intensity Shading

The constant-intensity shading or flat shading is a fast and simple method for rendering an object with polygon surfaces where a single intensity is calculated for each polygon. All points over the surface of the polygon are then displayed with the same intensity value. Constant shading can be useful for quickly displaying the general appearance of a curved surface, but seems the ***intensity discontinuity***. In general, flat shading of polygon facets provides an accurate rendering for an object if all of the following assumptions are valid

* 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 so that **N. L** and the attenuation function are constant over the surface.
* The viewing position is sufficiently far from the surface so that **V. R** is constant over the surface.

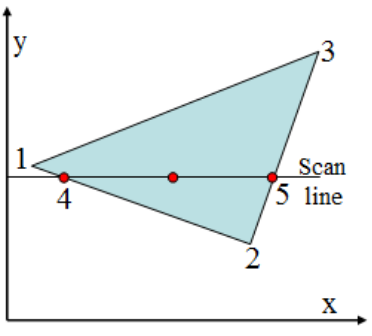
Even if all conditions are not true, we can still reasonable approximate surface lighting effects using small polygon facets with flat shading and calculate the intensity for each facet at the center of the polygon.

B. Gouraud Shading

This intensity-interpolation scheme renders a polygon surface by linearly interpolating intensity values across the surface. Intensity values for each polygon are matched with the values of adjacent polygons along the common edges, thus eliminating the intensity discontinuities that can occur in flat shading. Each polygon surface is rendered with Gouraud shading by performing the following calculations:

* 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.

The figure demonstrates the next step: interpolating intensities along the polygon edges. For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints.



For Gouraud shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point p is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2 \qquad I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

Advantages: Removes discontinuities of intensity at the edge compared to constant shading model

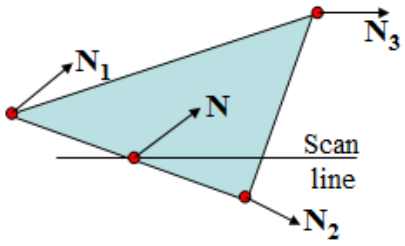
Limitations: Highlights on the surface are sometimes displayed with anomalous shapes and linear intensity interpolation can cause bright or dark intensity streaks, called ***Mach Bands*** to appear on the surfaces. Mach bands can be reduced by dividing the surface into a greater number of polygon faces or Phong shading (*requires more calculation*).

C. Phong Shading

A more accurate method for rendering a polygon surface is Phong shading, or normal vector interpolation shading which first interpolate normal vectors, and then apply the illumination model to each surface point. It displays more realistic highlights on a surface and greatly reduces the Mach-band effect. A polygon surface is rendered using Phong shading by carrying out the following steps:

* Determine the average unit normal vector at each polygon vertex.
* Linearly interpolate the vertex normals over the surface of the polygon.
* Apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.

The normal vector **N** for the scan-line intersection point along the edge between vertices 1 and 2 can be obtained by vertically interpolating between edge endpoint normals:



$$\mathbf{N} = \frac{y - y_2}{y_1 - y_2} \mathbf{N}_1 + \frac{y_1 - y}{y_1 - y_2} \mathbf{N}_2$$

Incremental methods are used to evaluate normal between scan lines and along each individual scan line (as in Gouraud). At each pixel position along a scan line the illumination model is applied to determine the surface intensity at that point. Intensity calculations using an ***approximated normal vector*** at each point along the scan line produce more accurate results than the ***direct interpolation of intensities***, as in Gouraud shading but it requires considerable more calculations.

D. Fast Phong Shading (FPS)

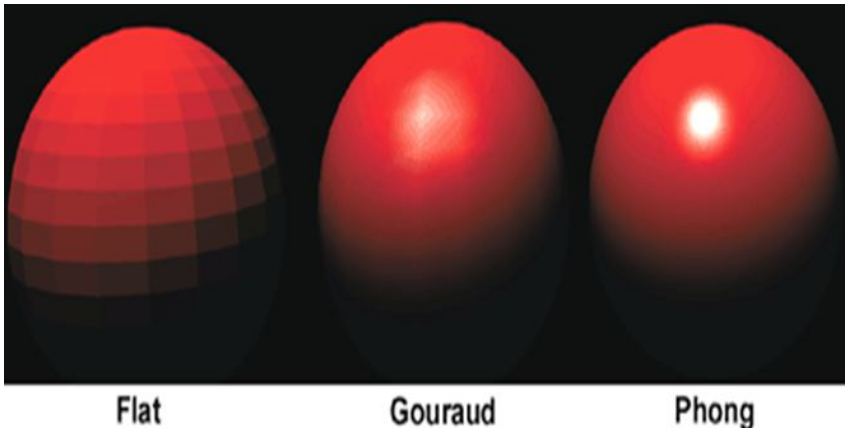
FPS approximates the intensity calculations using a Taylor series expansion and triangular surface patches. The surface normal **N** at any point (x, y) over a triangle as: **N = Ax + By + C**

Where A, B, C are determined from three vertex equations: $N_k = Ax_k + By_k + C$,
k = 1, 2, 3 (x_k, y_k vertex position)

Omitting the reflectivity and attenuation parameters, we can write the calculation for light-source diffuse reflection from a surface point *(x*, y) as:

$$\mathbf{I_{diff}(x, y)} = \frac{\mathbf{L \,.\, N}}{|\mathbf{L}| \,.\, |\mathbf{N}|} = \frac{\mathbf{L \,.\, (Ax + By + C)}}{|\mathbf{L}| \,.\, |\mathbf{Ax + By + C}|}$$

- FPS is two times slower than Gouraud shading
- Normal Phong shading is 6 to 7 times slower than Gouraud
- FPS can be extended to include specular reflections



Note: Depth Cueing

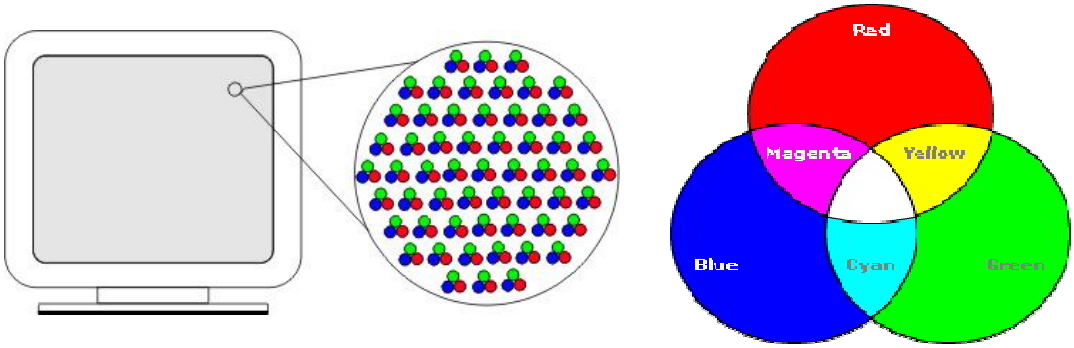
Depth cuing indicates the process in which the lines closest to the viewing position are displayed with the highest intensities, and lines farther away are displayed with decreasing intensities. Depth cueing is applied by choosing maximum and minimum intensity (or color) values and a range of distances over which the intensities are to vary.

7.4 Computer Color Models

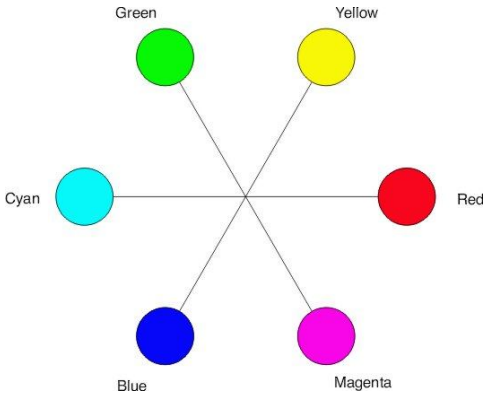
Models or methodologies used to specify colors in computer terms are RGB, HSB, HSL, CMYK, CIE, and others. The main purpose of these color models is for the sensing, representation, and display of images in electronic systems, such as televisions and computers. Although the eye perceives colors based upon red, green, and blue, there are actually two basic methods of making color in computer system: additive and subtractive.

A. Additive Color: RGB

Additive color models use light to display color. A color is created by combining colored light sources in three primary colors: red, green, and blue (**RGB**). This is the process used for cathode ray tube (CRT), liquid crystal (LCD), and plasma displays. On the back of the glass face of a CRT are thousands of phosphorescing chemical dots. These dots are each about 0.30mm or less in diameter (the **dot pitch**), and are positioned very carefully and very close together, arranged in **triads** of red, green, and blue. These dots are bombarded by electrons that "paint" the screen at high speeds (about 60 times a second). The red, green, and blue dots light up when hit by the electron beam. Our eye sees the combination of red, green, and blue light and interpolates it to create all other colors. Like CRTs, LCD and plasma screens utilize tiny red, green, and blue elements energized through tiny transparent conductors and organized in a Cartesian grid.



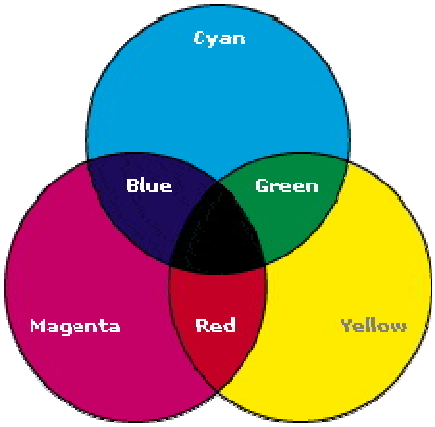
B. Subtractive Color: CMYK



The overlapping of additive colors (red, green and blue) results in subtractive colors (cyan, magenta and yellow). Subtractive models use printing inks. Colors perceived in subtractive models are the result of reflected light. The overlapping of subtractive colors (cyan, magenta and yellow) results the additive colors.

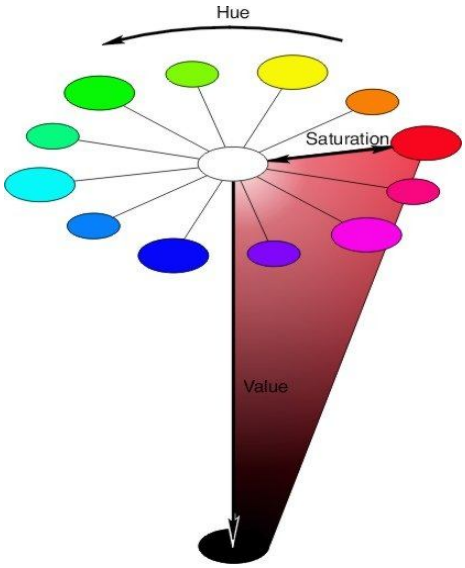
Subtractive color is the process used to create color in printing. The printed page is made up of tiny halftone dots of three primary colors: cyan, magenta, and yellow (designated as CMY). Four-color printing includes black (which is technically not a color but, rather, the absence of color). Since the letter B is already used for blue, black is designated with a K (so four-color printing is designated as CMYK). The CMYK color model is less applicable to multimedia production. It is used primarily in the printing trade where cyan, magenta, yellow, and black are used to print process color separations.

RGB Combination (R,G,B)	Perceived Color
Red only (255,0,0)	Red
Green only (0,255,0)	Green
Blue only (0,0,255)	Blue
Red and green (blue subtracted) (255,255,0)	Yellow
Red and blue (green subtracted) (255,0,255)	Magenta
Green and blue (red subtracted) (0,255,255)	Cyan
Red, green, and blue (255,255,255)	White
None (0,0,0)	Black



Note: HSB and HSL Color Model

These three dimensions of color: hue, saturation, and value constitute a color model that describes how humans naturally respond to and describe color: the **HSV model**. In the HSB (hue, saturation, brightness) and HSL (hue, saturation, lightness) models, we specify hue or color as an angle from 0 to 360 degrees on a color wheel, and saturation, brightness, and lightness as percentages. Saturation is the intensity of a color. At 100 percent saturation a color is pure; at 0 percent saturation, the color is white, black, or gray. Lightness or brightness is the percentage of black or white that is mixed with a color. A lightness of 100 percent will yield a white color; 0 percent is black; the pure color has 50 percent lightness.



CHAPTER – 8

Graphical Languages

8.1 Graphics Software

There are two general classifications for graphics:

A. **General Programming Packages:** 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. E.g. GL (Graphics Library) system on Silicon Graphics equipment, which includes basic functions for generating picture components (straight lines, polygons, circles, and other figures), setting colors and intensity values, selecting views, and applying transformations

B. **Special Purpose Application Packages:** 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. For example: the artist's painting programs and various business, medical, and *Computer-Aided Design (CAD) systems.*

There are many graphics software available in market; they can be categories as:

1. **Paint program:** Paint program works with bit map images.
2. **Photo manipulation program:** It works with bit map images and is widely used to edit digitized photographs.
3. **Computer Aided Design program:** CAD software is used in technical design fields to create models of objects that will be built or manufactured. CAD software allows user to design objects in 3 dimensions and can produce 3-D frames and solid models.
4. **3-D Modelling Programs:** It is used to create visual effects. 3-D modeling program works by creating objects like surface, solid, polygon etc.
5. **Animation:** Computer are used to create animation for use in various fields, including games, and movies composing to allow game makers and film makers to add characters and objects to scenes that did not originally contain them.

8.2 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 System)**, 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. **PHIGS** 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.

- **OpenGL** (**Open G**raphics **L**ibrary) 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 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. 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).

8.3 Major Graphic File Formats

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.

In each image file, the image size is dependent to the number of rows and column. The more rows and columns implies 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 true-color.

Image compression is a method of using algorithms to decrease file size. High resolution cameras produce large image files. Files 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 or 1 million pixels) images, or more, in true-color. 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. Thus, the image compression techniques defined on image file format have greater role to store the high resolution image with less memory requirement. 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**.

A. 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. 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. It is basically used for simple diagrams, shapes, logos and cartoon style images. 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.

B. JPEG: Like GIF, the **Joint Photographic Experts Group** format is one of the most popular formats for Web graphics that use the lossy compression techniques. 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. 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.

C. **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. The main advantage of BMP files is their wide acceptance, simplicity, and use in Windows programs.

D. **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 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.

E. **PNG:** The **Portable Network Graphics** format will likely be the 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. The lossless PNG format 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.

Note: If the images are for the Web or online, use JPEG, PNG, or GIF. If the images are for print, use TIFF.

8.4 Introduction to OpenGL

OpenGL (**Open G**raphics **L**ibrary) is a cross-language, multi-platform Application programming interface (API) for rendering 2D and 3D computer graphics. The API is typically used to interact with a Graphics processing unit (GPU), to achieve hardware-accelerated rendering. OpenGL was developed by Silicon Graphics Inc. (SGI) from 1991 and released in January 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation, and video games.

Many games also use OpenGL for their core graphics-rendering engines. Note also that OpenGL is just a graphics library; unlike DirectX, it does not include support for sound, input, networking, or anything else not directly related to graphics. The following list briefly describes the major graphics operations that OpenGL performs to render an image on the screen.

1. Construct shapes from geometric primitives (points, lines, polygons, images, and bitmaps).
2. Arrange the objects in three-dimensional space and select the desired vantage point for viewing the composed scene.
3. Calculate the colors of all the objects.
4. Convert the mathematical description of objects and their associated color information to pixels on the screen. This process is called *rasterization*.

OpenGL Rendering Pipeline

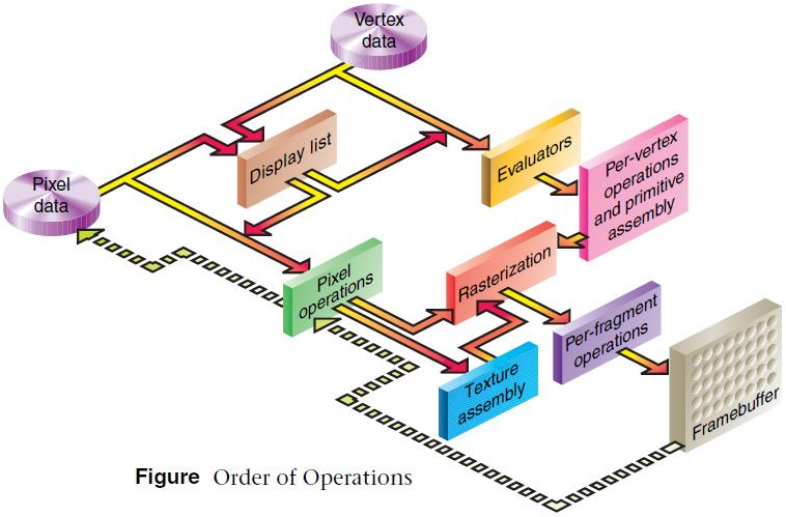


Figure Order of Operations

- **Display Lists:** All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use.
- **Evaluators:** All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called *basis functions*. Evaluators provide a method for deriving the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.
- **Per-Vertex Operations:** For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some types of vertex data (for example, spatial coordinates) are transformed by 4 X 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.
- **Primitive Assembly:** Clipping, a major part of primitive assembly, is the elimination of portions of geometry that fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending on how the line or polygon is clipped. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.
- **Pixel Operations:** While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.
- **Texture Assembly:** OpenGL applications can apply texture images to geometric objects to make the objects look more realistic, which is one of the numerous techniques enabled by texture mapping. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them.
- **Rasterization:** Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support anti-aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are generated for each fragment square.
- **Fragment Operations:** Before values are actually stored in the frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled. The first operation that a fragment might encounter is texturing, where a *texel* (texture element) is generated from texture memory for each fragment and applied to the fragment. Next, primary and secondary colors are combined, and a fog calculation may be applied. If your application is employing fragment shaders, the preceding three operations may be done in a shader. After the final color and depth generation of the previous operations, the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is does hidden-surface removal) are evaluated, if enabled. Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processed fragment is drawn into the appropriate buffer, where it has finally become a pixel and achieved its final resting place.