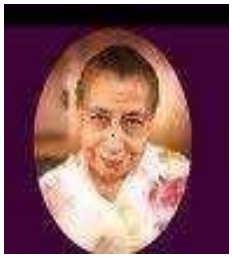
ANNAI WOMEN'S COLLEGE

(ARTS & SCIENCE)
TNPL ROAD, PUNNAMCHATRAM, KARUR – 639136



DEPARTMENT OF COMPUTER SCIENCE

COURSE MATERIAL

Faculty Name: Mrs. M. Priya ***Class: III B.Sc(cs)***

Subject Name: COMPUTER GRAPHICS ***Subject code: 16SMBECS2:1***

MAJOR BASED ELECTIVE II (A)
COMPUTER GRAPHICS

Objective:

To understand the concepts on basic Graphical Techniques, Raster Graphics, Two Dimensional and Three Dimensional Graphics

Unit I

Overview of Computer Graphics System: Video Display Devices – Raster Scan Systems – Random – Scan Systems - Graphics Monitors and Workstations – Input Devices – Hardcopy Devices – Graphics Software.

Unit II

Output Primitives: Line Drawing Algorithms – Loading the Frame Buffer –Line Function – Circle – Generating Algorithms. Attributes of Output Primitives: Line Attributes – Curve Attributes – Color and Grayscale levels– Area fill Attributes – Character Attributes – Bundled Attributes – Inquiry Functions.

Unit III

2D Geometric Transformations: Basic Transformation – Matrix Representations – Composite Transformations – Window to View port Co-Ordinate Transformations. Clipping: Point Clipping – Line Clipping – Cohen-Sutherland Line Clipping – Liang Barsky Line Clipping – Polygon Clipping – Sutherland – Hodgman Polygon Clipping – Curve Clipping – Text Clipping.

Unit IV

Graphical User Interfaces and Interactive Input Methods: The User Dialogue – Input of Graphical Data – Input Functions – Interactive Picture Construction Techniques. Three Dimensional Concepts: 3D-Display Methods – #Three Dimensional Graphics Packages

Unit V

3D Geometric and Modeling Transformations: Translation – Scaling – Rotation – Other Transformations.Visible Surface Detection Methods: Classification of Visible Surface Detection Algorithm –Backface Detection – Depth-Buffer Method – A-Buffer Method –Scan-Line Method – Applications of Computer Graphics.

Text Book:

1. Donald Hearn M. Pauline Baker, Computer Graphics C Version, Second Edition,
Pearson Education, 2014.

1. OVERVIEW OF GRAPHICS SYSTEMS

Definition:

Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. Computer graphics is a rendering tool for the generation and manipulation of images.

In other words, we can say that it is a visual representations of data displayed on a monitor. It can be a series of images (most often called video) or a single image. It is used for making movie, video game, scientific modeling, design for catalogs and other commercial art.

Classification of Computer Graphics:

Computer graphics as drawing pictures on computers, also called rendering. 2D computer graphics are usually split into two categories:

1. Vector Graphics
2. Raster graphics

Vector Graphics

- Vector graphics is the creation of digital images through a sequence of commands or mathematical statements.
- In Vector graphics lines, shapes, and text are used to create a more complex image.
- Vector graphics are made with programs like Adobe Illustrator and Inkscape etc…
- Vector graphics image is shown in Fig 1.1.

Raster Graphics

- Raster Graphics or Bitmap Image is a dot matrix data structure, representing rectangular grid of pixels, or points of color.
- Raster images are stored in image files with varying formats.
- Raster graphics use pixels to make up a larger image.
- Raster programs are made by Paintbrushes Adobe Photoshop and Corel Paint Shop Pro.
- Sometimes people do use only pixels to make an image. This is called pixel art and it has a very unique style. Raster image is shown in Fig 1.2.

Fig 1.1 Vector Image

Fig 1.2 Raster Image

VIDEO DISPLAY DEVICES:

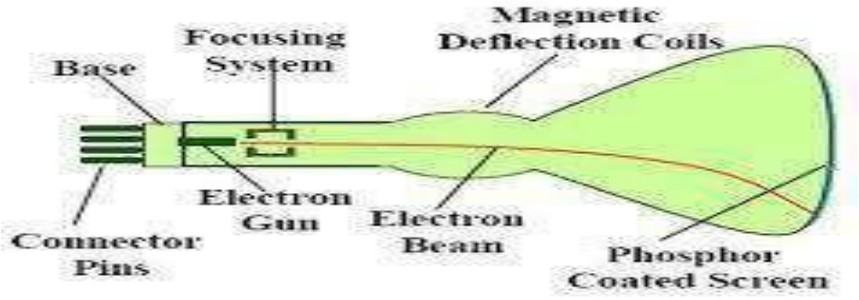
The primary output device in a graphics system is a video monitor is shown in Fig 1.3. The operation of most video monitors is based on the standard cathode-ray tube (CRT) design.

Refresh Cathode-Ray Tubes

- A beam of electrons (cathode rays), emitted by an electron gun, passes through focusing and deflection systems that direct the beam toward specified positions on the phosphor-coated screen.

Fig 1.3 computer graphics workstation

- The phosphor then emits a small spot of light at each position contacted by the electron beam.



Basic design of a magnetic deflection CRT

Figure 1.4 Basic design of a magnetic deflection CRT

- Because the light emitted by the phosphor fades very rapidly, some method is needed for maintaining the screen picture.
- One way to keep the phosphor glowing is to redraw the picture repeatedly by quickly directing the electron beam back over the same points. This type of display is called a *refresh CRT*.
- The primary components of an electron gun in a CRT are the heated metal cathode and a control grid (Fig.1.5).
- Heat is supplied to the cathode by directing a current through a coil of wire, called the *filament.*
- This causes electrons to be 'boiled off" the hot cathode surface. In the vacuum inside the CRT envelope, the free, negatively charged electrons are then accelerated toward the phosphor coating by a high positive voltage.

- Sometimes the electron gun is built to contain the accelerating anode and focusing system within the same unit.

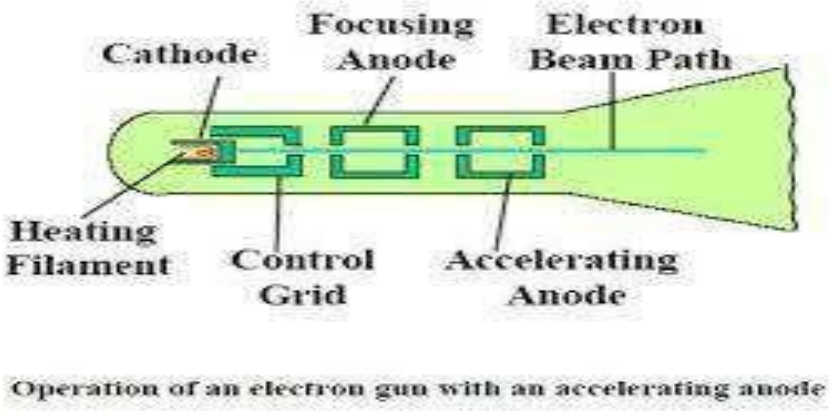


Fig 1.5 Operation of an electron gun with an accelerating anode

- Intensity of the electron beam is controlled by setting voltage levels on the control grid, which is a metal cylinder that fits over the cathode.
- A high negative voltage applied to the control grid will shut off the beam by repelling electrons; a smaller negative voltage on the control grid simply decreases the number of electrons passing through.
- The amount of light emitted by the phosphor coating depends on the number of electrons striking the screen.
- The focusing system in a CRT is needed to force the electron beam to converge into a small spot as it strikes the phosphor.
- Focusing is accomplished with either electric or magnetic fields. Electrostatic focusing is commonly used in television and computer graphics monitors.
- Additional focusing hardware is used in high precision systems to keep the beam in focus at all screen points.
- Deflection of the electron beam can be controlled either with electric fields or with magnetic fields. Cathode ray tubes are constructed with magnetic deflection coils mounted on the outside of the CRT envelope.
- Two pairs of coils are used, with the coils in each pair mounted on opposite sides of the neck of the CRT envelope.
- One pair is mounted on the top and bottom of the neck and the other pair is mounted on opposite sides of the neck.
- Horizontal deflection is accomplished with one pair of coils, and vertical deflection by the other pair.
- One pair of plates is mounted horizontally to control the vertical deflection, and the other pair is mounted vertically to control horizontal deflection (Fig 1.6).

Persistence:

- Persistence means how long the phosphors continue to emit light after the CRT beam is removed.
- A phosphor with low persistence is useful for animation.
- A higher persistence phosphor is useful for displaying highly complex, static pictures.

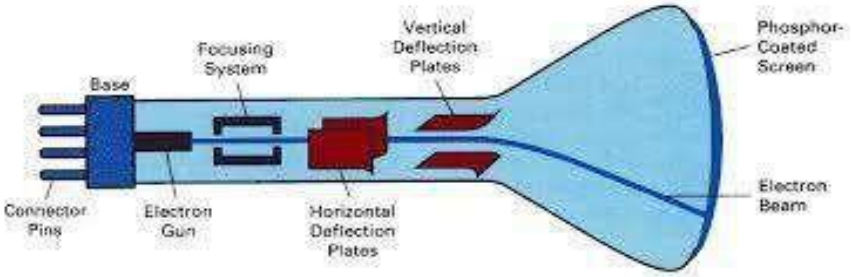


Fig 1.6 Electrostatic deflection of the electron beam in a CRT

Resolution:

- Resolution is the number of pixels contained on a display monitor, expressed in terms of the number of pixels on the horizontal axis and vertical axis.
- It can be also referred as maximum number of points that can be displayed without overlap on a CRT.
- The sharpness of the image on a display depends on the resolution and the size of the monitor.

Aspect ratio:

- The ratio of vertical points to the horizontal points necessary to produce length of lines in both directions of the screen is called the aspect ratio.
- An aspect ratio of ¾ means that a vertical line plotted with three points has the same length as a horizontal line plotted with four points.

Raster-Scan Displays

- Common type of graphics monitor employing a CRT is the raster-scan display, based on television technology.

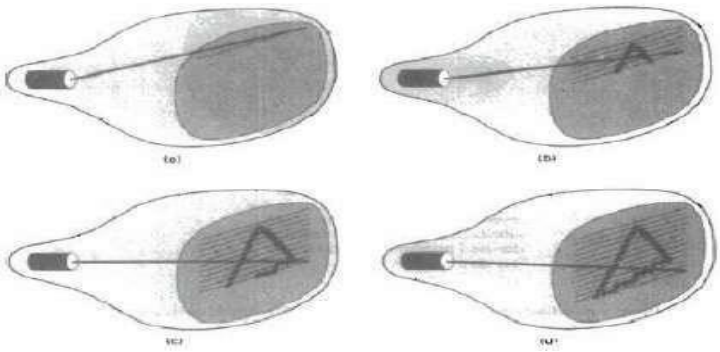


Fig. 1.7 Raster Scan System displays

- In a raster-scan system, the electron beam is swept across the screen, one row at a time from top to bottom.
- The electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.

Refresh Buffer or Frame Buffer

Picture definition is stored in a memory area called the refresh buffer or frame buffer.

Scan Line

The frame buffer holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and "painted" on the screen one row at a time is called scan line (Fig. 1.7).

Pixel

- Each screen point is referred to as a pixel or pel or picture element.
- Intensity range for pixel positions depends on the capability of the raster system.
- In a simple black-and-white system, each screen point is either on or off. So only one bit per pixel is needed to control the intensity of screen positions.
- For a bi-level system, a bit value of 1 indicates that the electron beam is to be turned on at that position, and a value of 0 indicates that the beam intensity is to be off.

Bitmap

On a black-and-white system with one bit per pixel, the frame buffer is commonly called a bitmap.

Pixmap

- Systems with multiple bits per pixel, the frame buffer is often referred to as a pixmap.
- Refreshing on raster scan displays is carried out at the rate of 60 to 80 frames per second. Refresh rates are described in units of cycles per second or Hertz.
- Using these units, we would describe a refresh rate of 60 frames per second as simply 60 Hz.

Horizontal Retrace

- At the end of each scan line, 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.

Vertical Retrace

- At the end of each frame the electron beam returns to the top left corner of the screen to begin the next frame is called vertical retrace.

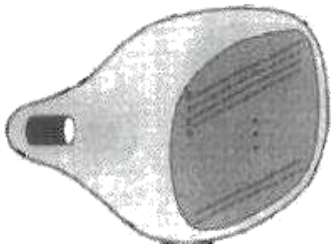


Fig 1.8 Interlacing scan lines on a Raster Scan Display

Interlace

- Each frame is displayed in two passes using an interlaced refresh procedure.
- In the first pass, the beam sweeps across every other scan line from top to bottom.
- Then after the vertical retrace, the beam sweeps out the remaining scan lines (Fig 1.8).

Random-Scan Displays

- In a random-scan display unit, a CRT has the electron beam directed only to the parts of the screen where a picture is to be drawn (Fig 1.9).
- Random-scan monitors draw a picture one line at a time and for this reason are also referred to as vector displays or stroke-writing or calligraphic displays.
- Refresh rate on a random-scan system depends on the number of lines to be displayed.

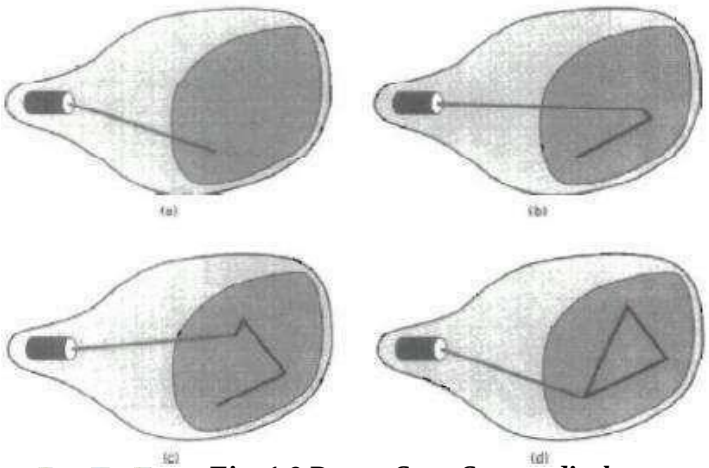


Fig. 1.9 Raster Scan System displays

Refresh Display File

- Picture definition is now stored as a set of line-drawing commands in an area of memory referred to as the refresh display file.
- It is called the display list, display program, or simply the refresh buffer.
- Random-scan displays are designed to draw all the component lines of a picture 30 to 60 times each second.
- High-quality vector systems are capable of handling approximately 100,000 "short" lines at this refresh rate.
- Random-scan systems are designed for line-drawing applications and cannot display realistic shaded scenes.

Color CRT Monitors

A CRT monitor displays colour pictures by using a combination of phosphors that emit different-colored light. By combining the emitted light from the different phosphors, a range of colors can be generated.

The two basic techniques for producing color displays.

1. Shadow-mask method.
2. Beam-Penetration Method

Beam-Penetration Method

- In beam-penetration method two layers of phosphor, usually red and green, are coated onto the inside of the CRT screen.
- The displayed color depends on how far the electron beam penetrates into the phosphor layers.
- A beam of slow electrons excites only the outer red layer.
- A beam of very fast electrons penetrates through the red layer and excites the inner green layer.
- At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow.

Shadow-mask method

- Shadow-mask methods are commonly used in raster scan systems because they produce wider range of colors than the beam penetration method.
- A shadow-mask CRT has three phosphor color dots at each pixel position.
- One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. This type of CRT has three electron guns, one for each color dot, and a shadow-mask grid just behind the phosphor-coated screen.

Types of Shadow-mask:

There are two types of shadow mask

1. Delta – delta Shadow mask
2. Inline shadow mask

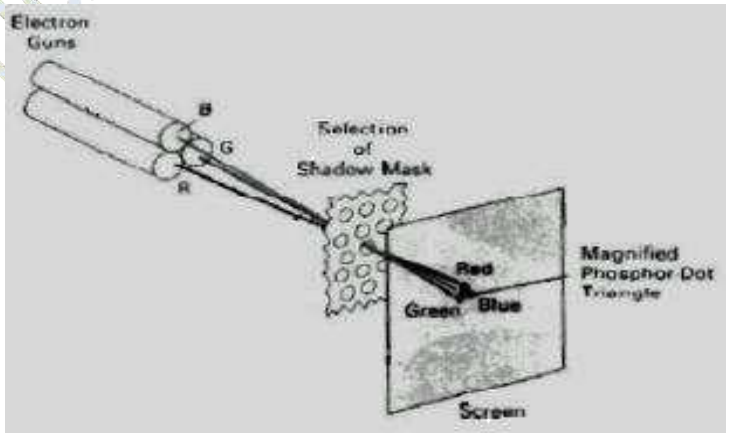


Fig 1.10 Operation of a delta-delta, shadow mask CRT

- ***Delta –delta shadow mask*** method are used in color CRT systems. Three electron beams are deflected and focused, which contains a series of holes aligned with the phosphor dot patterns (Fig 1.10).
- When three beams pass through a hole in the shadow mask, they activate a dot triangle color spot on the screen.
- Another arrangement for the electron gun is ***an in-line arrangement*** in which the three electron guns and the corresponding red-green-blue color dots on the screen are aligned in one scan line instead of triangular pattern.
- These in line arrangement of electron guns are used in high resolution color CRT.

Full Color System or True Color System

An RGB color system with 24 bits of storage per pixel is generally referred to as a full-color system or a true-color system.

Direct-View Storage Tubes

It stores the picture information inside the CRT instead of refreshing the screen. Two electron guns are used in a DVST.

1. Primary Gun
2. Flood Gun

The primary gun is used to store the picture pattern. The second, the flood gun, maintains the picture display. The DVST has both advantages and disadvantages compared to refresh CRT.

Advantages

No refreshing is needed; very complex pictures can be displayed at very high resolutions without flicker.

Disadvantages

- They do not display color and that selected parts of a picture cannot be erased.
- To eliminate a picture section, the entire screen must be erased. Erasing and redrawing process can take several seconds for a complex picture.

Flat-Panel Displays

- The term flat-panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT.
- A significant feature of flat-panel displays is that they are thinner than CRTs, and we can hang them on walls or wear them on our wrists.
- Current uses for flat-panel displays include small TV monitors, calculators, pocket video games, laptop computers, etc.,

There are two categories:

- **Emissive displays or Emitters**
- **Non emissive displays or non emitters**

1. Emissive displays or Emitters

The emissive displays devices that convert electrical energy into light. Plasma panels, thin-film electroluminescent displays and light emitting diodes are examples of emissive displays.

Plasma panels:

- It is also called gas –discharge displays, are constructed by filling the region between two glass plates with mixture of gases, usually includes neon.
- A series of vertical conducting ribbons is placed on one glass panel, and set of horizontal ribbons is built into other glass panel.
- Firing voltage is applied to a pair of horizontal and vertical conductors, the gas at the intersection of two conductors break down into glowing plasma of electrons and ions.
- Picture definition is stored in a refresh buffer, and the firing voltages are applied to refresh the pixel positions 60 times per second. Alternating current methods are used to provide faster application of firing voltages and brighter displays.

Disadvantages:

Plasma panel only applicable for monochromatic devices, but systems have been developed for displaying color and grayscale.

Thin-Film Electroluminescent Displays:

- The construction of Thin-Film Electroluminescent Displays is similar to plasma panel.
- But the difference is that the region between the glass plates is filled with a phosphor such as Zinc sulphide doped with manganese, instead of gas.
- When high voltage is applied to a pair of electrodes, electrical energy is absorbed by manganese atoms then release the spot of light similar to plasma panel.
- It is more powerful than plasma panel and produce good color and gray scale.

LED (Light –emitting diode):

- A matrix of diodes is arranged to form the pixel positions in the display, and picture definition is stored in a refresh buffer.
- Information is read from the refresh buffer and converted to voltage that is applied to the diodes into light pattern in the display.

2. Non emissive displays or non emitters

Non emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns.

Example: LCD

- Liquid-crystal displays (LCDS) are commonly used in small systems, such as calculators and portable, laptop computers.
- Each pixel of an LCD consists of a layer of molecules aligned between two transparent electrodes with light polarizer.

- *Passive-matrix* LCD is an LCD technology that uses a grid of vertical and horizontal conductors comprised of Indium Tin Oxide (ITO) to create an image.
- Another method for constructing LCD is to place a transistor at each pixel position, using thin film transistor technology. The transistors are used to control the voltage at pixel locations are called *active matrix* displays.

RASTER-SCAN SYSTEM:

In raster graphics, in addition to the central processing unit, or CPU, a special-purpose processor, called the *video controller or display controller*, is used to control the operation of the display device. Organization of a simple raster system in shown in Fig.1.11.

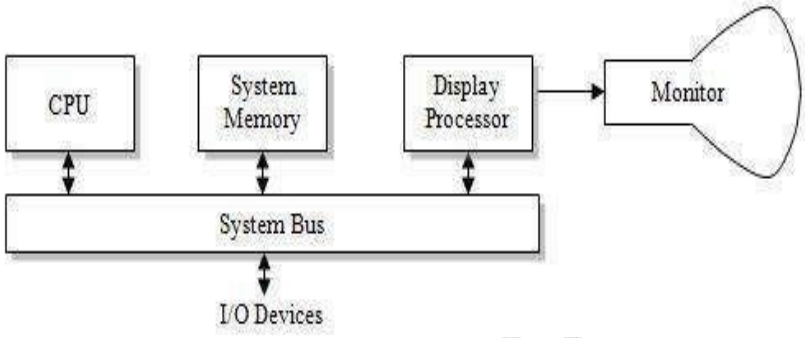


Fig 1.11 Architecture of a simple raster system

Video Controller

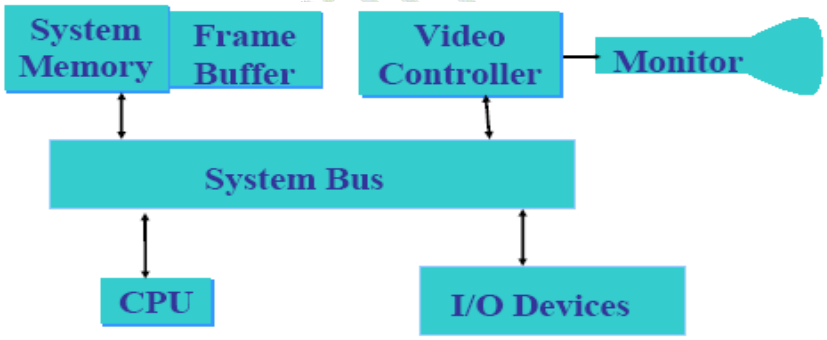


Fig 1.12 Architecture of raster system with a fixed portion of the system memory

A fixed area of the system memory is reserved for the frame buffer. So the video controller is given direct access to the frame-buffer memory (Fig 1.12).

Frame-buffer locations, and the corresponding screen positions, are referenced in Cartesian coordinates. The coordinate origin is defined at lower left screen corner.

- Then the first quadrant of a two dimensional system, positive x values increasing to the right and positive y values increasing from bottom to top.
- Scan lines are labeled from y_{max} at the top of the screen to 0 at the bottom, each scan line screen pixel positions are labeled from 0 to $x_{max.}$

- There are two registers are used to store the coordinates of the screen pixels.
- Initially, the x register is set to 0 and the y register is set to y_{max}.
- The value stored in the frame buffer for this pixel position is then retrieved and used to set the intensity of the CRT beam.
- Then the x register is incremented by 1, and the process repeated for the next pixel on the top scan line.
- In high quality system, two frame buffers are provided, one for refreshing other for filling intensity values.

Raster-Scan Display Processor

A raster system containing a separate display processor, sometimes referred to as a graphics controller or a display coprocessor (Fig 1.13).

Scan Conversion

- Task of the display processor is digitizing a picture definition given in an application program into a set of pixel-intensity values for storage in the frame buffer. This digitization process is called *scan conversion*.
- Display processors are also designed to perform a number of additional operations.
- These functions include generating various line styles (dashed, dotted, or solid), displaying color areas, and performing certain transformations and manipulations on displayed objects.

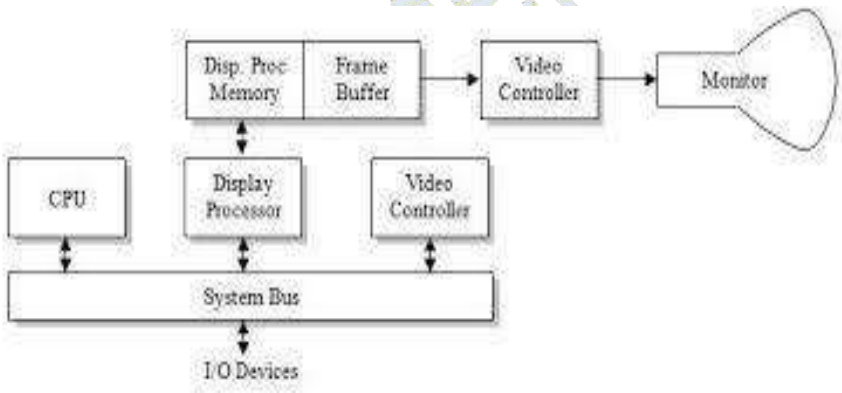


Fig 1.13 Raster Graphics System with Display Processor

Run Length Encoding

- Intensity information is to store each scan line as a set of integer pairs.
- One number of each pair indicates an intensity value, and the second number specifies the number of adjacent pixels on the scan line.
- This technique, called run-length encoding.

Cell Encoding

It is another approach to encode the raster as a set of rectangle areas called cell encoding.

RANDOM SCAN SYSTEMS

The organization of simple random scan system is shown in Fig 1.14.

- An application program is input and is stored in the system memory.
- Graphics commands in the application program are translated by the graphics package into a display file stored in the system memory.
- This display file is then accessed by the display processor to refresh the screen.
- The display processor in a random-scan system is referred to as a display processing unit or a graphics controller.

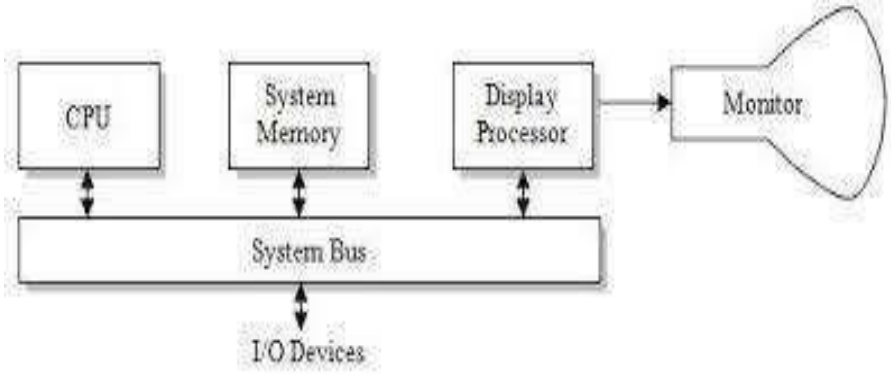


Fig 1.14 Architecture of simple Random Scan System

GRAPHICS MONITORS AND WORKSTATIONS

- Graphics systems are designed as small general-purpose computer systems with graphics capabilities.
- Full-color systems that are designed specifically for graphics applications.
- High-definition graphics monitor used in applications such as air traffic control, simulation, medical imaging, and CAD.
- This system has a diagonal screen size of 27 inches, resolutions ranging from 2048 by 1536 to 2560 by 2048, with refresh rates of 80 Hz or 60 Hz non interlaced.
- Workstation refers to any computer device or program that makes a computer capable of displaying and manipulating pictures.

Fig 1.15 General Purpose Computer System that can be used for Graphics Application

For example, laser printers and plotters are graphics devices because they permit the computer to output pictures.

1.16 Computer Graphics Workstations

INPUT DEVICES

Keyboard

- The keyboard helps in inputting the data to the computer.
- The layout of the keyboard is like that of traditional typewriter, although there are some additional keys provided for performing some additional functions.
- Keyboards are of two sizes 84 keys or 101/102 keys, but now 104 keys or 108 keys keyboard is also available for Windows and Internet.

Fig 1.17 Keyboard

Mouse

- Mouse is most popular Pointing device.
- It is a very famous cursor-control device. It is a small palm size box with a round ball at its base which senses the movement of mouse and sends corresponding signals to CPU on pressing the buttons.
- Generally, it has two buttons called left and right button and scroll bar is present at the mid. Mouse can be used to control the position of cursor on screen, but it cannot be used to enter text into the computer.

Advantages

- Easy to use
- Not very expensive
- Moves the cursor faster than the arrow keys of keyboard

Joystick

- Joystick is also a pointing device, which is used to move cursor position on a monitor screen.
- It is a stick having a spherical ball at its both lower and upper ends.

- The lower spherical ball moves in a socket
- The joystick can be moved in all four directions.
- It is mainly used in Computer Aided Designing (CAD) and playing computer games.

Fig 1.18 Joystick

Fig 1.19 Light pen

Light Pen

- Light pen is a pointing device, which is similar to a pen.
- It is used to select a displayed menu item or draw pictures on the monitor screen.
- It consists of a photocell and an optical system placed in a small tube.
- When light pen's tip is moved over the monitor screen and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signal to the CPU.

Track Ball

- Track ball is an input device that is mostly used in notebook or laptop computer, instead of a mouse.
- This is a ball, which is half inserted and by moving fingers on ball, pointer can be moved.
- Since the whole device is not moved, a track ball requires less space than a mouse.
- A track ball comes in various shapes like a ball, a button and a square.

Fig 1.20 Track Ball

Fig 1.21 Scanner

Scanner

- Scanner is an input device, which works more like a photocopy machine.
- It is used when some information is available on a paper and it is to be transferred to the hard disc of the computer for further manipulation.
- Scanner captures images from the source which are then converted into the digital form that can be stored on the disc.
- These images can be edited before they are printed.

Digitizer

- Digitizer is an input device, which converts analog information into a digital form.
- Digitizer can convert a signal from the television camera into a series of numbers that could be stored in a computer.
- They can be used by the computer to create a picture of whatever the camera had been pointed at.
- Digitizer is also known as Tablet or Graphics Tablet because it converts graphics and pictorial data into binary inputs.

Fig 1.22 Digitizer

Bar Code Readers

- Bar Code Reader is a device used for reading bar coded data. Bar coded data is generally used in labelling goods, numbering the books, etc.
- It may be a hand-held scanner or may be embedded in a stationary scanner.

Fig 1.23 Barcode Readers

- Bar Code Reader scans a bar code image, converts it into an alphanumeric value, which is then fed to the computer to which bar code reader is connected.

HARD COPY DEVICES

- To a obtain hard-copy output for our images in several formats.
- For presentations or archiving, we can send image files to devices or service bureaus that will produce 35-mm slides or overhead transparencies.
- We can put our pictures on paper by directing graphics output to a printer or plotter.
- The quality of the pictures obtained from a device depends on dot size and the number of dots per inch, or lines per inch, that can be displayed.

Printers

Printer is the most important output device, which is used to print information on paper.

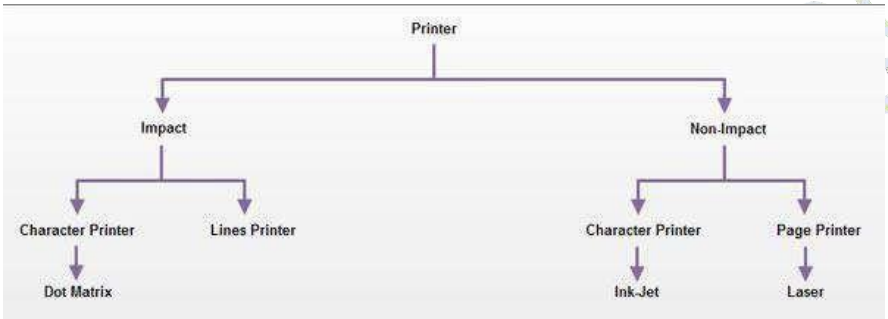
There are two types of printers:

1 Impact Printers

2 Non-Impact Printers

Impact Printers

The printers that print the characters by striking against the ribbon and onto the paper are called impact printers.



1. Character Printer
2. Line Printer

Character Printer:

- It prints only one character at a time.
- It has relatively slower speed. Eg. Dot matrix printers.

Dot Matrix Printer:

- It prints characters as combination of dots.
- Dot matrix printers are the most popular among serial printers.
- These have a matrix of pins on the print head of the printer which form the character.
- The computer memory sends one character at a time to be printed by the printer. There is a carbon between the pins & the paper.
- The words get printed on the paper when the pin strikes the carbon. There are generally 24 pins.

Non-Impact Printers:

- These printers use non-Impact technology such as ink-jet or laser technology.
- These printers provide better quality of O/P at higher speed.

There are two types:

1. Ink-Jet Printer
2. Laser Printer

Ink-Jet Printer:

- It prints characters by spraying patterns of ink on the paper from a nozzle or jet.
- It prints from nozzles having very fine holes, from which a specially made ink is pumped out to create various letters and shapes.

Laser Printer:

- It is a type of printer that utilizes a laser beam to produce an image on a drum.
- This is also the way copy machines work. Because an entire page is transmitted to a drum before the toner is applied, laser printers are sometimes called page printers.

GRAPHICS SOFTWARE

There are two general classifications for graphics software:

General Programming Packages

A general graphics programming package provides an extensive set of graphics functions that can be used in a high-level programming language, such as C or FORTRAN.

Example: Generating picture components straight lines, polygons, circles, and other figures.

Special-Purpose Applications Packages

Application graphics packages are designed for nonprogrammers, so that users can generate displays without worrying about how graphics operations work.

Example: Artist's painting programs and various business, medical, and CAD systems

Coordinate Representations

- Coordinate values for a picture are converted to Cartesian coordinates before they can be input to the graphics package.
- Different Cartesian reference frames are used to construct and display a scene.

Modeling Coordinates

- We can construct the shape of individual objects, such as trees or furniture, in a scene within separate coordinate reference frames called modeling coordinates, or sometimes local coordinates or master coordinates.

World Coordinates

Once individual object shapes have been specified, we can place the objects into appropriate positions within the scene using a reference frame called world coordinates.

Graphics Functions

- A general-purpose graphics package provides users with a variety of functions for creating and manipulating pictures.
- The basic building blocks for pictures are referred to as output primitives. They include character strings and geometric entities, such as points, straight lines, curved lines, filled areas (polygons, circles, etc.).

- **Attributes** are the properties of the output primitives. It includes intensity and color specifications, line styles, text styles, and area-filling patterns.

Geometric Transformations

To change the size, position, or orientation of an object within a scene using geometric transformations.

Modeling Transformations

It is used to construct a scene using object descriptions given in modeling coordinates

Viewing Transformations

- Viewing transformations are used to specify the view that is to be presented and the portion of the output display area that is to be used.
- Pictures can be subdivided into component parts, called structures or segments or objects, depending on the software package in use
- Interactive graphics applications use various kinds of input devices, such as a mouse, a tablet, or a joystick.

Software Standards

- The primary goal of standardized graphics software is portability.
- When packages are designed with standard graphics functions, software can he moved easily from one hardware system to another and used in different implementations and applications.

Graphical Kernel System (GKS)

- It is the first graphics software standard by the International Standards Organization (ISO).
- It is also includes in the American National Standards Institute (ANSI).

PHIGS (Programmer's Hierarchical Interactive Graphics standard)

- It is the second software standard to be developed and approved by the standards organizations.It is an extension of GKS.

PHIGS Workstations

- Workstation refers to a computer system with a combination of input and output devices that is designed for a single user.
- In PHIGS and GKS, however, the term workstation is used to identify various combinations of graphics hardware and software.
- A PHIGS workstation can be a single output device, a single input device, a combination of input and output devices, a file, or even a window displayed on a video monitor.

2. OUTPUT PRIMITIVES

Introduction

- The Primitives are the simple geometric functions that are used to generate various Computer Graphics required by the User.
- Basic Output primitives are point-position (pixel), and a straight line.
- Some other output primitives are rectangle, conic section, circle, or may be a surface.

POINT AND LINES

Point Function

- A point function is the most basic Output primitive in the graphic package.
- A point function contains location using x and y coordinate and the user may also pass other attributes such as its intensity and color.
- The location is stored as two integer tuple, the color is defined using hex codes.
- The size of a pixel is equal to the size of pixel on display monitor.

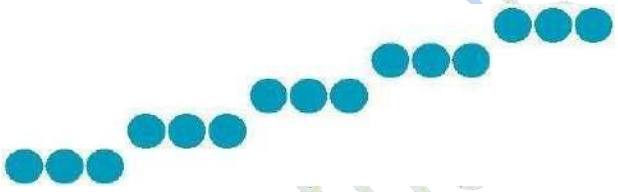


Fig 2.1 line is generated as a series of pixel position

Line Function

- A line function is used to generate a straight line between any two end points.
- Usually a line function is provided with the location of two pixel points called the starting point and the end point.
- The two dimensional line function for specifying straight-line segment is **polyline** (n, wePoints) Where
  1. n – integer value equal to the number of coordinate positions.
  2. wePoints – array of input world coordinate values.
- This line function is used to define a set of n-1 connected straight line segments.

For example

The following statements generate two connected line segments with end point (50, 100) (150, 250) and (250, 100).

```
wcPoints • x[1] = 50;
wcPoints • y[1] = 100;
wcPoints • x[2] = 150;
wcPoints • y[2] = 250;
wcPoints • x[3] = 250;
wcPoints • y[1] = 100;
```

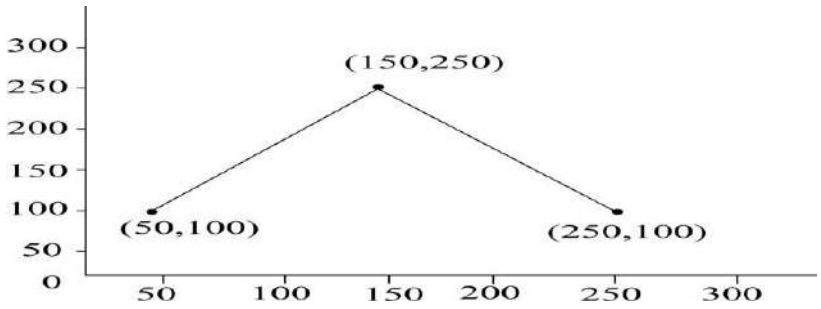


Fig: 2.2 Implementation of line function

LINE DRAWING ALGORITHM

To determine pixel positions along a straight-line path from the geometric properties of the line.The Cartesian slope-intercept equation for a straight line is

$$y = m \cdot x + b \text{ ----------} \quad 1$$

where m as the slope of the line and b as the y intercept.

Given that the two endpoints of a line segment are specified at positions (x_1, y_1) and (x_2, y_2), as shown in Fig.2.2.

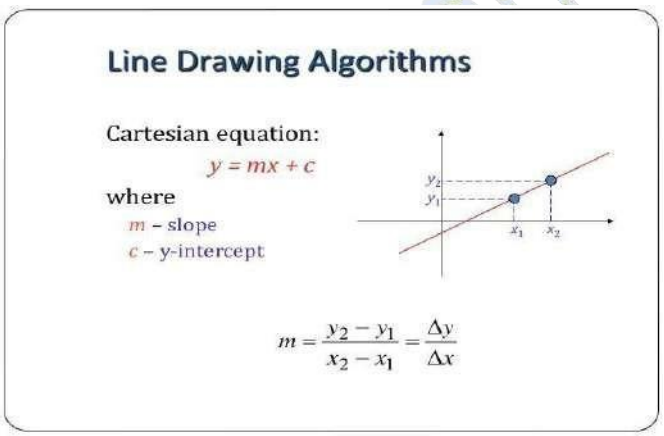


Fig 2.3 Line path between endpoint positions (x1, y1) and (x2, y2)

To determine values for the slope m and y intercept b with the following calculations:

$$m = y_2 - y_1 / x_2 - x_1 \text{ ----------} 2$$

$$b = y_1 - m.\ x_1 \quad - \ 3$$

Algorithms for displaying straight lines are based on the line equation 1 and the calculations given in Eqs. 2 and 3.

For any given x interval Δx along a line, we can compute the corresponding y interval Δy from Eq.2 as

$$\Delta y = m \cdot \Delta x \quad \text{.....4}$$

Similarly, we can obtain the x interval Δx corresponding to a specified Δy as

$$\Delta x = \Delta y / m \quad \text{.....5}$$

For lines with slope magnitudes |m| < 1, Δx can be set proportional to a small horizontal deflection voltage, and the corresponding vertical deflection is then set proportional to Δy as calculated from Eq-4.

For lines whose slopes have magnitudes |m| > 1, Δy can be set proportional to a small vertical deflection voltage with the corresponding horizontal deflection voltage set proportional to Δx, calculated from Eq.5.

For lines with m = 1, Δx = Δy and the horizontal and vertical deflections voltages are equal. In each case, a smooth line with slope m is generated between the specified endpoints.

On raster system, lines are plotted with pixels, and step sizes in the horizontal and vertical directions are constrained by pixel separations.

Scan conversion process for straight lines is illustrated in Fig 2.3.

DDA Algorithm

The digital differential analyzer (DDA) is a scan-conversion line algorithm based on calculating either Δy or Δx, using Eq. 4 or Eq. 5.

First consider a line with positive slope, as shown in Fig. If the slope is less than or equal to 1, sample at unit x intervals (Δx = 1) and compute successive y values as

$$y_{k+1} = y_k + m \quad \text{.....6}$$

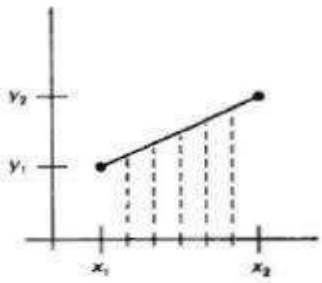


Fig 2.4 straight line segment with five sampling positions along the x axis between x1 and x2.

Subscript k takes integer values starting from 0, for the first point, and increases by 1 until the final endpoint is reached.

Since m can be any real number between 0.0 and 1.0, each calculated y value must be rounded to the nearest integer corresponding to a screen pixel position in the x column.

For lines with a positive slope greater than 1.0, reverse the roles of x and y. That is, we sample at unit y intervals (Δy = 1) and calculate consecutive x values as

$$x_{k+1} = x_k + 1/m \quad \text{.....7}$$

In this case, each computed x value is rounded to the nearest pixel position along the current y scan line.

Equations 6 and 7 are based on the assumption that lines are to be processed from the left endpoint to the right endpoint Fig 2.2. If this processing is reversed, so that the starting endpoint is at the right, then either we have $\Delta x = -1$ and

$$y_{k+1} = y_k - m \quad \ldots\ldots 8$$

or (when the slope is greater than 1) we have $\Delta y = -1$ with

$$x_{k+1} = x_k - 1/m \quad \ldots\ldots 9$$

Negative slopes are calculated using Eq-s 6 through 9. If the absolute value of the slope is less than 1 and the starting endpoint is at the left, we set $\Delta x = 1$ and calculate y values with Eq-6.

When the starting endpoint is at the right (for the same slope), we set $\Delta x = -1$ and obtain y positions using Eq. 8.

For a negative slope with absolute value greater than 1, we use $\Delta y = -1$ and Eq. 9 or we use $\Delta y = 1$ and Eq.7.

Algorithm

```
#define ROUND (a) ((int) (a+0.5))
void lineDDA (int xa, int ya, int xb, int yb)
{
int dx = xb - xa, dy = yb - ya, steps, k;
float xIncrement, yIncrement, x = xa, y = ya;
if (abs (dx) > abs (dy) steps = abs (dx) ;
 else steps = abs dy);
xIncrement = dx / (float) steps;
yIncrement = dy / (float) steps;
setpixel (ROUND(x), ROUND(y) );
for(k=0;k<steps;k++)
{
x += xIncrement'
y += yIncrement;
setpixel((Round(x),Round (y))
```

Algorithm Description

Step 1: Accept Input as two endpoint pixel positions (xa, ya), (xb, yb)

Step 2: Horizontal and vertical differences between the endpoint positions are assigned to parameters dx and dy (Calculate dx = xb-xa and dy = yb-ya).

Step 3: The difference with the greater magnitude determines the value of parameter steps.

Step 4: Starting with pixel position (xa, ya), determine the offset needed at each step to generate the next pixel position along the line path.

Step 5: loop the following process for steps number of times

1. Use a unit of increment or decrement in the x and y direction
2. if xa is less than xb the values of increment in the x and y directions are 1 and m
3. if xa is greater than xb then the decrements -1 and – m are used.

Example: Consider the line from (0, 0) to (4, 6)

1. xa = 0, ya = 0 and xb = 4 yb = 6
2. dx = xb - xa = 4-0 = 4 and dy = yb - ya = 6-0 = 6
3. x = 0 and y = 0
4. 4 > 6 (false) so, steps = 6
5. Calculate xIncrement = dx/steps = 4 / 6 = 0.66 and yIncrement = dy/steps = 6/6 = 1
6. Setpixel(x, y) = Setpixel(0, 0) (Starting Pixel Position)
7. Iterate the calculation for xIncrement and yIncrement for steps (6) number of times
8. Tabulation of the each iteration is given below.

k	x	Y	Plotting points (Rounded to Integer)
0	0+0.66=0.66	0+1=1	(1,1)
1	0.66+0.66=1.32	1+1=2	(1,2)
2	1.32+0.66=1.98	2+1=3	(2,3)
3	1.98+0.66=2.64	3+1=4	(3,4)
4	2.64+0.66=3.3	4+1=5	(3,5)
5	3.3+0.66=3.96	5+1=6	(4,6)

Resultant Diagram for the above tabulation:

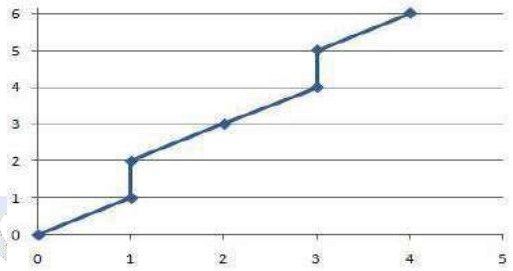


Fig 2.5 Pixel positions along the line path between endpoints (0, 0) and (4, 6) plotted with DDA line algorithm

Advantages of DDA Algorithm

1. It is the simplest algorithm
2. It is a is a faster method for calculating pixel positions

Disadvantages of DDA Algorithm

1. Floating point arithmetic in DDA algorithm is still time-consuming
2. End point accuracy is poor

Bresenham's Line Algorithm

- An accurate and efficient raster line-generating algorithm developed by Bresenham that uses only incremental integer calculations.
- In addition,Bresenham's line algorithm can be adapted to display circles and other curves.
- To illustrate Bresenham's approach, we first consider the scan-conversion process for lines with positive slope less than 1.0.
- Pixel positions along a line path are determined by sampling at unit x intervals. Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive column (x position) and plot the pixel whose scan-line y value is closest to the line path.

Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, next we need to decide which pixel to plot in column x_{k+1}. Our choices are the pixels at positions $(x_k + 1, y_k)$ and $(x_k + 1, y_k + 1)$.

For example, is shown in the following figure *2.4*. From position (2, 3) we need to determine at next sample position is whether (3, 3) or (3, 4). We choose the point which is closer to the original line.

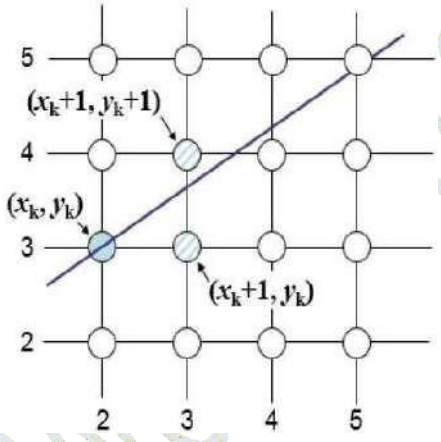


Fig 2.6 a straight line segment is to be plotted, starting from the pixel at column 2 on scan line 3.

At sampling position $x_k + 1$, we label vertical pixel separations from the mathematical line path as d_{lower} and d_{upper} in Fig 2.5.

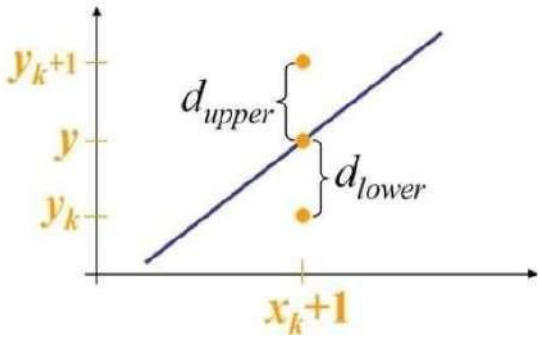


Fig 2.7 Distance between pixel position

The y coordinate on the mathematical line at pixel column position $x_k + 1$ is calculated as

$$y = m(x_k+1) + b \text{ ---------- } 10$$

Then

$$d_{lower} = y - y_k$$

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

And

$$d_{upper} = (y_k +1) - y$$

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

The difference between these two separations is

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1 \text{ ----------11}$$

A decision parameter pk for the kth step in the line algorithm can be obtained by rearranging Eq. 11

By substituting $m = \Delta x / \Delta y$ we get

$$P_k = \Delta x (d_{lower} - d_{upper})$$

$$= 2\Delta y.x_k - 2\Delta x.y_k + c \text{ ----------12}$$

The sign of p_k is the same as the sign of $d_{lower} - d_{upper}$.

At step k + 1, the decision parameter is evaluated from Eq. 12 as

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

Subtracting Eq. 12 from the preceding equation, we have

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

But $x_{k+1} = x_k + 1$, so that

$$p_{k+1} = p_k + 2\,\Delta y - 2\,\Delta x(y_{k+1} - y_k) \text{ ---------- } 13$$

Where the term $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter p_k .

- This recursive calculation of decision parameters is performed at each integer x position, starting at the left coordinate endpoint of the line.
- The first parameter, p_0, is evaluated from Eq.12 at the starting pixel position (x_0, y_0) and with m evaluated as $\Delta y / \Delta x$:

$$p_0 = 2\Delta y - \Delta x \text{ ---------- } 14$$

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

1. Input the two line endpoints and store the left endpoint in (x_0, y_0)
2. Set the color for frame-buffer position (x_0, y_0); i.e., plot the first point.
3. Calculate the constants Δx, Δy, $2\Delta y$, and $2\Delta y - 2\Delta x$, and obtain the starting value for the decision parameter as

$$\mathbf{p_0 = 2\Delta y - \Delta x}$$

4. At each x_k along the line, starting at k = 0, perform the following test. If $p_k < 0$, the next point to plot is (x_{k+1}, y_k) and

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

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

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

6. Perform step 4 $\Delta x - 1$ times.

Implementation of Bresenham Line drawing Algorithm

```
void lineBres (int xa, int ya, int xb, int yb)
{
        int dx = abs( xa – xb) , dy = abs (ya - yb);
        int p = 2 * dy – dx;
        int twoDy = 2 * dy, twoDyDx = 2 *(dy - dx);
        int x , y, xEnd;      /* determine which point to use as
                                    start, which as end */
 if (xa > x b )
{
        x = xb;
        y = yb;
        xEnd = xa;
 }
 else
{
        x = xa;
        y = ya;
        xEnd = xb;
}
setPixel(x, y);
while(x < xEnd)
{
        x++;
        if (p < 0)
        p+ = twoDy;
else
 {      y++;
        p+ = twoDyDx;
 }
setPixel(x,y);
}
 }
```

Example:

Consider the line with endpoints (20, 10) to (30, 18)

The line has the slope m = (18 - 10)/ (30 - 20) = 8/10 = 0.8

$\Delta x = 10$

$\Delta y = 8$

The initial decision parameter has the value $p0 = 2\Delta y - \Delta x = 6$

and the increments for calculating successive decision parameters are

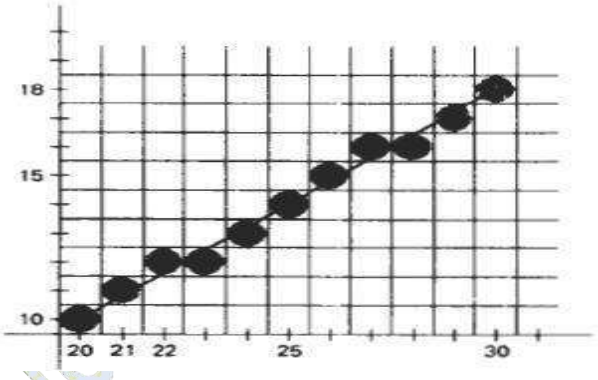
$2\Delta y = 16$

$2\Delta y - 2\Delta x = -4$

We plot the initial point (x0, y0) = (20, 10) and determine successive pixel positions along the line path from the decision parameter.

Tabulation:

k	p_k	(x_k+1, y_K+1)
0	6	(21,11)
1	2	(22,12)
2	-2	(23,12)
3	14	(24,13)
4	10	(25,14)
5	6	(26,15)
6	2	(27,16)
7	-2	(28,16)
8	14	(29,17)
9	10	(30,18)



2.8 Pixel positions along the line path between endpoints (20, 10) and (30, 18) plotted with Bresenham's line algorithm

Advantages

1. Algorithm is fast
2. Uses only integer calculations

Disadvantages

1. It is meant only for basic line drawing.

LOADING THE FRAME BUFFER

- After scan converting the straight line segments and other objects in the raster system, frame buffer positions must be calculated.

- It is done by set pixel procedure that stores intensity values for the pixels at corresponding addresses within the frame buffer array.
- Scan conversion algorithms generate pixel positions at successive intervals.
- To calculate frame-buffer addresses, incremental methods are used.

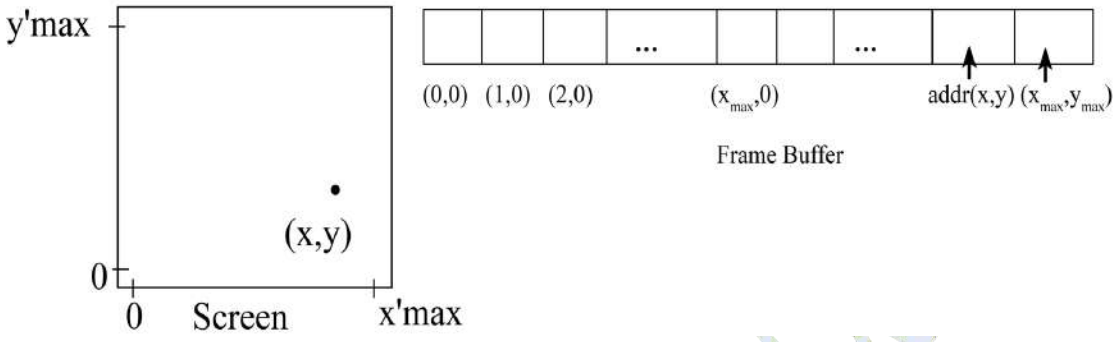


Figure 2.9 Pixel screen positions stored within the frame buffer

- For example, in the figure 2.9, the frame buffer array is addressed in row major order. The pixel positions vary from (0,0) to (x_{max}, y_{max}).
- The pixel postion (x,y) is calculated as follows:

  addr (x,y) = addr (0,0) + y(x_{max}+1) + x
- We can calculate the frame buffer address for the next pixel logo in the scanline by using incremental method as follows:

  addr (x+1,y) = addr (x, y) +1
- For calculating the frame buffer address for the pixel position (x+1, y+1).

  addr (x+1,y+1) = addr (x,y) + x_{max}+2
- Where the constant x_{max} +2 is precomputed once for all line segments.

CIRCLE GENERATING ALGORITHMS

PROPERTIES OF CIRCLE

- A circle is defined as the set of points that are all at a given distance r from a center position (xc, yc).

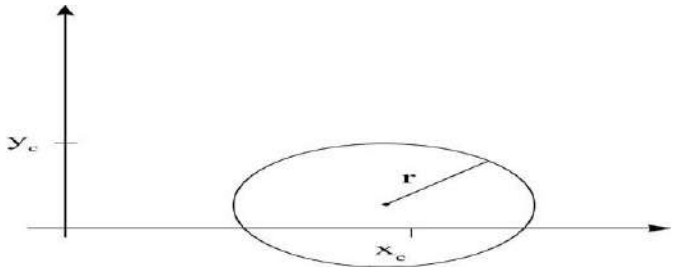


Figure 2.10 Circle with Center Coordinate (x_c, y_c) and Radius r

The distance relationship is expressed by the Pythagorean Theorem in Cartesian coordinates as follows

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

y values at each position is calculated as

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

and the x axis steps from $x_c - r$ to $x_c + r$. This method is not a best method for generating a circle.

Problems

(1) It involves considerable computation at each step.

(2) The spacing between each plotted pixel position is not uniform.

Solutions

(1) The spacing can be adjusted by interchanging x and y whenever the absolute value of the slope of the circle is greater than 1.

- It increases the computation and processing of the algorithm.

(2) Another way to adjust the unequal spacing is to calculate points along the circular boundary using polar coordinates r and θ.

- The circle equation in parametric polar form yields the following pair of equations:

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

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

Where θ is a fixed angular step size.

By using the above equation, a circle is plotted with equally spaced points along the circumference.

Symmetry of a circle

•By considering the symmetry of a circle, computations can be reduced.

•The shape of the circle is similar in all the four quadrants.

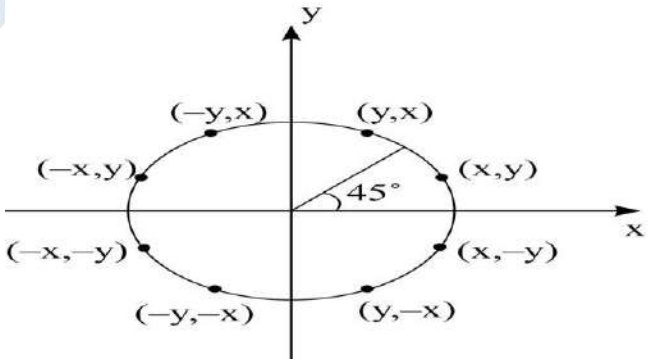


Figure 2.11: Symmetry of a circle

- There is symmetry between octants (shown in figure 2.11).

- Circle sections in adjacent octants within one quadrant are symmetric with respect to the 45° line dividing the two octants.

Advantage

- We can generate all pixel positions around a circle by calculating only the points within the sector from x=0 to x=y.

MIDPOINT CIRCLE ALOGITHM

- In midpoint method, the circle function is defined as follows:

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

- Any point (x,y) on the boundary of the circle with radius r or satisfies the following equation.

  $f_{circle}(x, y) = 0$

- If the point is in the interior of the circle, the circle function is negative.
- If the point is outside the circle, the circle function is positive.
- The relative position of any point (x,y) can be determined by checking the sign of the circle function as follows:

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

- The circle function test is performed for the mid positions between pixels near the circle path at each sampling step. So the circle function is the decision parameter in the midpoint algorithm.

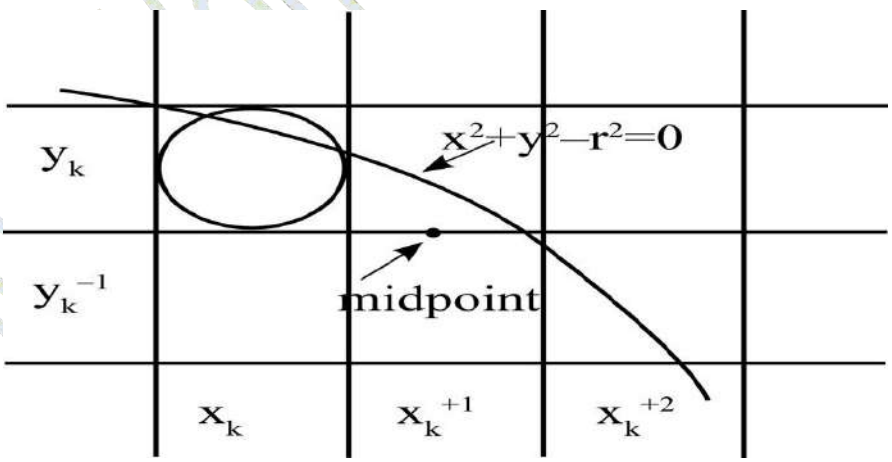


Figure 2.12 Mid Point between Candidate Pixels

The figure 2.10 shows the midpoint between the two candidate pixels at sampling position x_{k+1}.

It we plot the pixel at (x_k, y_k), we have to find whether the next pixel position is (x_k+1, y_k) or (x_k+1, y_k-1).

It can be decided by evaluating the circle function

$$p_K = f_{circle}(x_k+1, y_k-1/2)$$

Apply it on the circle function

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

$$P_K = (x_k+1)^2 + (y_k-1/2)^2 - r^2 \quad \rightarrow (1)$$

If $P_k<0$, this midpoint is inside the circle and the pixel on scanline y_k is closer to the circle boundary.

Otherwise, the midposition is outside (or) on the circle boundary, so we select the pixel on scanline y_k-1.

Successive decision parameters are obtained by incremental calculations.

The next decision parameter is obtained by evaluating the circle function at sampling position $\mathbf{x_{k+1}+1 = x_k+2}$.

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

$$= [(x_k+1)+1]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = [(x_k+1)+1]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$$

$$p_{k+1} - p_k = (2) - (1)$$

$$= \left[(x_k+1)+1\right]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2 - \left[(x_k+1)^2 + (y_k-1/2)^2 - r^2\right]$$

$$= [(x_k+1)+1]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - \cancel{r^2} - (x_k+1)^2 - \left(y_k-\tfrac{1}{2}\right)^2 + \cancel{r^2}$$

$$= \left[\cancel{(x_k+1)^2} + (2x_k+1) + 1 + y_{k+1}^2 - y_{k+1} + \tfrac{1}{4} - \cancel{(x_k+1)^2} - \left(y_k^2 - y_k + \tfrac{1}{4}\right)\right]$$

$$= (2x_k+1) + 1 + y_{k+1}^2 - y_{k+1} + \cancel{\tfrac{1}{4}} - y_k^2 + y_k - \cancel{\tfrac{1}{4}}$$

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

Rearrange the terms as folows

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

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

where y_{k+1} is either y_k or y_{k-1}, depending on the sign of p_k

- Increments for ontaining P_{k+1} are either

  $2x_{k+1} + 1$ (i.e) $2x_k + 2$ (if p_k is negative)

  or

  $2x_{k+1} + 1 - 2y_{k+1}$

- The terms $2x_{k+1}$ and $2y_{k+1}$ are evaluated incrementally as follows

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

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

- The start position (0, r) with the value (0, 2r).
- Successive values are obtained by adding 2 to the previous value of 2x, and subtracting 2 from the previous value of 2y.

Evaluation of Initial Decision Parameter

- State position $(x_o, y_o) = (0, r)$

$$p_o = f_{circle}\left(1, r - \frac{1}{2}\right)$$

$$= 1^2 + \left(r - \frac{1}{2}\right)^2 - r^2$$

$$= 1 + \left(r^2 - r + \frac{1}{4}\right) - r^2$$

$$= 1 - r + \frac{1}{4}$$

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

Hence, the radius r is specified as an integer.

Round of p_o to

$$p_o = 1 - r$$

where r is an integer and all increments are integers.

Midpoint Circle Algorithm

Step 1: Read the values of radius r and circle center (x_c, y_c).

Obtain the first point on the circumference of a circle.

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

Step 2: Calculate the initial value of the decision parameter.

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

Step 3: At each x_k position, perform the following test:

initialize k=0

If pk<0, the next point along the circle centerd on (0,0) is (x_{k+1}, y_k),

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

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

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

where

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

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

Step 4: Find out the symmetry points in the other seven octants.

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

$x = x + x_c$

$y = y + y_c$

Step 6: Repeat the steps 3 to 5 until $x \geq y$.

Procedure:

The following procedure implements midpoint circle drawing algorithm.

```
procedure midpointCircle (xc,yc, r: integer);
var
   p,x,y : integer;
   procedure plotpix;
   begin
            setPixel (xc+x, yc + y, 1);
            setPixel (xc—x, yc + y, 1);
            setPixel (xc+x, yc — y, 1);
            setPixel(xc—x, yc — y, 1);
             setPixel (xc—y, yc+x, 1);
             setPixel (xc+y, yc—x, 1);
             setPixel (xc—y, yc—x,1);
         end; plotpix
begin
         x:=0;
         y:=r;
         plotpix;
         p: = 1—r;
         while x<y do
             begin
                 if p<0 then
                 x: = x+1
                 else
                     begin
                         x:=x+1;
                             y:=y—1;
                     end;
               if p<0 then
                 p:= p+2*x+1
               else
                 p:=p+2*(x—y)+1
               plotpix
               end;
  end;{midpoint cirlce}
```

ATTRIBUTES OF OUTPUT PRIMITIVES

Any parameter that affects the way a primitive is to be displayed is referred to as an attribute parameter. Attribute parameters are color, size etc. It is used to determine the fundamental characteristics of a primitive.

TYPES OF ATTRIBUTES

1. Line Attributes
2. Curve Attributes
3. Color and Grayscale Levels
4. Area Fill Attributes
5. Character Attributes
6. Bundled Attributes

Line Attributes

Basic attributes of a straight line segment are

1. Line Type
2. Line Width
3. Pen and Brush Options
4. Line Color

Line type

Line type attribute includes solid lines, dashed lines and dotted lines.

To set line type attributes in a **PHIGS** application program, a user invokes the function

setLinetype (lt)

where parameter *lt* is assigned a positive integer value of 1, 2, 3 or 4 to generate lines that are solid, dashed, dash dotted respectively. Other values for line type parameter it could be used to display variations in dot-dash patterns.

Line width

Implementation of line width option depends on the capabilities of the output device to set the line width attributes.

To set the line-width attributes using the following command

setLinewidthScaleFactor (lw)

- Line width parameter *lw* is assigned a positive number to indicate the relative width of line to be displayed.
- A value of 1 specifies a standard width line.
- To set *lw* to a value of 0.5 to plot a line whose width is half that of the standard line.
- Values greater than 1 produce lines thicker than the standard.

Line Cap

- To adjust the shape of the line ends to give them a better appearance by adding line cap (Fig: 2.8).
- There are three types of line cap. They are
    1. Butt cap
    2. Round cap
    3. Projecting square cap

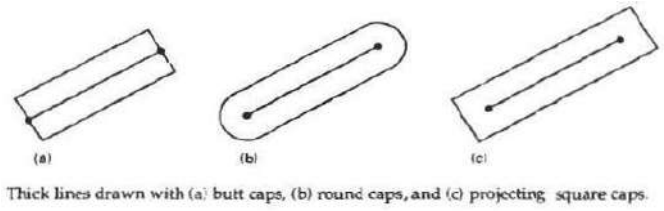


Fig 2.13 Types of Line Cap

Butt cap

It obtained by adjusting the end positions of the component parallel lines so that the thick line is displayed with square ends that are perpendicular to the line path.

Round cap

- It obtained by adding a filled semicircle to each butt cap.
- The circular arcs are centered on the line endpoints and have a diameter equal to the line thickness.

Projecting square cap

It extends the line and adds butt caps that are positioned one-half of the line width beyond the specified endpoints (Fig: 2.9)

There are three possible methods for smoothly joining two line segments,

1. Miter Join
2. Round Join
3. Bevel Join

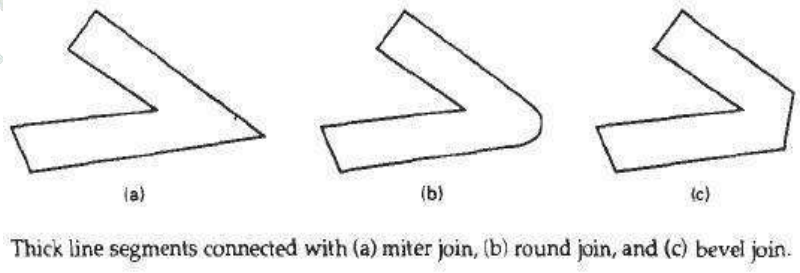


Fig 2.14 Types of Line Joining

Miter join

It is accomplished by extending the outer boundaries of each of the two lines until they meet.

Round join

It is produced by capping the connection between the two segments with a circular boundary whose diameter is equal to the width.

Bevel join

It is generated by displaying the line segment with but caps and filling in triangular gap where the segments meet.

Pen and Brush Options

- In some graphics packages, lines can also be displayed using selected pen or brush options.
- Options in this category include shape, size, and pattern. Some possible pen or brush shapes are given in following figure 2.10.

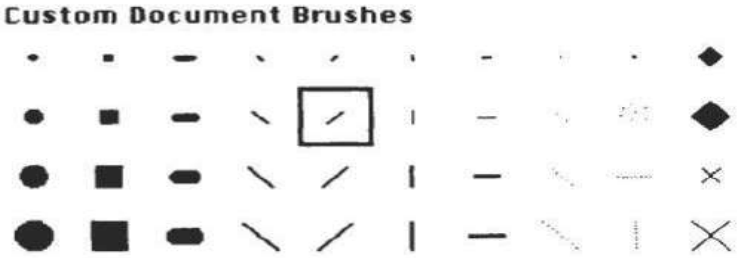


Fig 2.15 Various Pen and Brush Shapes

Line color

- A poly line routine displays a line in the current color by setting this color value in the frame buffer at pixel locations along the line path using the set pixel procedure.
- To set the line color value in PHlGS with the function

setPolylineColourIndex (lc)

Non negative integer values, corresponding to allowed color choices, are assigned to the line color parameter *lc.*

Example:

Various line attribute commands in an applications program is given by the following sequence of statements

```
setLinetype(2);
setLinewidthScaleFactor(2);
setPolylineColourIndex (5);
polyline(n1, wc points1);
setPolylineColorIindex(6);
poly line (n2, wc points2);
```

Area Fill Attributes

Options for filling a defined region include a choice between a solid color and a pattern fill and choices for particular colors and patterns. These fill options can be applied to polygon regions or to areas defined with curved boundaries depending on the capabilities of available package.

The areas can be displayed using various brush styles, colors and transparency parameters.

Fill Styles

Areas are displayed with three basic fill styles, are shown in Fig: 2.11.

1. Hollow with a color border
2. Filled with a solid color
3. Filled with a specified pattern or design.

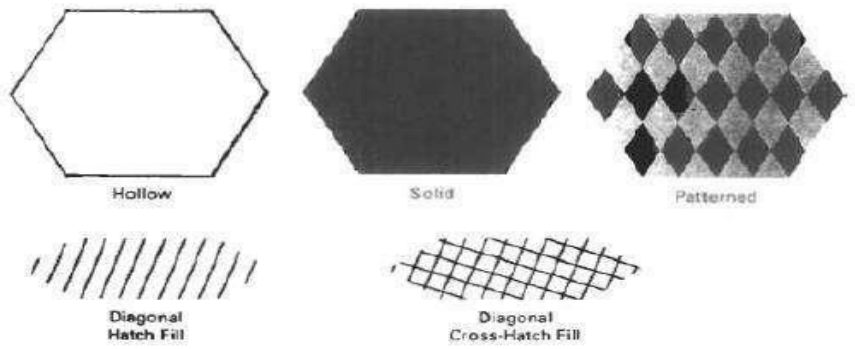


Fig 2.16 Polygon Fill styles

A basic fill style is selected in a PHIGS program with the function

setInteriorStyle (fs)

- Values for the fill-style parameter *fs* include hollow, solid, and pattern.
- Another value for fill style is hatch, which is used to fill an area with selected hatching patterns such as parallel lines or crossed lines.
- The color for a solid interior or for a hollow area outline is chosen with where fill color parameter *fc* is set to the desired color code

setInteriorColourIndex (fc)

where fill-color parameter *fc* is set to the desired color code. Some other fill options are used to specify the edge type, edge width and edge color of a region.

Pattern Fill

To select fill patterns with the following function

setInteriorStyleIndex (pi)

Where pattern index parameter *pi* specifies a table position

For example, the following set of statements would fill the area defined in the fillArea command with the second pattern type stored in the pattern table:

```
SetInteriorStyle( pattern);
SetInteriorStyleIndex(2);
Fill area (n, points);
```

Index (pi)	Pattern (cp)
1	$\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$
2	$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$

For fill style pattern, table entries can be created on individual output devices with the following function

setPatternRepresentation (ws,pi,nx,ny,cp)

Parameter pi sets the pattern index number for workstation code *ws*, and *cp* is a two dimensional array of color codes with *nx* columns and *ny* rows. For example the following function could be used to set the first entry in the pattern table for workstation 1.

```
cp [1,1 ]   =  4;          cp [2,2]    =  4;
cp [1,2]    =  0;          cp[2,1]     =  0;
```

setPatternRepresentation (1,1,2,2,cp);

When color array cp is to be applied to fill a region, we need to specify the size of an array with the following function

setPatternSize(dx,dy)

Where parameters *dx* and *dy* give the coordinate width and height of the array mapping. Then a reference position for starting a pattern fill is assigned with the following statement;

setPatternReferencePoint (position);

Where parameter position is a pointer to coordinates (xp,yp) that fix the lower left corner of the rectangular pattern.

Tiling:

The process of filling an area with a rectangular pattern is called tiling and it is also referred to as tiling patterns.

Soft Fill:

Soft fill or tint fill algorithms are applied to repaint areas so that the fill color is combined with background color. An example of this type of fill is *linear soft fill* algorithm repaints an area by merging a fore ground color F with a single background color B, Where F is not equal B.

Character Attributes

- The appearance of displayed character is controlled by attributes such as font, size, color and orientation.

- Attributes can be set both for entire character strings (text) and for individual characters defined as marker symbols.

Text Attributes

- The choice of font or type face is set of characters with a particular design style as courier, Helvetica, times roman, and various symbol groups.
- The characters in a selected font also are displayed with styles (solid, dotted, double) in bold face in italics and in outline or shadow styles.

A particular font and associated style is selected in a PHIGS program by setting an integer code for the text font parameter *tf* in the function

setTextFont (tf)

Control of text color (or intensity) is managed from an application program with

setTextColourIndex (tc)

Where text color parameter *tc* specifies an allowable color code.

We can adjust text size by scaling the overall dimensions of characters or by scaling only the character width.

Character size is specified by points, where 1 point is 0.013837 inch. Point measurements specify the size of the body of a character.

The distance between the bottom-line and the top line of the character body is same for all characters in particular size and typeface, but width of the body may vary.

Proportionally spaced fonts assign a smaller body width to narrow characters such as i, j, l and f compared to broad characters such as W or M.

Character height is defined as the distance between the base line and cap line of characters.

Text size can be adjusted without changing the width to height ratio of characters with

setCharacterHeight (ch)

Parameter ch is assigned a real value greater than 0 to set the coordinate height of capital letters.

Height 1
Height 2
Height 3

The width of text can be set with function.

setCharacterExpansionFactor (cw)

Where the character width parameter cw is set to a positive real value that scales the body width of character.

Width 0.5

Width 1.0

Width 2.0

Spacing between characters is controlled separately with

setCharacterSpacing (cs)

Spacing

S p a c i n g

S p a c i n g

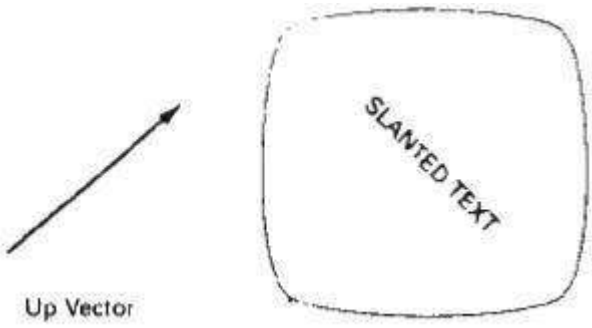
Where the character-spacing parameter *cs* can he assigned any real value.

The orientation for a displayed character string is set according to the direction of the character *up vector*

setCharacterUpVector (upvect)

Parameter *upvect* in this function is assigned two values that specify the x and y vector components.

Text is displayed so that the orientation of characters from base line to cap line is in the direction of the up vector. For example, *upvect* = (1, 1) is displayed the text in 45^0 as shown in the following figure.



To arrange character strings vertically or horizontally

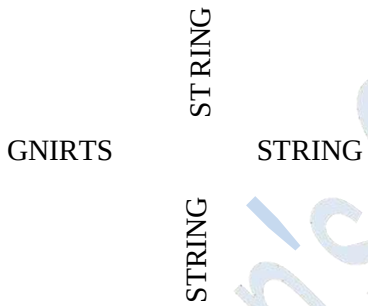
setTextPath (tp)

tp can be assigned the value: right, left, up, or down

Another attribute for character strings is alignment. This attribute specifies how text is to be positioned with respect to the start coordinates. Alignment attributes are set with

SetTextAlignment (h,v)

Where parameters h and v control horizontal and vertical alignment.



Horizontal alignment is set by assigning h a value of left, center, or right.

Vertical alignment is set by assigning v a value of top, cap, half, base or bottom.

A precision specification for text display is given with

SetTextPrecision (tpr)

tpr is assigned one of values string, char or stroke.

Marker Attributes

Marker symbol is a single character that can he displayed in different colors and in different sizes. To select a particular character to be the marker symbol with

setMarkerType (mt)

Where marker type parameter mt is set to an integer code

Typical codes for marker type are the integers 1 through 5, specifying, respectively, a dot (.) a vertical cross (+), an asterisk (*), a circle (o), and a diagonal cross (X).

To set the marker size with

setMarkerSizeScaleFactor (ms)

With parameter marker size *ms* assigned a positive number. This scaling parameter is applied to the nominal size for the particular marker symbol.

Values greater than 1 increase the marker size and values less than one reduce the marker size.

Marker color is specified with

SetPolymarkerColourIndex (mc)

Selected color code parameter *mc* is stored in the current attribute list and used to display subsequently specified marker primitives

Bundled Attributes

A single attribute that specifies exactly how a primitive is to be displayed with that attribute setting. These specifications are called individual or unbundled attributes.

A particular set of attributes values for a primitive on each output device is chosen by specifying appropriate table index. Attributes specified in this manner are called bundled attributes.

The table for each primitive that defines groups of attribute values to be used on particular output devices is called a *bundle table.*

The choice between a bundled or an unbundled specification is made by setting a switch called the *aspect source flag* for each of these attributes

setIndividualASF(attributeptr, flagptr)

Where parameter *attributerptr* points to a list of attributes and parameter *flagptr* points to the corresponding list of aspect source flags.

Each aspect source flag can be assigned a value of individual or bundled.

Bundled line Attributes

Entries in the bundle table for line attributes on a specified workstation are set with the function

setPolylineRepresentation (ws, li, lt, lw, lc)

Parameter *ws* is the workstation identifier and line index parameter li defines the bundle table position.

Parameter *lt, lw, tc* are then bundled and assigned values to set the line type, line width, and line color specifications for designated table index.

Example

setPolylineRepresentation (1, 3, 2, 0.5, 1)
setPolylineRepresentation (4, 3, 1, 1, 7)

A poly line that is assigned a table index value of 3 would be displayed using dashed lines at half thickness in a blue color on work station 1; while on workstation 4, this same index generates solid, standard-sized white lines.

Once the bundled tables have been set up, a group of bundled line attributes is chosen for each workstation by specifying table index value;

setPolylineIndex (li);

Bundled Area fills Attributes

Table entries for bundled area-fill attributes are set with

setInteriorRepresentation (ws, fi, fs, pi, fc)

Which defines the attributes list corresponding to fill index fi on workstation ws.

- Parameter fs, pi and fc are assigned values for the fill style, pattern index and fill color respectively.
- A particular attribute bundle is selected from the table with the function

setInteriorIndex (fi);

Bundled Text Attributes

Table entries for bundled text attributes are set with

setTextRepresentation (ws, ti, tf, tp, te, ts, tc)

Bundles values for text font, precision,expansion factor, size and color in a table position for work station ws that is specified by value assigned to text index parameter ti.

A particular text index value is chosen with the function

setTextIndex (ti);

Bundled Marker Attributes

Table entries for bundled marker attributes are set with

setPolymarkerRepresentation (ws, mi, mt, ms, mc)

That defines marker type, marker scale factor, marker color for index mi on workstation ws. Bundle table selections are made with the function

setPolymarkerIndex (mi);

COLOUR AND GRAYSCALE LEVELS

- Colour options are numerically coded with values ranging from 0 through the positive integers. These color codes are converted to intensity level settings for the electron beams in CRT monitors.
- Color Tables
- Color information can be stored in the frame buffer in two ways :
    - The colour codes can be directly put in the frame buffer (or)
    - Colour codes can be maintained in a separate table and pixel values can be used as an index into this table.

Color code	Stored Color Values in Frame buffer			Displayed color
	RED	GREEN	BLUE	
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White

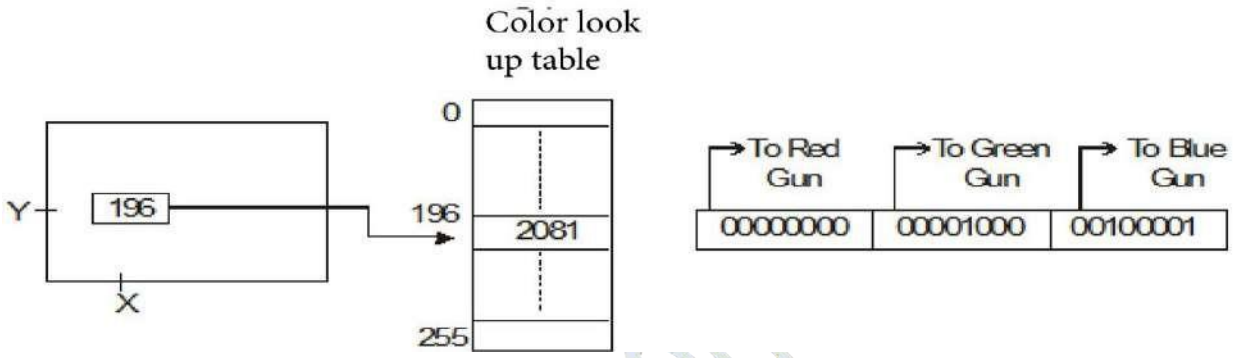


Figure.2.17: Colour look-up table

Advantages of storing colour codes in lookup table

(1) Colour table can provide a reasonable number of simultaneous colours without requiring large frame buffers.

(2) Table entries can be changed at anytime, and allows the user to experiment easily with different color combinations in a design, scene or graph without changing the attribute settings for the graphics data structure.

(3) Visualization applications can store values for some physical quantity such as energy in the frame buffer.

(4) Use lookup table to get various color encodings without changing the pixel values.

(5) In visualization and image processing applications, color tables are used for setting color thresholds so that all pixel values above or below a specified threshold can be set the same colour.

Grayscale

- With monitors that have no color capability, color functions can be used in an application program to set the shades of gray, or grayscale for the displayed primitives.
- Numeric values from 0 to 1, can be used to specify grayscale levels, which are converted to appropriate binary codes for storage in the raster.
- The table given below shows the specification for intensity codes for a four level grayscale system.

Intensity Codes	Stored Intensity in the Frame buffer	Values (Binary code)	Displayed Grayseale
0.0	0	(00)	Black
0.33	1	(01)	Darkgray
0.67	2	(10)	Light Gray
1.0	3	(11)	White

- Intensity is calculated based on the colour index as follows:
  Intensity = 0.5 [min (r,g,b) + max (r,g,b)]

INQUIRY FUNCTION

- Inquiry functions are used to retrieve the current settings of attributes and other parameters such as workstation types and status from the system lists.
- By using inquiry function, current values of any specified parameter can be saved and then they can be reused later or they can be used to check the current state of the system if any error encounters.
- Current attribute values are checked by specifying the name of the attribute in the inquiry function as follows

inquirePolylineIndex (last li)

To copy the current values of attributes

inquireInteriorColorIndex (last_fc)

- The above function the current values of line index and fill color into parameters last **last li** and **lastfc.**

3. TWO DIMENSIONAL GEOMETRIC TRANSFORMATIONS

Definition

Transformation is a basic concept in computer graphics. It means to alter the orientation,size, and shape of an object with geometric transformation in a 2-D plane.

BASIC TRANSFORMATIONS

There are three basic transformations they are

1. Translation
2. Rotation
3. Scaling

Translation

A translation is applied to an object by representing it along a straight line path from one coordinate location to another. It is also refers to the shifting of a point or move an object from one place to some other place.

In order to move an object in 2-D space, we need to add or subtract some value from its x and y coordinates. That distance is known as '**translational distance**'.

By adding *translation distances*, tx and ty to the original coordinate position (x,y) to move the point to a new position (**x'**, **y'**)

$$x' = x + tx, \quad y' = y + ty$$

The translation distance point (tx, ty) is called *translation vector or shift vector*.

Translation equation can be expressed as single matrix equation by using column vectors to represent the coordinate position and the translation vector as

$$P = \begin{bmatrix} x1 \\ x2 \end{bmatrix}, \quad P' = \begin{bmatrix} x1' \\ x2' \end{bmatrix}, \quad T = \begin{bmatrix} tx \\ ty \end{bmatrix}$$

2-D translation equation in matrix form can be written as

$$P' = P + T$$

Sometimes matrix transformation equations are expressed in terms of coordinate row vectors instead of column vector such representation as P =[x y] and T = [t_x t_y].

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

Similar methods are used to translate curved objects. To change the position of a circle or ellipse, we translate center coordinates and redraw the figure in new location.

Rotation

Rotation is used to rotate a point about an axis. The axis can be any of the coordinates or simply any other specified line also.

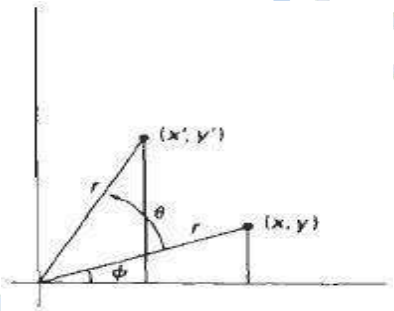
A two-dimensional rotation is applied to an object by repositioning it along a circular path in the *xy* plane.

To generate a rotation, specify a rotation angle θ and the position (x_r, y_r) of the rotation point (or pivot point).

Positive values for the rotation angle define counter clock wise rotation about pivot point. A negative value rotates objects in clock wise direction.

The transformation can also be described as a rotation about a **rotation axis** perpendicular to *xy* plane and passes through pivot point.

We first determine the transformation equations for rotation of a point position P when the pivot point is at coordinate origin. The angular and coordinate relationships of the original and transformed point positions are shown in the figure 3.3



Where r is the constant distance of the point from the origin, angle Φ is the original angular position of the point θ is the rotation angle.

Rotation of a point from position (x, y) to position (x', y') through angle θ relative to coordinate origin. The transformed coordinates in terms of angle θ and Φ as

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

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

The original coordinates of the point in polar coordinate are

$$x = r\cos\Phi, \qquad y = r\sin\Phi$$

Substituting expression 2 into 1, we obtain the transformation equation for rotating a point at position (x,y) through an angle θ about origin

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

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

We can write Rotation Equation in the matrix form:

$$P' = R.\ P$$

Where the Rotation Matrix are

$$\mathbf{R} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

When coordinate positions are represented as row vectors instead of column vectors, the matrix product in rotation euation is transposed so that transformed row coordinate vector is [**x' y'**] calculated as

$$P'^{T} = (R.\ P)^{T}$$

$$= P^{T}.\ R^{T}$$

Where P^T= [x y], and the transpose R^T of matrix R is obtained by interchanging rows and columns. For a rotation matrix, the transpose is obtained by simply changing the sign of the sine terms. The transformation equations for rotation of a point about any specified rotation position (x_r,y_r) :

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

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

As with translation, rotations are rigid body transformations that move objects without deformation. Every point on an object is rotated through the same angle.

Scaling

It is the concept of increasing (or decreasing) the size of a picture (in one or in either directions). 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 factor S_x & S_y to produce the transformed coordinates (x', y')

$$\mathbf{x' = x\ .\ S_x} \qquad \mathbf{y' = y\ .\ S_y}$$

Scaling factor S_x scales object in x direction while S_y scales in y direction. The transformation equation in matrix form

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

(or)

$$\mathbf{P' = S.\ P}$$

Where S is 2 by 2 scaling matrix

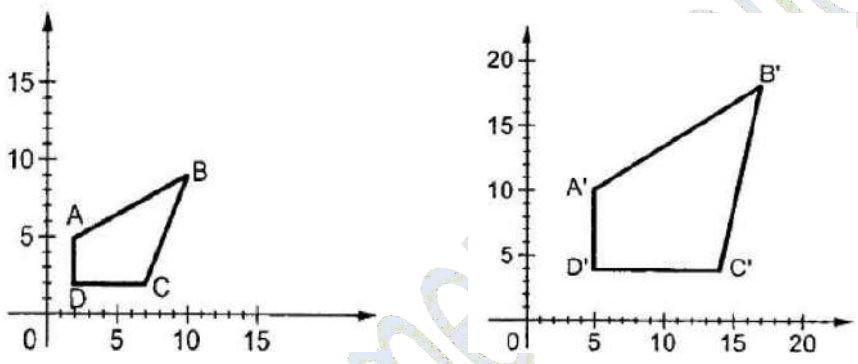
Turning a square (a) Into a rectangle (b) with scaling factors sx = 2 and sy = 1.

Any positive numeric values are valid for scaling factors s_x and s_y. If values less than 1 reduce the size of the objects as well as it moves the objects closer to the coordinate origin, while values greater than 1 produce an enlarged object and it moves coordinate positions farther from the origin.

There are two types of scaling such as

1. Uniform scaling
2. Non Uniform Scaling or differential scaling

To get uniform scaling it is necessary to assign same value for s_x and s_y. Unequal values for sx and sy result in a non uniform scaling.



The location of a scaled object by choosing a specified position is called *fixed point* scaling that is to remain unchanged after the scaling transformation. The Coordinates for fixed point (x_f, y_f) can be chosen as one of the vertices, the object centroid, or any other position. For a vertex with coordinates (x,y) ,the scaled coordinates (x' ,y') are calculated as;

$$\mathbf{x' = x_f + (x - x_f)\, s_x}$$

$$\mathbf{y' = y_f + (y - y_f)\, s_y}$$

To separate the multiplicative and additive terms in the above scaling transformation as,

$$\mathbf{x' = x \,.\, s_x + x_f\,(1 - s_x)}$$

$$\mathbf{y' = y \,.\, s_y + y_f\,(1 - s_y)}$$

Where $x_f\,(1 - s_x)$ and $y_f\,(1 - s_y)$ are constant for all points in the objects

MATRIX REPRESENTATION AND HOMOGENEOUS COORDINATES

Many graphics applications involve sequences of geometric transformations. For example an animation requires an object to be translated and rotated at each increment of the motion. In order to combine sequence of transformations we have to eliminate the matrix addition.

The basic transformation can be expressed in the general matrix form

$$\mathbf{P' = M_1 \,.P + M_2}$$

With coordinate positions p and p' represented as column vectors. Matrix M_1 is 2 by 2 array containing multiplicative factors, and M_2 is a two element column matrix containing translation terms.

To produce a sequence of transformations, first the coordinates are scaled then these scaled coordinates are rotated, finally rotated coordinates are translated.

To combine the multiplicative and additive terms for two dimensional transformations into a single matrix representation by represent matrix as 3 X 3 instead of 2 X 2 introducing an additional dummy coordinate h. Here points are specified by three numbers instead of two.

This coordinate system is called as *Homogeneous coordinate system* and it allows expressing transformation equation as matrix multiplication.

Cartesian coordinate (x, y) is represented as homogeneous coordinate triple (x_h, y_h, h) where

$$x = x_h / h$$

$$y = y_h / h$$

A general homogeneous coordinate representation can also be written as *(h.x,h.y,h)*. For two dimensional geometric transformations, choose homogeneous parameter h to be any non -zero value. For example, simply set h=1 so that the coordinates are represented by (x, y, 1).

For Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

The abbreviated form of the above matrix is,

$$\mathbf{P' = T (t_x,t_y) . P}$$

Where T (t_x,t_y) as the 3 by 3 translation matrix. The inverse of the transformation is obtained by replacing the translation parameters t_x and t_y with their negative $-t_x$ and $-t_y$..

For Rotation

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

Or as

$$\mathbf{P' = R (\theta). P}$$

where rotation transformation operator **R (θ)** is the 3 by 3 rotation matrix. The inverse of the rotation matrix when θ is replaced with – θ.

For Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Or as

$$\mathbf{P' = S (S_x, S_y). P}$$

Where S (S_x,S_y) is the 3 by 3 scaling matrix. Replacing these parameters with their multiplicative inverses (1/ s_x and 1/ s_y) yields the inverse scaling matrix.

COMPOSITE TRANSFORMATIONS

A composite transformation is a sequence of transformations; one followed by the other. In a matrix, any sequence of transformations as a *composite transformation matrix* by calculating the matrix product of the individual transformations.

For example, If a transformation of the plane T1 is followed by a second plane transformation T2, then the result represented by a single transformation T which is the composition of T1 and T2 is written as T = T1·T2.

Translation

If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P, the final transformed location P ' is calculated as

$$\mathbf{P' = T (tx2, ty2). \{T(tx1, ty1).P\}}$$
$$\mathbf{= \{T(tx2, ty2).T(tx1,ty1)\}.P}$$

Where P and P' are represented as homogeneous-coordinate column vectors.

$$\begin{bmatrix} 1 & 0 & tx2 \\ 0 & 1 & ty2 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & tx1 \\ 0 & 1 & ty1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx1+tx2 \\ 0 & 1 & ty1+ty2 \\ 0 & 0 & 1 \end{bmatrix}$$

(or)

$$\mathbf{T(tx2, ty2).T(tx1,ty1) = T(tx1+tx2, ty1+ty2)}$$

Which demonstrated the two successive translations are additive.

Rotations

Two successive rotations applied to point P produce the transformed position

$$P' = R(\theta 2).\ \{R(\theta 1).P\} = \{R(\theta 2).R(\theta 1)\}.P$$

By multiplying the two rotation matrices, we can verify that two successive rotation are additive

$$R(\theta 2).R(\theta 1) = R(\theta 1 + \theta 2)$$

So that the final rotated coordinates can be calculated with the composite rotation matrix as

$$P' = R(\theta 1 + \theta 2).P$$

Scaling

Concatenating transformation matrices for two successive scaling operations produces the following composite scaling matrix

$$\begin{bmatrix} sx2 & 0 & 0 \\ 0 & sy2 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} sx1 & 0 & 0 \\ 0 & sy1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx1.sx2 & 0 & 0 \\ 0 & sy1.sy2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Or

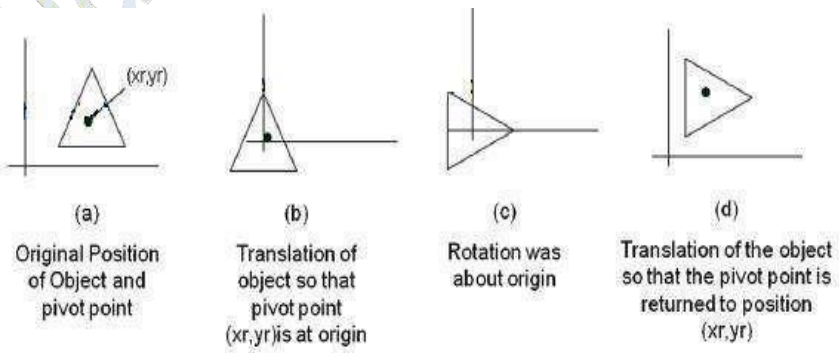
$$S(S_{x2}, S_{y2}) .S(S_{x1}, S_{y1}) = S(S_{x1}. S_{x2}, S_{y1}. S_{y2})$$

The resulting matrix indicates that successive scaling operations are multiplicative.

General Pivot-Point Rotation

To generate rotation about any selected pivot point (x_r ,y_r) by performing the following sequence of translate-rotate-translate operations;

1. Translate the object so that pivot-position is moved to the coordinate origin
2. Rotate the object about the coordinate origin
3. Translate the object so that the pivot point is returned to its original position



The composite transformation matrix for this sequence is obtain with the concatenation

$$\begin{bmatrix} 1 & 0 & xr \\ 0 & 1 & yr \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -xr \\ 0 & 1 & -yr \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} cos\theta & -sin\theta & xr(1-cos\theta) + yrsin\theta \\ sin\theta & cos\theta & yr(1-cos\theta) - xrsin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

which can also be expressed as

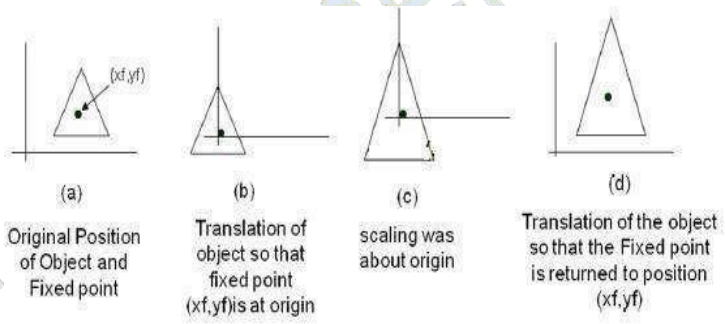
$$T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) = R(x_r, y_r, \theta).$$

Where $T(-x_r,-y_r) = T^{-1}(x_r,y_r)$

General fixed point Scaling

A transformation sequence produce scaling with respect to a selected fixed position (x_f,y_f) using a scaling function .

1. Translate object so that the fixed point coincides with the coordinate origin
2. Scale the object with respect to the coordinate origin
3. Use the inverse translation of step 1 to return the object to its original position



$$\begin{bmatrix} 1 & 0 & xf \\ 0 & 1 & yf \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -xf \\ 0 & 1 & -yf \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} sx & 0 & xf(1-sx) \\ 0 & sy & yf(1-sy) \\ 0 & 0 & 1 \end{bmatrix}$$

It can also be expressed as

$$T(x_f, y_f).S(s_x, s_y).T(-x_f, -y_f) = S(x_f, y_f, s_x, s_y)$$

OTHER TRANSFORMATIONS

Some additional transformations are

1. Reflection
2. Shear

Reflection

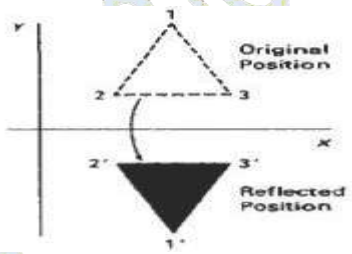
A reflection is a transformation that produces a mirror image of an object. The mirror image for a two-dimensional reflection is generated relative to an axis of reflection by rotating the object 180 about the reflection axis.

We can choose an axis of reflection in the *xy* plane or perpendicular to the *xy* plane or coordinate origin.

Reflection of an object about the **x axis** is accomplished with the transformation matrix,

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

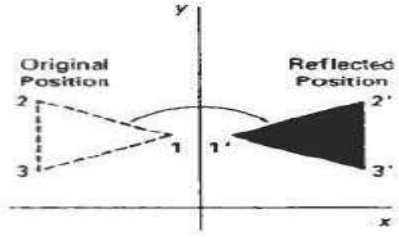
This transformation keeps x values as same, but "flips" the y values of coordinate positions. The resulting orientation of an object after it has been reflected is shown on the figure.



Reflection of an object about y axis

A reflection about the y axis flips x coordinates values and keeping y coordinates as same. The matrix for the transformation is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

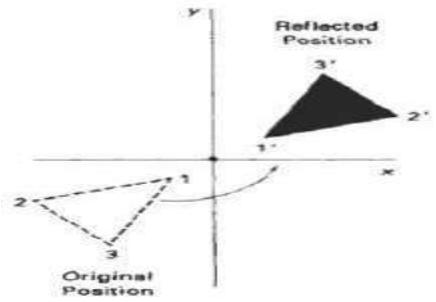


Reflection of an object about y axis

Flip both the x and y coordinates of a point by reflecting relative to an x axis that is perpendicular to the *xy* plane and it passes through the coordinate origin. The matrix for the transformation is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

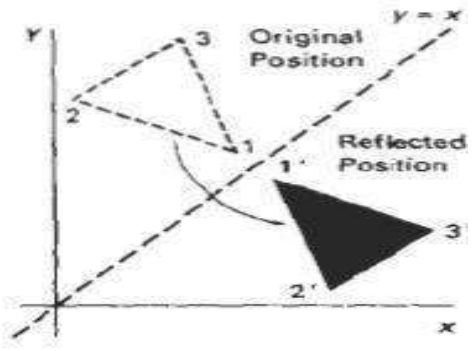
This transformation refered to as a reflection relative to the coordinate orgin.



Reflection of an object about the coordinate origin

Reflection about the diagonal line y = x is accomplished with the transformation matrix is,

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



Reflection of an object with respect to the line y = x

To obtain transformation matrix for reflection about diagonal y = -x for the transformation sequence is;

1. Clock wise rotation by 45^0
2. Reflection about y axis
3. counter clock wise by 45^0

The resulting transformation matrix is follows;

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

SHEAR

A Transformation that distorts the shape of an object is called the shear transformation. Two common shearing transformations are used. One shifts x coordinate values and other shift y coordinate values. In both cases only one coordinate (x or y) changes its coordinates and other preserves its values.

X – Shear

An x-direction shear preserves the y coordinates, but changes the x values which cause vertical lines to tilt right or left as shown in figure

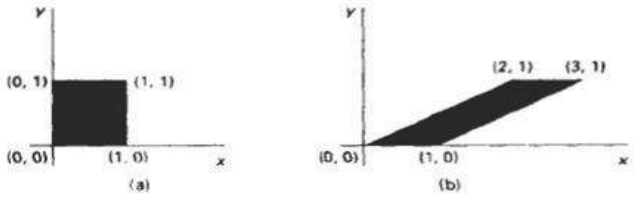


Fig A unit square converted to a parallelogram using the x direction shear matrix with sh_x=2

The Transformations matrix for x-shear is

$$\begin{bmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms the coordinates as

$$x' = x + x\,sh_x \,.y$$
$$y' = y$$

Any real number can be assigned to the shear parameter sh_x. A coordinate position (x,y) is shifted horizontally from x axis (y=0).

Y - Shear

The y- direction shear preserves the x coordinates, but changes the y values which cause horizontal lines which slope up or down. The Transformations matrix for y-shear is

$$\begin{bmatrix} 1 & 0 & 0 \\ shy & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms the coordinates as

$$x' = x$$
$$y' = y + y\, sh_x .x$$

XY – Shear

The transformation matrix for xy-shear

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

which transforms the coordinates as

$$x' = x + x\, sh_x .y$$
$$y' = y + y\, sh_x .x$$

Shearing Relative to other reference line

We can generate x shear and y shear transformations relative to other reference lines. In x shear transformations we can use y reference line and in y shear we can use x reference line.

X - Shear with y reference line

We can generate x-direction shears relative to other reference lines with the transformation matrix

$$\begin{bmatrix} 1 & shx & -shx.yref \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

which transforms the coordinates as

$$x' = x + x\, sh_x (y_{ref} - y)$$
$$y' = y$$

Example

$$Sh_x = ½ \text{ and } y_{ref} = -1$$

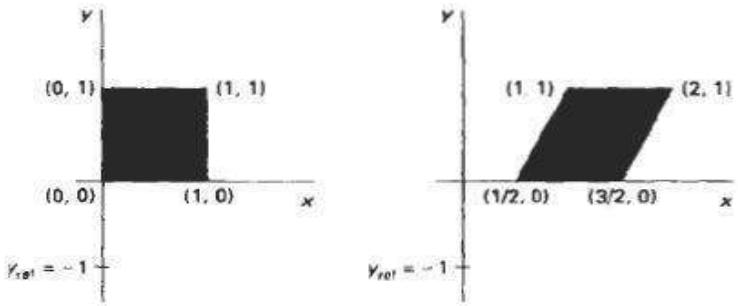


Fig A unit square is transformed to a shifted parallelogram with sh_x=1/2 and y_{ref}= -1

Y - Shear with x reference line

We can generate y-direction shears relative to other reference lines with the transformation matrix which transforms the coordinates as

$$x' = x$$

$$y' = sh_y (x - x_{ref}) + y$$

Example

$$Sh_y = ½ \text{ and } x_{ref} = -1$$

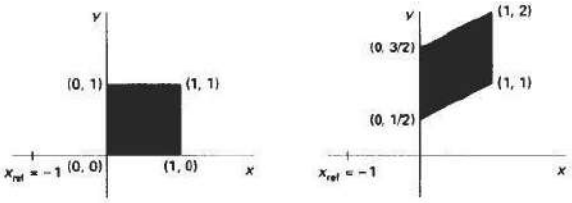


Fig A unit square is transformed to a shifted parallelogram with sh_y=1/2 and x_{ref}= -1

This transformation shifts a coordinate position vertically by an amount proportional to its distance from the reference line x = $x_{ref.}$

WINDOW – TO- VIEWPORT COORDINATE TRANSFORMATION

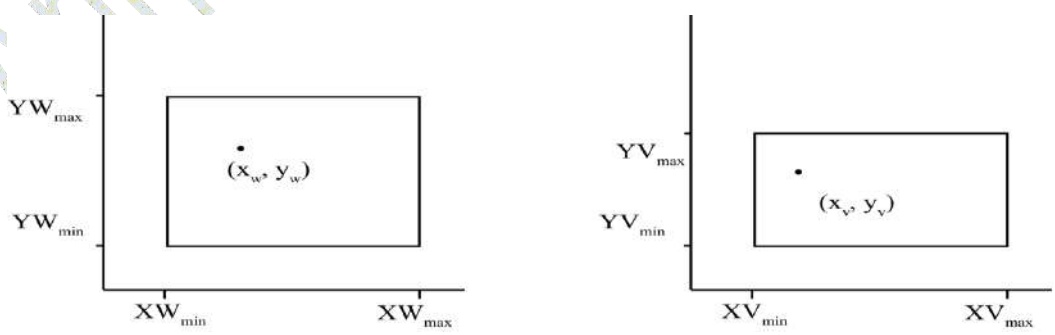


Figure: Window-to-Viewport Mapping

- In the above figure A point at position (x_w , y_w) in the window is mapped into a position (x_v ,y_v) in the view port.
- To maintain the position in the viewport it is in window,

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solve these expressions by introducing the scaling factors (s_x , s_y) and find the values of (x_v , y_v) for the viewport position.

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

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

Find (x_v, y_v),

$$x_v = xv_{min} + (xw - xw_{min})sx$$

$$y_v = yv_{min} + (yw - yw_{min})sy$$

- The above equation for (x_v, y_v) can be derived with a set of transformations that converts the window area into the viewport area as follows

  Step 1: Perform a scaling transformation at the position (xw_{min}, yw_{min}) and scale the window area to the size of the viewport.

  Step 2: Translate the scaled window area to the viewport position.

- If (sx = sy) then relative proportions are maintained, otherwise, the world objects will be stretched or contracted in either x or y directions when displayed on the output device.

Workstation Transformation

- Workstation transformation is done by selecting a window area in normalized space and a viewport area in the display device.
- Workstation transformation is used to partition a view so that different parts of normalized space can be displayed on different output devices.

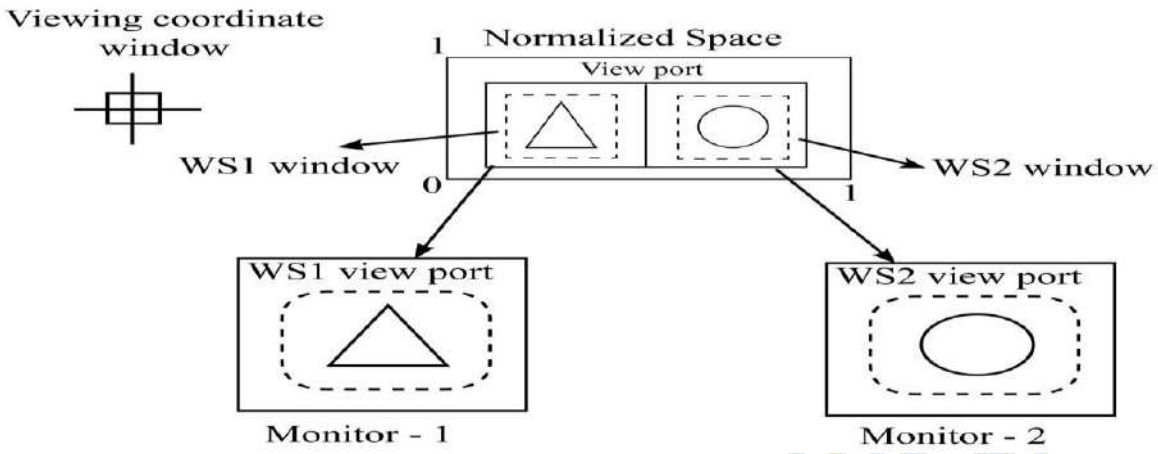


Figure: Work Station Transformations

CLIPPING OPERATIONS

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 clipping. The region against which an object is to clip is called a clip window.

Types of Clipping

1. Point Clipping
2. Line Clipping
3. Area Clipping (Polygons)
4. Curve Clipping
5. Text Clipping

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.

$$xw_{min} \leq x \leq xw_{max}$$

$$yw_{min} \leq y \leq yw_{max}$$

where the edges of the clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) can be either the world-coordinate window boundaries or viewpoint boundaries. If any of these four inequalities is not satisfied, the point is clipped.

LINE CLIPPING

- A line clipping procedure involves the following phases, we can test a given line segment to determine whether it lies completely inside the clipping window.
- If it does not, we try to determine whether it lies completely outside the window.

- If we cannot identify a live as completely inside or completely outside the window, we must perform intersection calculations with one or more clipping boundaries. We process lines through the "inside - outside" tests by checking the line end points.
- Let us consider the figure 3.15. A line with both end points inside all clipping boundaries such as the line from P_1 to P_2 is saved. A line with both end points outside any one of the clip boundaries (line P_3 P_4) is outside the window. All other lines cross one or more clipping boundaries and require calculation of multiple intersection points.
- To minimize calculations, clipping algorithms are used that can efficiently identify outside lines and reduce intersection calculations.
- For a line segment with endpoints (x_1, y_1) and (x_2, y_2) and one or both end points outside the clipping rectangle, the parametric representation.

$$x = x_1 + u\,(x_2 - x_1)$$
$$y = y_1 + u\,(y_2 - y_1), \quad 0 \le u \le 1$$

- could be used to determine values of parameter u.If the value of u is within 0 to 1, the line segment does not cross into the clipping area.
- If the value of u is outside the range of 0 to 1, the line does not enter the interior of the window at the boundary.

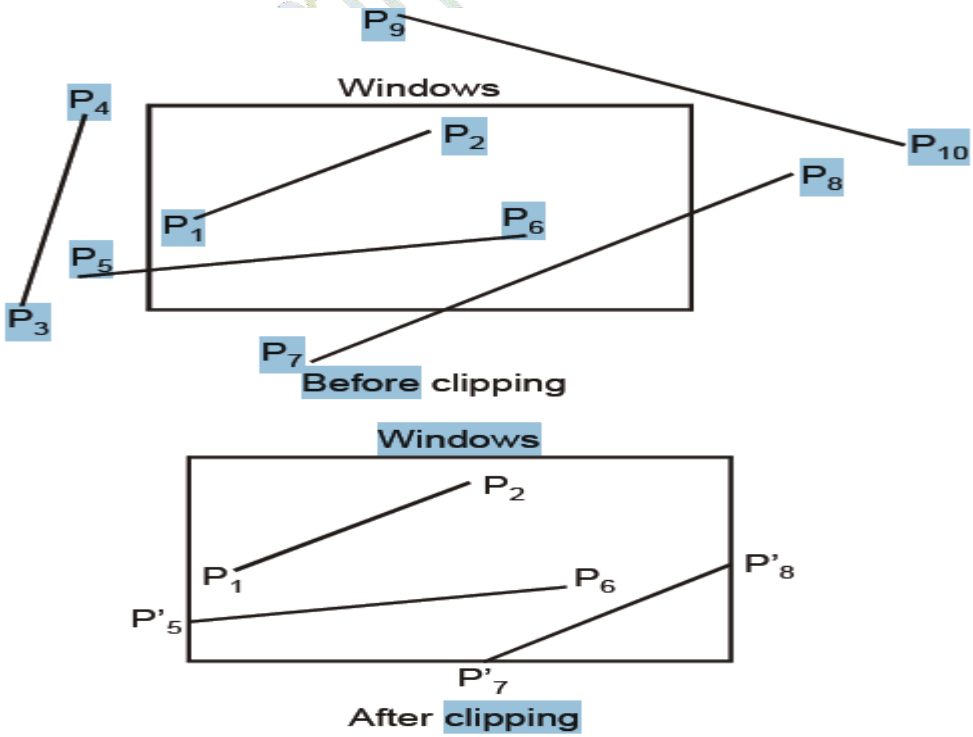


Figure: Line Clipping

COHEN SUTHERLAND LINE CLIPPING

- This method speeds up the processing of line segments by performing initial tests that reduce the number of intersections that must be calculated.
- Every line end point in a picture is assigned a four digit binary code, called a region code that identifies the location of the point. It is shown in figure.

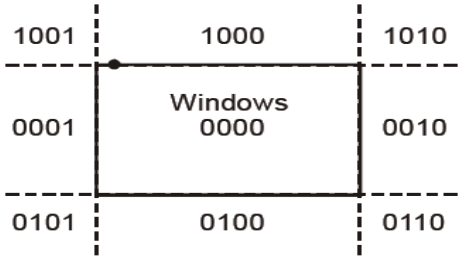


Figure 3.16 Region Code

- Each bit position in the region code is used to indicate one of the 4 relative coordinate positions of the point with respect to the clip window; to the left, right, top or bottom.
- By numbering the bit positions in the region code as 1 through 4 from right to left, the coordinate regions can be correlated with the bit positions as

bit 1: left
bit 2: right
bit 3: below
bit 4: above

- The value 1 in any bit position indicates that the point is in that relative position; otherwise the bit position is set to 0.
- If a point in within the clipping rectangle, the region code is 0000; A point that is below and to the left of the rectangle has a region code of 0101.
- Bit values in the region code are determined by comparing endpoint coordinate values (x, y) to the clip boundaries. Bit 1 is set to 1 if x < x10 min.
- The other 3 bit values can be determined using similar comparisons.
- Any lines that are completely contained within the window boundaries have a region code of 0000 for both endpoints and these lines can be trivially accepted.
- Any lines that have a 1 in the same bit position in the region codes for each end points are completely outside the clipping window, and these lines are trivially rejected.
- Discard the line that has a region code of 1001 for one end point and a code of 0101 for the other end point.

- The clipping of lines is explained in the figure.In the above example, startup with the bottom end point of the line from P1 to P2, P1 is checked against the left, right and bottom boundaries. Then and intersection point '1P is found with the bottom boundary and the line section '1P to P2 is discarded.
- Since P2 in outside the clip window, again it in checked with the left point of the window. Intersection point '2P is calculated, but this point in above the window, so "2P is calculated and the line from '1P to "2P is saved.
- In the next line, point P3 is to the left of the clipping window and '3P is calculated and the line P3 to '3P is eliminated. By checking the region code for the line '3P to P4, the line is below the clip window and it can be discarded.

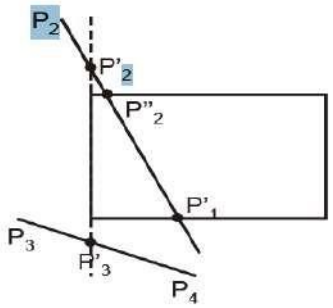


Figure :Line Clipping

- The intersection points can be calculated as

$$y = y1 + m\,(x - x1)$$

- where x value is set either to xw_{min} or to xw_{max} and slope of the line is calculated as

$$m = (y2 - y1)/(x2 - x1)$$

In the horizontal boundary,

$$x = x_1 + \frac{y - y_1}{m}$$

with y set either to yw_{min} or to yw_{max}.

LIANG - BARSKY ALGORITHM

The following parametric equations represent a live from (x1, y1) to (x2, y2) along with its infinite extension:

$$\begin{cases} x = x_1 + \Delta x \cdot u \\ y = y_1 + \Delta y \cdot u \end{cases}$$

Where, $\Delta x = x2 - x1$ and $\Delta y = y2 - y1$ and $0 \le u \le 1$

Consider the Figure and notice that when we traverse along the extended line with u increasing from $-\infty$ to $+\infty$ we first move from the outside to the inside of the clipping window's two boundary lines

(bottom and left), and then move from the inside to the outside of the other two boundary lines (top and right).

If we use u1 and u2, where u1 ≤ u2, u2 = minimum (1, u1, ur) where u1, ub, ut and ur correspond to the intersection point of the extended like with the window's left, bottom, top and right boundary, respectively.

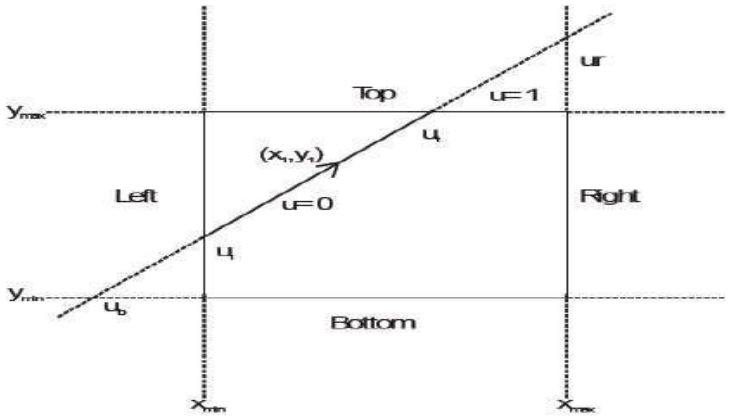


Figure: Extension of line for clipping

For point (x, y) inside the clipping window, we have

$$x_{min} \le x_1 + \Delta x.u \le x_{max}$$

$$y_{min} \le y_1 + \Delta y.u \le y_{max}$$

The four inequalities can be rewritten as

$$p_k u = q_k,\ k = 1, 2, 3, 4$$

Where,

$p_1 = -\Delta x$ $\quad q_1 = x_1 - x_{min}$ (left)

$p_2 = \Delta x$ $\quad q_2 = x_{max} - x_1$ (right)

$p_3 = -\Delta y$ $\quad q_3 = y_1 - y_{min}$ (bottom)

$p_4 = \Delta y$ $\quad q_4 = y_{max} - y_1$ (top)

Some facts are,

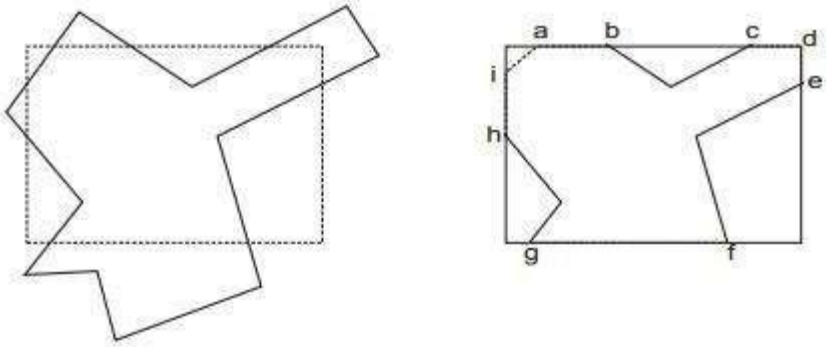
- if $p_k = 0$, the line is parallel to the corresponding boundary and
  - if $q_k < 0$, the line is completely outside the boundary and can be truncated
  - if $q_k \ge 0$, the line is inside the boundary and needs further consideration.
- If pk < 0, the extended line proceeds from the outside to the inside of the corresponding boundary line.
- If pk > 0, the extended line proceeds from the inside to the outside of the corresponding boundary line.

- When pk≠0, the value of u that corresponds to the intersection point is *qk*/*pk*.
- The Liang Barsky algorithm for finding the visible portion of the line can be started as a four-step process.

1. If *pk* =0 and *qk* < 0 for any *k*, eliminate the line and stop, otherwise proceed to the next step.
2. For all *k* such that *pk* < 0, calculate *rk* = *qk*/*pk*.
   Let *u*1 be the maximum of the set containing 0 and the calculated *r* values.
3. For all *k* such that *pk* > 0, calculate *rk* = *qk* / *pk*.
   Let *u*2 be the minimum of the set containing 1 and the calculated *r* values.
4. If *u*1 > *u*2, eliminate the line since it is completely outside the clipping window otherwise, use *u*1 and *u*2 to calculate the endpoints of the clipped line.

POLYGON CLIPPING

A polygon is nothing but the collection of lines. As polygon is a closed solid area, after clipping, it should remain closed. To achieve this, we require an algorithm to generate additional line segments which make the polygon as a closed area. For example, in the following figure the lines a–b, c–d, d–e, f–g and h–i are added to polygon description to make it closed.



(a) Before Clipping *(b) After Clipping*

SUTHERLAND - HODGEMAN POLYGON CLIPPING

A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn.

Beginning with the original set of polygon vertices, first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.

The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper as shown in figure 3.20. At each step a new set of polygon vertices is generated and passed to next window boundary clipper. This is the fundamental idea used in the Sutherland-Hodgeman algorithm.

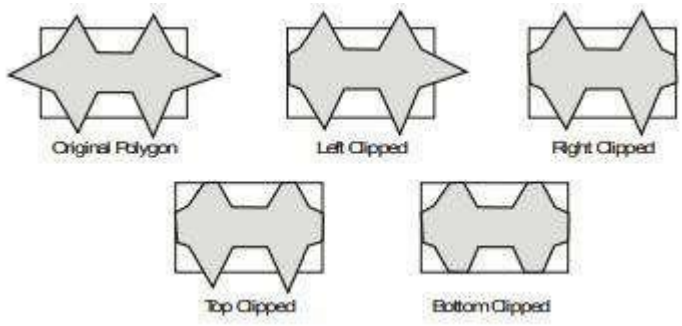


Figure: 3.20.Clipping a polygon against successive window boundaries

The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane.

Each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane.

This results in four possible relationships between the edge and the clipping boundary or plane as shown in Figure .3.21.

(1) If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside, then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (Figure 3.21a).

(2) If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list (Figure 3.21b).

(3) If the first vertex of the edge is inside the window boundary, and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (Figure 3.21c).

(4) If both vertices of the edge are outside the window boundary, nothing is added to the output list. (Figure 3.21d). Once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

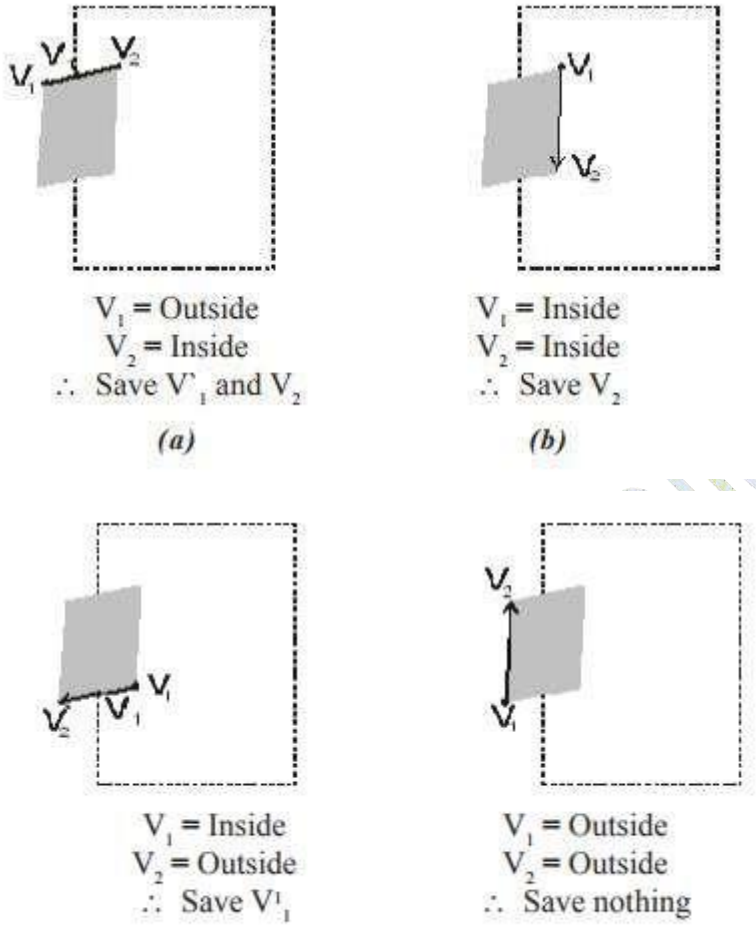


Figure 3.21 Clipping Polygon by the edge.

Going through above four cases, we can realize that there are two key processes in this algorithm.

(1) Determining the visibility of a point or vertex.

(2) Determining the intersection of the polygon edge and the clipping plane. Determining the visibility of a point or vertex can be described as follows.

Consider that two points A and B define the window boundary and point under consideration is V, then these three points define a plane. Two vectors which lie in that plane are AB and AV. If this plane is considered in the xy plane, then the vector across product AV × AB has only a Z component given by (xV–xA) (yB–yA) – (yV–yA) (xB–xA). The sign of the Z component decides the position of point V with respect to window boundary.

If Z is Positive – point is on the right side of the window boundary. Zero – point is on the window boundary. Negative – point is on the leftside of the window boundary. Sutherland - **Hodgeman Polygon clipping Algorithm:**

Step 1: Read coordinates of all vertices of the polygon.

Step 2: Read coordinates of the clipping window.

Step 3: Consider the left edge of the window.

Step 4: Compare the vertices of each edge of the polygon, individually with the clipping plane.

Step 5: Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.

Step 6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.

Step 7. Stop.

CURVE CLIPPING

Curve clipping procedure involves non linear equations:

Step 1: The boundary rectangle for a circle or other curved object can be used to test for overlap with the clip window.

Step 2: If the bounding rectangle of the object is completely inside the window, the object is saved.

Step 3: Otherwise, the object is discarded. It is illustrated in the figure 3.22.

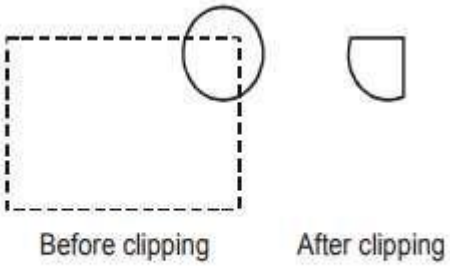


Figure 3.22 Curving Clipping.

TEXT CLIPPING

The simplest method for processing character strings relative to a window boundary is to use the all-or-none-string clipping. It is shown in figure 3.22 if all the string is inside the clip window, it is saved, otherwise the entire string is discarded.

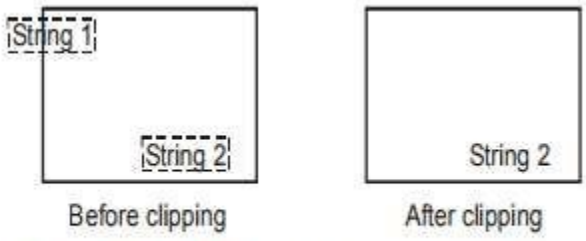


Figure 3.23 All – or – none string clipping.

All-or-none-character Clipping

Here we discard only those characters that are not completely inside the window (figure 3.24). In this case, the boundary limits of individual characters are compared to the window, any character that either overlaps or is outside a window boundary is clipped.

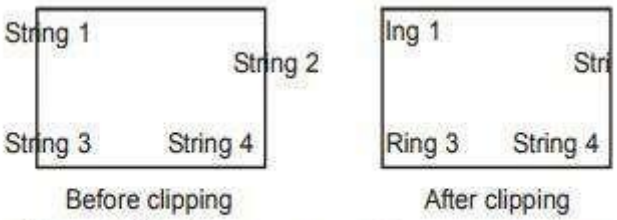


Figure 3.24 All – or – none character clipping.

Clipping Individual Character Components Here, the characters are treated as lines. If an individual character overlaps a clip window boundary we clip off the parts of the character that are outside the window (figure 3.25) outline character fonts formed with line segments can be processed using line clipping algorithm.

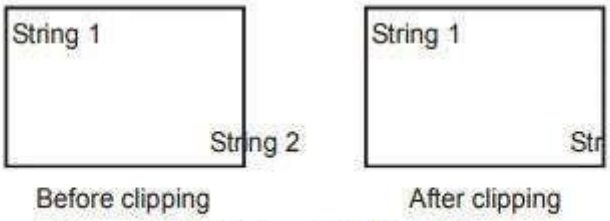


Figure 3.25 Clipping individual character.

4. THREE DIMENSIONAL DISPLAY METHODS

Three Dimensional Concepts

Three Dimensional Display Methods

- To obtain a display of a three dimensional scene that has been modeled in world coordinates, we must setup a co-ordinate reference for the 'camera'.
- This coordinate reference defines the position and orientation for the plane of the camera film (Figure 4.1) 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.
- The objects can be displayed in wire frame form, or we can apply lighting and surface rendering techniques to shade the visible surfaces.

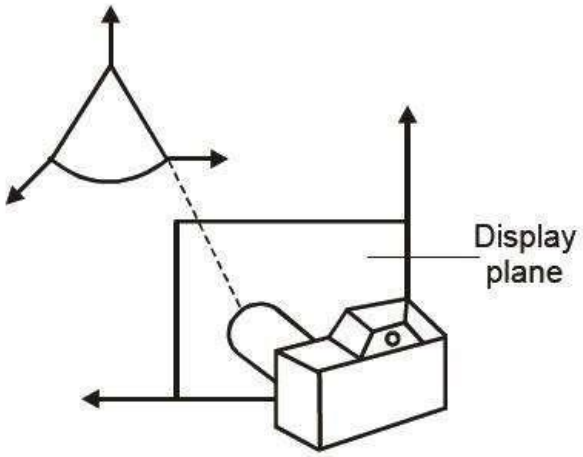


Figure 4.1 Coordinate reference for obtaining a view of a 3D scene.

Parallel Projection

- Parallel projection is a method for generating a view of a solid object is to project points on the object surface along parallel lines onto the display plane.
- In parallel projection, parallel lines in the world coordinate scene project into parallel lines on the two dimensional display planes.
- This technique is used in engineering and architectural drawings to represent an object with a set of views that maintain relative proportions of the object.
- The appearance of the solid object can be reconstructed from the major views.

Perspective Projection

- It is a method for generating a view of a three dimensional scene is to project points to the display plane along converging paths.

- This makes objects further from the viewing position to be displayed smaller than objects of the same size that are nearer to the viewing position.
- In a perspective projection, parallel lines in a scene that are not parallel to the display plane are projected into converging lines.

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

Depth Cueing

- Depth information is important to identify the viewing direction, which is the front and which is the back of displayed object.
- Depth cueing is a method for indicating depth with wire frame displays is to vary the intensity of objects according to their distance from the viewing position.
- 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.

Visible line and surface identification

- A simplest way to identify the visible line is to **highlight** the visible lines or to display them in a different color.
- Another method is to display the non visible lines as **dashed** lines.

Surface Rendering

- Surface Rendering method is used to generate a degree of realism in a displayed scene.
- Realism is attained in displays by setting the surface intensity of objects according to the lighting conditions in the scene and surface characteristics.
- Lighting conditions include the intensity and positions of light sources and the background illumination.
- Surface characteristics include degree of transparency and how rough or smooth the surfaces are to be.

Exploded and Cutaway Views

- Exploded and cutaway views of objects can be used to show the internal structure and relationship of the objects parts.
- An alternative to exploding an object into its component parts is the cut away view which removes part of the visible surfaces to show internal structure.

Three-dimensional and Stereoscopic Views

- In Stereoscopic views, three dimensional views can be obtained by reflecting a raster image from a vibrating flexible mirror.
- The vibrations of the mirror are synchronized with the display of the scene on the CRT.

- As the mirror vibrates, the focal length varies so that each point in the scene is projected to a position corresponding to its depth.
- Stereoscopic devices present two views of a scene; one for the left eye and the other for the right eye.
- The two views are generated by selecting viewing positions that corresponds to the two eye positions of a single viewer.
- These two views can be displayed on alternate refresh cycles of a raster monitor, and viewed through glasses that alternately darken first one lens then the other in synchronization with the monitor refresh cycles.

THREE DIMENSIONAL GRAPHICS PACKAGES

The 3D package must include methods for mapping scene descriptions onto a flat viewing surface.

There should be some consideration on how surfaces of solid objects are to be modeled, how visible surfaces can be identified, how transformations of objects are preformed in space, and how to describe the additional spatial properties.

World coordinate descriptions are extended to 3D, and users are provided with output and input routines accessed with specifications such as

- Polyline3(n, WcPoints)
- Fillarea3(n, WcPoints)
- Text3(WcPoint, string)
- Getlocator3(WcPoint)
- Translate3(translate Vector, matrix Translate)

Where points and vectors are specified with 3 components and transformation matrices have 4 rows and 4 columns.

INTERACTIVE INPUT METHODS AND GRAPHICAL USER INTERFACES

Graphical Input Data

Graphics software can use a variety of input data:

- Coordinate positions
- Character-string specifications
- Geometric transformation values
- Viewing conditions
- Illumination parameters

Many graphics systems and various standards organizations (ISO, ANSI) provide input functions for such data.

Input functions are often classified by the type of data they process.

Logical Classification of Input Devices

When input functions are classified by data type,any device that can supply needed data is called a logical input device for that data type.

- Logical input-data classifications:
- Locator – specify one coordinate position
- Stroke – specify a set of coordinate positions
- String – specify some text input
- Valuator – specify a scalar value
- Choice – select a menu option
- Pick – select a component of a picture

Locator Devices

- Common approach is to use the screen cursor at some location.
- Mouse, touchpad, thumbwheel, dial, touch-screen,etc. can be used for screen cursor positioning.
- Keyboards, particularly arrow keys (with modifiers) can be used to indicate screen cursor movement.
- Light pens can also be used, but there are less common than the other techniques.

Stroke Devices

- Locator devices can also be used to provide multiple coordinate positions.
- "Click and drag" actions with a mouse provide stroke input, specifically a start and end coordinate (as for a line).
- Buttons (e.g. on a keyboard) can be used to modify the actions. For example, pressing a shift key may constrain coordinate positions to a horizontal or vertical line.

String Devices

- The obvious choice for string input is a keyboard.
- The keyboard need not necessarily be a "full" keyboard as found on a traditional computer; it is often also one of these:
    - A keyboard with a limited set of keys appropriate to a specific application
    - A simulated keyboard displayed on a touch-sensitive screen.

Valuator Devices

- Numerous devices can be used as valuators:
    - Dials (e.g. potentiometers or rotary switches)

- Keyboards
- Joysticks, pressure-sensitive devices, etc.
- Various on-screen widgets like slider bars, buttons, rotating scales, etc.

- In all cases it is appropriate to provide user feedback so verification (and correction) of the input value is possible.

Choice Devices

Numerous possibilities exist:

- One of many pushbuttons
- Keyboards
- On-screen menus and cursor-positioning devices
- Multi-level menus for complicated choices
- Voice input (text suggests this for small # of choices)

Pick Devices

- Can be used to select an entire object, a facet of a surface, a polygon edge, a vertex, etc.
- Using the mouse, we translate a screen coordinate position to a world coordinate position using inverse transformations.
- Object selection first tries to determine if the coordinate position is uniquely associated with a single object.
- If that fails, then coordinate extents of individual object components (e.g. line segments) can be tested.

Object Selection

- If coordinate-extent tests don't resolve to a single object, then distance to individual line segments could be computed (see next slide).
- Pick actions can also specify selection of multiple objects at once, especially if these multiple objects are in a specified region, or are somehow grouped to represent a single object.
- Keyboards can be used to allow selection of an on-screen objects and sequencing through the objects.
- Named objects can also be selected by keyboard.

Figure 20-1 Distances to line segments from a pick position.

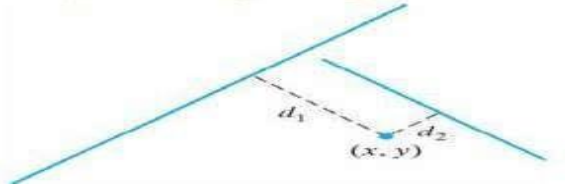


Figure 20-2 A pick window with center coordinates (x_p, y_p), width w, and height h

Graphical Data Input Functions

- Typical input options
- Interaction mode:
  - Program-initiated input
  - User-initiated input
  - Simultaneous operation
- Physical input device selection
- When and where to obtain input for a particular data set

Input Modes

- Request Mode
  - Application initiates data input
  - Processing waits (blocked) until input is available
- Sample Mode
  - Application and input devices operate simultaneously
  - When application needs new data, it "samples" the data that has been provided
- Event Mode
  - Application and input devices operate simultaneously
  - Input actions are saved in an event queue, which is processed by the application when desire

Echo Feedback

- As mentioned earlier, it is appropriate in many cases for input activity to result in some feedback to a user.
- The feedback can take many forms, depending on the type of input
  - Echo of keyboard input Display of a selection window
  - Highlighting of selected objects
  - Range of values or resolution (for valuator input)

Callback Functions

- These include functions that specify what actions are to take place when certain events (like input actions) occur.
- Examples include
    - "normal" and special keyboard entry
    - mouse button presses and mouse motion

Interactive Picture-Construction

Basic Positioning Methods

Obvious possibility include using a mouse or a trackball Specific object type could be specified (e.g. square, ellipse) and "filled in" given parameters (e.g. from keyboard or mouse)

- Numeric position information could be displayed as text on the screen (or other device)

Keyboard, dials, etc. could be used to make small interactive adjustments in coordinate values

Dragging

- Common technique – select an object and draw to a different location Often performed with mouse – press button to select object, move mouse to desired location, release button

Constraints

- With additional input (e.g. from keyboard), the parameters for an interactively-drawn object can be limited. For example, holding shift key while drawing a line may constrain the angle between the line and a coordinate axis

5. THREE DIMENSIONAL TRANSFORMATIONS

BASIC TRANSFORMATION

Geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the z-coordinate.

Translation

In a three dimensional homogeneous coordinate representation, a point or an object is translated from position P = (x, y, z) to position () P' = x',y',z with the matrix operation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \rightarrow (1)$$

(or) $P' = T.P$

Parameters t_x, t_y and t_z specifying translation distances for the coordinate directions x, y and z are assigned any real values.

The matrix representation in equation (1) in equivalent to the three equations.

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

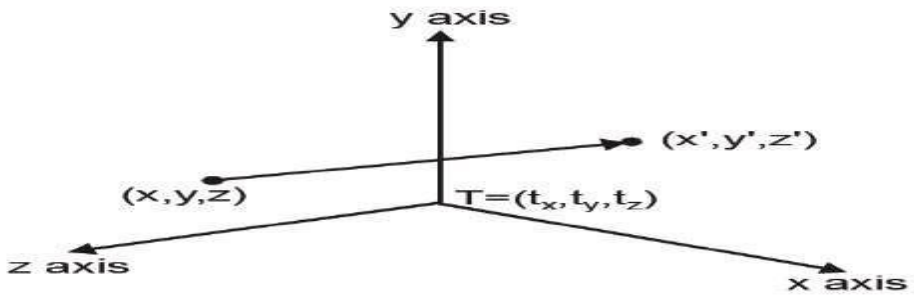


Figure 4.6 Translation of a point.

Inverse of the translation matrix in equation (1) can be obtained by negating the translation distances tx, ty and tz.

This produces a translation in the opposite direction and the product of a translation matrix and its inverse produces the **identity matrix**.

Rotation

- To generate a rotation transformation for an object an axis of rotation must be designed to rotate the object and the amount of angular rotation is also be specified.

- Positive rotation angles produce counter clockwise rotations about a coordinate axis.

Co-ordinate Axes Rotations

The 2D z axis rotation equations are easily extended to 3D.

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

$$y' = x\sin\theta + y\cos\theta \qquad \rightarrow (2)$$

$$z' = z$$

Parameters θ specifies the rotation angle. In homogeneous coordinate form, the 3D z axis rotation equations are expressed as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \qquad \rightarrow (3)$$

which we can write more compactly as

$$P' = R_2(\theta).P \qquad \rightarrow (4)$$

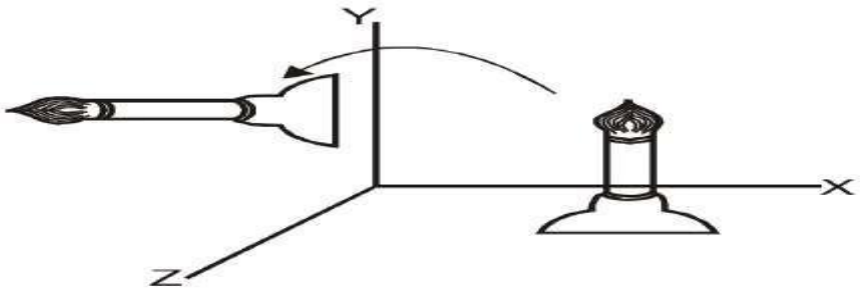


Figure 4.7 Rotation of an object about z axis.

Transformation equations for rotation about the other two coordinate axes can be obtained with a cyclic permutation of the coordinate parameters x, y and z in equation (2) i.e., we use the replacements.

$$x \rightarrow y \rightarrow z \rightarrow x \qquad \rightarrow (5)$$

Substituting permutations (5) in Equation (2), we get the equations for an x-axis rotation:

$$\left.\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned}\right\} \qquad \rightarrow (6)$$

which can be written in the homogeneous coordinate form

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \rightarrow (7)$$

(or) $P' = R_x(\theta).P$ → (8)

Rotation of an object around the x-axis is demonstrated in Figure 4.8.

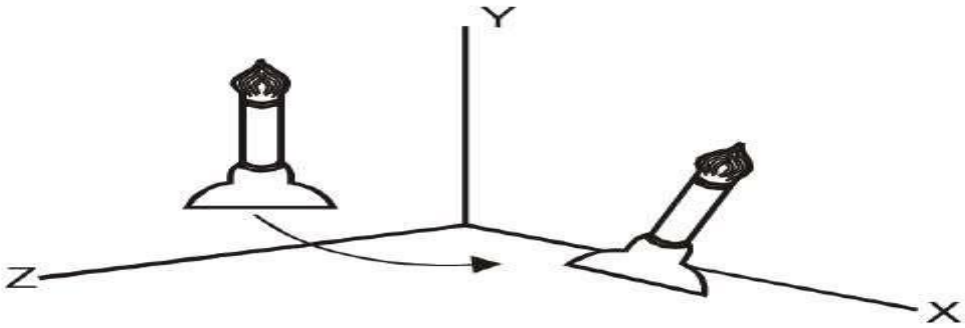


Figure 4.8 Rotation of an object about y axis.

Cyclically permuting coordinates in equation (6) give the transformation equation for a y axis rotation.

$$\left.\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \\ y' &= y \end{aligned}\right\} \quad \rightarrow (9)$$

The matrix representation for y-axis rotation is

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \rightarrow (10)$$

(or) $P' = R_y(\theta).P$

An example of y axis rotation is shown in Figure 4.9.

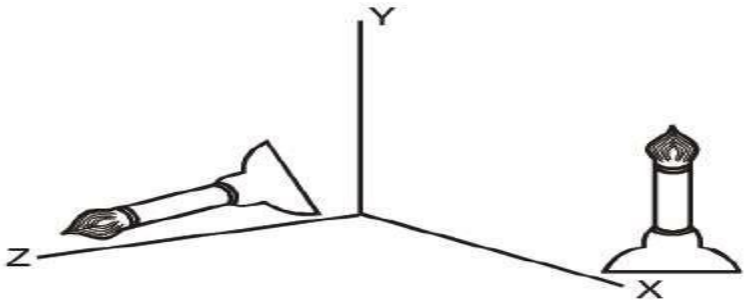


Figure 4.9 Rotation of an object about x axis.

- An inverse rotation matrix is formed by replacing the rotation angle θ by –θ.

- Negative values for rotation angles generate rotations in a clockwise direction, so the identity matrix is produced when any rotation matrix is multiplied by its inverse.
- Since only the sine function is affected by the change in sign of the rotation angle, the inverse matrix can also be obtained by interchanging rows and columns. (i.e.) we can calculate the inverse of any rotation matrix R by evaluating its transpose (R–1 = RT).

General Three Dimensional Rotations

- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate axes rotations.
- We obtain the required composite matrix by

  (1) Setting up the transformation sequence that moves the selected rotation axis onto one of the coordinate axes.

  (2) Then set up the rotation matrix about that coordinate axis for the specified rotation angle.

  (3) Obtaining the inverse transformation sequence that returns the rotation axis to its original position.
- In the special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes, we can attain the desired rotation with the following transformation sequence

  (1) Translate the object so that the rotation axis coincides with the parallel coordinate axis.

  (2) Perform the specified rotation about that axis.

  (3) Translate the object so that the rotation axis is moved back to its original position.
- When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we need to perform some additional transformations.
- In such case, we need rotations to align the axis with a selected coordinate axis and to bring the axis back to its original orientation.

- Given the specifications for the rotation axis and the rotation angle, we can accomplish the required rotation in five steps:

  (1) Translate the object so that the rotation axis passes through the coordinate origin.

  (2) Rotate the object so that the axis of rotation coincides with one of the coordinate axes.

  (3) Perform the specified rotation about that coordinate axis.

  (4) Apply inverse rotations to bring the rotation axis back to its original orientation.

  (5) Apply the inverse translation to bring the rotation axis back to its original position.

Scaling

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

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \rightarrow (11)$$

$$(\text{or}) \qquad P' = S.P \quad \rightarrow (12)$$

- Where scaling parameters sx, sy and sz are assigned any position values.
- Explicit expressions for the coordinate transformations for scaling relative to the origin are

$$\left.\begin{aligned} x' &= x \cdot s_x \\ y' &= y \cdot s_y \\ z' &= x \cdot s_z \end{aligned}\right\} \quad \rightarrow (13)$$

- Scaling an object changes the size of the object and repositions the object relative to the coordinate origin.
- If the transformation parameters are not equal, relative dimensions in the object are changed.
- The origin shape of the object is preserved with a uniform scaling (sx = sy = sz). (Figure 4.10) shows the result of scaling an object uniformly with each scaling parameter set to 2.

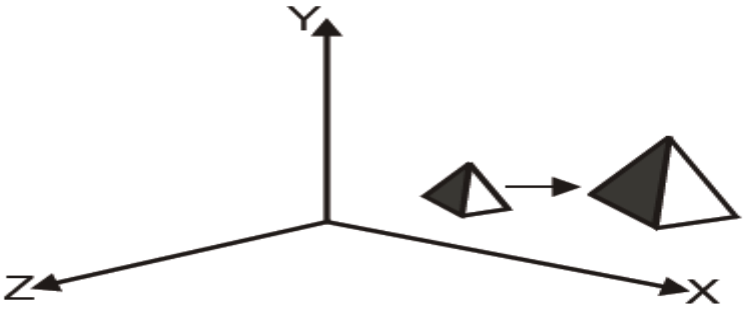


Figure 4.10 Scaling and translation of an object.

- Scaling with respect to a selected fixed position (xf, yf, zf) can be represented with the following transformation sequence:

(1) Translate the fixed point to the origin.

(2) Scale the object relative to the coordinate origin using equation (11).

(3) Translate the fixed point back to its original position.

This sequence of transformation is shown in Figure 4.11.

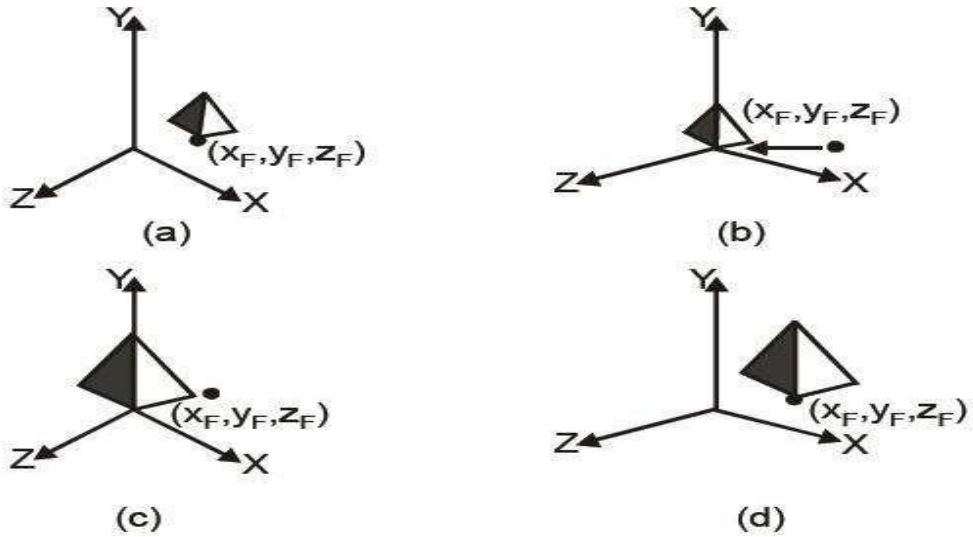


Figure 4.11 Scaling with a sequence of transformations.

The matrix representation for an arbitrary fixed point scaling can be expressed as the concatenation of the **translate-scale-translate** transformations are

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

$$\begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \rightarrow (14)$$

- Inverse scaling matrix m formed by replacing the scaling parameters sx, sy and sz with their reciprocals.
- The inverse matrix generates an opposite scaling transformation, so the concatenation of any scaling matrix and its inverse produces the identity matrix.

OTHER TRANSFORMATIONS

Reflections

- A 3D reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.
- Reflection relative to a given axis are equivalent to 180° rotations about the axis.
- When the reflection plane in a coordinate plane (either xy, xz or yz) then the transformation can be a conversion between left-handed and right-handed systems.
- An example of a reflection that converts coordinate specifications from a right handed system to a left-handed system is shown in Figure 4.12.
- This transformation changes the sign of z coordinates, leaves the x and y coordinate values unchanged.
- The matrix representation for this reflection of points relative to the xy plane is

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

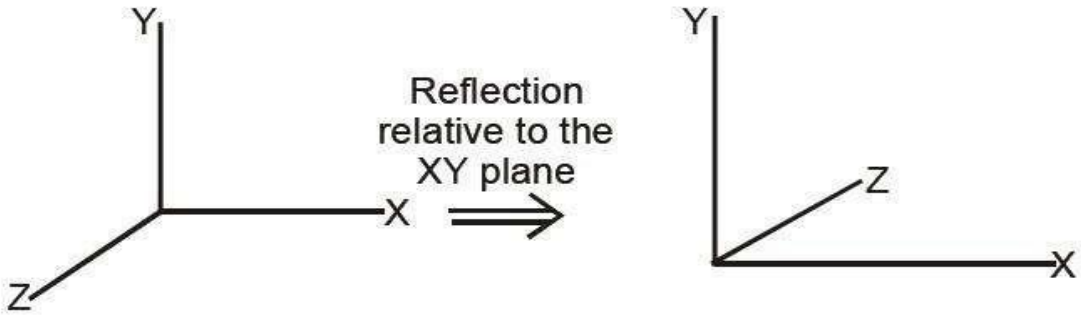


Figure 4.12 Conversion of coordinate specifications.

- Reflections about other planes can be obtained as a combination of rotations and coordinate plane reflections.

Shears

- Shearing transformations are used to modify object shapes.

- They are also used in three dimensional viewing for obtaining general projections transformations.
- The following transformation produces a z-axis shear.

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

- Parameters a and b can be assigned any real values.
- This transformation matrix is used to alter x and y coordinate values by an amount that is proportional to the z value, and the z coordinate will be changed.
- Boundaries of planes that are perpendicular to the z axis are shifted by an amount proportional to z figure 4.13 shows the effect of shearing matrix on a unit cube for the values a = b = 1.

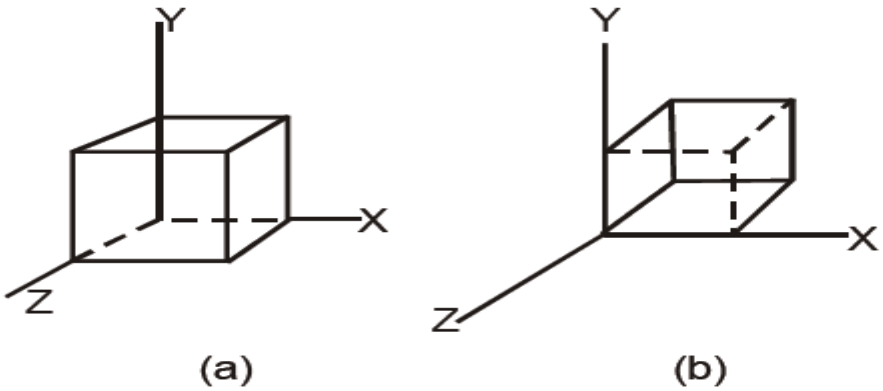


Figure 4.13 Shearing of a unit cube.

CLASSIFICATION OF VISIBLE SURFACE DETECTION ALGORTHIM

BACK-FACE DETECTION

- A fast and simple object-space method for identifying the back faces of a polyhedron is based on the '**inside-outside**' tests.
- A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C and D if

$$A_x + B_y + C_z + D < 0$$

- When an inside point is along the line of sight to the surface, the polygon must be a backface.
- This test is done by considering the normal N to a polygon surface, which has Casterian components (A, B, C).

- If V is a vector in the viewing direction from the eye (or "camera") position, then the polygon is a backface if

$$V.N > 0$$

- If object descriptions have been connected to projection coordinates and the viewing direction is parallel to the viewing Zv axis then V = (0, 0, Vz) and

$$V . N = Vz . C$$

- So we have to consider the sign of C, the z component of the normal vector N. It is shown in Figure.5.14.

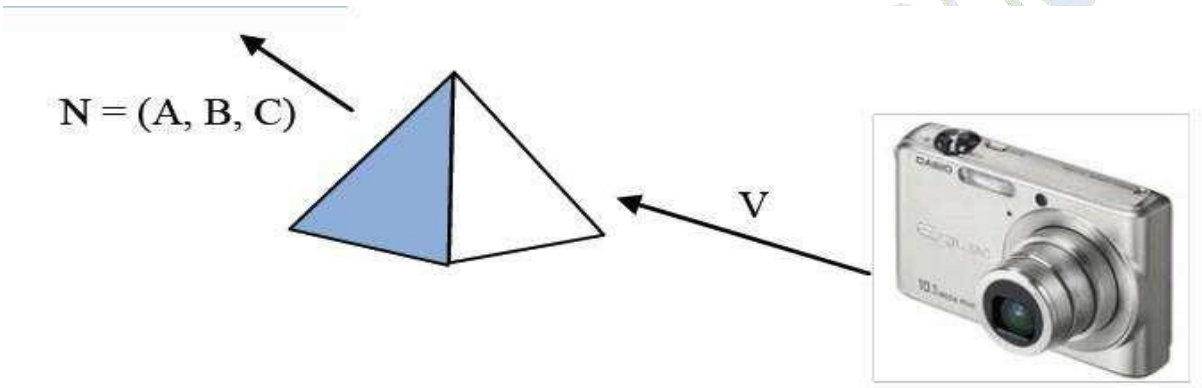


Figure.5.14 Vector V in the viewing directions and a back face normal vector N of a polyhedron.

- In a right-handed viewing system with viewing direction along the negative Zv axis (Figure.5.15) the polygon is back-face if C < 0.
- If C = 0, we cannot see any face, since the viewing direction is graying that polygon.
- Thus we can label any polygon as a back-face if its normal vector has a Z component value

$$C \leq 0$$

- Back-faces have normal vectors that point away from the viewing position and are identified by C ³ 0 when the viewing direction is along the positive *Zv* axis.

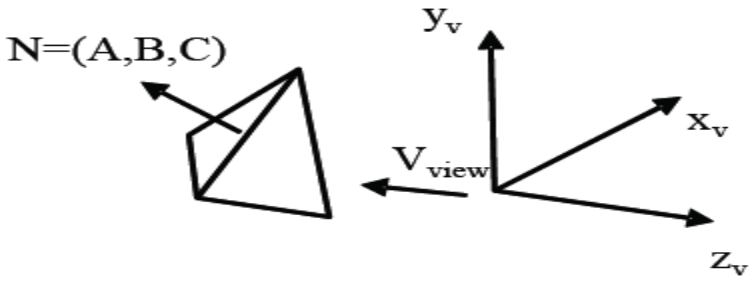


Figure.5.15 Back face when the viewing direction is along the negative Z_v axis.

DEPTH BUFFER METHOD (OR) Z- BUFFER ALGORITHM

- This method compares surface depths at each pixel position on the projection plane.
- The surface depth is measured from the view plane along the Z axis of a viewing system.
- When object description is converted to projection coordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z-values.
- The Z buffer algorithm is usually implemented in the normalized coordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane.
- The implementation requires another buffer memory called **z-buffer** along with the frame buffer memory required for raster display devices.
- A z-buffer is used to store depth values for each (x, y) position as surfaces are processed and the frame buffer stores the intensity values for each position.
- At the beginning, z-buffer is initialized to 0, representing the z-value at the back clipping plane and the frame buffer is initialized to the background color.
- Each surface listed in the display file is then processed one scan line at a time, calculating the depth (z value) at each (x, y) pixel position.
- The calculated depth value is compared to the value previously stored in the z-buffer at that position.
- If the calculated depth value is greater than the value stored in the z-buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same x,y location in the frame buffer.
- For example in the following Fig.5.16 among three surfaces, surface s has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

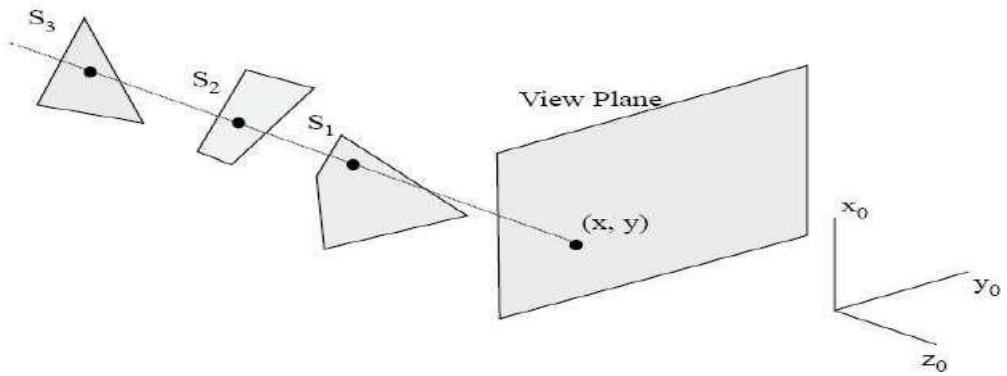


Fig.5.16 Depth Buffer Method.

Z-buffer Algorithm

Step-1: Initialize the *z*-buffer and frame buffer so that for all buffer positions

Z-buffer (*x*, *y*) = 0 and frame buffer(*x*, *y*) = Ibackground

Step-2: During scan conversion process, for each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

Calculate *z*-value for each (*x*,*y*) position on the polygon.

If *z*>Z-buffer(*x*,*y*) then set

z-buffer(*x*,*y*)= Z, frame-buffer(*x*,*y*) = Isurface(*x*,*y*)

Step-3: Stop

$I_{background} \Rightarrow$ the value for the background intensity

$I_{surface} \Rightarrow$ the projected intensity value for the surface at pixel position (x,y)

- After processing all the surfaces, the ***z-buffer***, contains ***depth values*** for the variable surfaces and the ***frame buffer*** contains the corresponding intensity values for those surfaces.
- To calculate *z*-values, the plane equation

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

- is used where (*x*, *y*, *z*) is any point on the plane, and the coefficient A, B, C and D are constants describing the spatial properties of the plane.

$$\therefore z = \frac{-A_x - By - D}{C}$$

- If at (*x*,*y*), the above equation evaluates to *z*, then at (*x*+D*x*, *y*) the value of *z* is

$$z_1 = \frac{A}{C}(\Delta x)$$

- Only one subtraction is needed to calculate z(x+1, y), given z(x, y) since the quotient A/C is constant and $\Delta x=1$.
- A similar incremental calculation can be performed to determine the first value of z on the next scanline, decrementing by B/C for each Δy.

Advantages

(1) It is very easy to implement.

(2) It can be implemented in hardware to overcome the speed problem.

(3) Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages

(1) It requires an additional buffer and hence the large memory.

(2) It is a time consuming process as it requires comparison for each pixel instead of the entire polygon.

UFFER METHOD

- An extension of the ideas in the depth buffer method is the A-buffer method.
- A-buffer method represents an antialiased, area-averaged, accumulation-buffer method.
- It expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.
- More than one surface intensity can be taken into consideration at each pixel position and object edges can be antialiased.
- Each position in the A-buffer has 2 fields.

  (1) Depth field-stores a positive or negative real number.

  (2) Intensity field-stores surface-intensity information or a pointer value.
- If the **depth field is positive**, the number stored at that position in the depth of a single surface overlapping the corresponding pixel area.
- The intensity field stores the RGB components of the surface color at that point and the percent of pixel coverage.
- If the **depth field is negative**, it indicates multiple surface contributions to the pixel intensity.
- The intensity field stores a pointer to a linked list of surface data.
- The data for each surface in the linked list includes :
  1. RGB intensity components.
  2. Opacity parameter.
  3. Depth.
  4. Percent of area coverage.
  5. Surface identifier.
  6. Other surface rendering parameters.
  7. 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, the intensity of each pixel is calculated as an average of the contributions from the overlapping surfaces.

SCANLINE METHOD

- A Scan line method of hidden surface removal is an another approach of ***image space method***.
- This method deals with more than one surface.
- As each scanline is processed, it examines all polygon surfaces intersecting that line to determine which are visible.
- It performs the depth calculation and finds which polygon is nearest to the view plane.
- Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.
- The scanline algorithm maintains active edge list.
- The active edge list contains only edges that cross the current scan line, sorted in order of increasing x.
- The scanline method of hidden surface removal also stores 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.
- Scanlines are processed from left to right.
- At the leftmost boundary of the surface, the surface flag is ***turned ON***, and at the rightmost boundary, it is ***turned OFF***.

The Figure.5.17 illustrates the scanline method for hidden surface removal. Here, the active edge list for scanline 1 contains the information for edges AD, BC, EH, and FG.

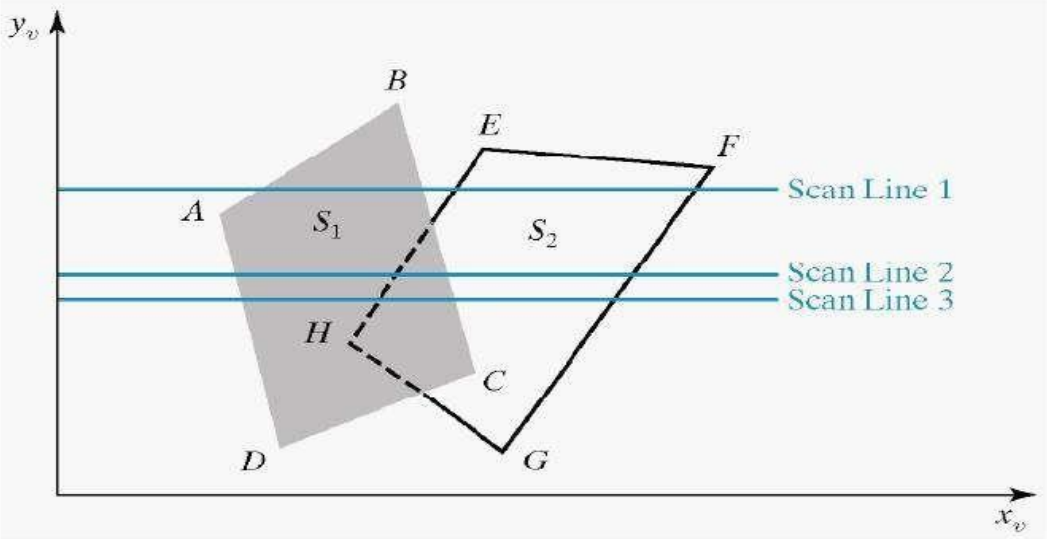


Figure .5.17 Scanlines crossing the projection of the surfaces

- For the positions along this scan line between edges AD and BC, only the flag for surface S1 is ON.
- Therefore, no depth calculations are necessary and intensity information for flag for surface S2 is ON and during that position of scan line the intensity information for surface S2 is entered into

- the frame buffer.

- For ***scan line 2***, the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S1 is ON.
- Between edges EH and BC, the flags for both surfaces are ON. In this portion of scanline2, the depth calculations are necessary.
- Here we have assumed that the depth of S1 is less than the depth of S2 and hence the intensities of surface S1 are loaded into the frame buffer.
- Then, for edge BC to edge FG portion of scan line 2 intensities of surface S2 are entered into the frame buffer because during that portion only, flag for S2 is ON.