As Per New Revised Credit System Syllabus

DBATU

Final Year B.TECH. Semester - VII
Course in COMPUTER ENGINEERING

COMPUTER GRAPHICS

(OPEN ELECTIVE – X BTCOE 704(B))

Dr. S. A. CHIWHANE

Mrs. P. P. AHIRE

w m
n line.com
w iralibooks
@nirali.prakasnan

NIRALI PRAKASHAN
ADVANCEMENT OF KNOWLEDGE

SYLLABUS

Unit 1 : Basic Concepts **[6 Hrs]**

Introduction to computer graphics, lines, line segments, pixels and frame buffers, anti-aliasing techniques and character generation methods. Graphics Display devices (monochrome, color) interactive devices, Scanners and digitizers, touch panels, tablets, mouse, joysticks, trackball, light pen.

Unit 2 : 2D Transformation **[7 Hrs]**

Line and circle plotting using Breshemham's and other algorithms, transformation matrices, scaling, rotation, translation, picture transformation, mirror image. Window and Clipping: Introduction, viewing transforms, 2-D clipping, Sutherland Cohen approach, Cyrus Beck Method, Midpoint subdivision algorithm, Liang-Barsky line clipping algorithm, polygon clipping, text clipping, generalized clipping.

Unit 3 : 3D Graphics **[7 Hrs]**

Introduction, 3-D geometry, Coordination system, 3D transformation, rotation about an arbitrary axis, orthogonal projections, multiple views, isometric projection, perspective projections, 3-D clipping. Hidden Surfaces and Lines: Introduction, Back face removal algorithm, Z-buffers, Scan line and Painter's algorithm hidden surface removal, curved surface generation, generation of solids, sweep method, interpolation.

Unit 4 : Graphical User Interface **[6 Hrs]**

X-Windows, use of graphics tools like OPENGL, DirectX, Windows and Motif, Graphic Standards.

Unit 5 : Animation **[6 Hrs]**

Introduction, devices for producing animation, computer assisted animation, real time animation, method for controlling animation (fully explicit control, procedural).

CONTENTS

UNIT I

BASIC CONCEPTS

1.1 INTRODUCTION

- In 1950, the first computer-driven display attached to MIT's Whirlwind I computer was used to generate simple pictures. This display made use of Cathode-Ray Tube i.e. CRT similar to the one used in TVs. Previously, CRT was used as an information storage device by F. Williams.
- During the 1950's interactive computer graphics made few processes since the computer of that period were very unsuited for interactive use. Only towards the end of the decade, with the development of machines like MIT's TX-0 and TX-2 did interactive computing become feasible and interest in computer graphics then began to increase rapidly.
- By the mid 1960's, large computer graphics research projects were underway at MIT, General Motor, Bell Telephone Laboratories etc.
- The 1960's represent the heavy years of computer graphics research, the 1970's have been the decade in which this research began to bear fruit. Interactive graphics display is now use in many countries and are widely used for educational purposes. The instant appeal of computer graphics to users of all ages has helped it to spread into many applications.

1.1.1 Definition of Computer Graphics

- Computer graphics is the creation and manipulation of images/pictures with the aid of computer.

  There are two types of computer graphics :

  1. **Passive CG :** In this, the observer has no control over the image.
  2. **Active CG :** In this, the observer has control over the image.
- Computer graphics is a study of techniques to improve communication between human and machine. The major product of computer graphics is a picture. With the help of computer graphics pictures can be represented in 2-D as well as in 3-D space.

1.1.2 Importance of Computer Graphics

- The importance of computer graphics in computer science can be understood by the following points :
  - ➢ The electronic industry is more dependent on the technology. Engineers can draw the circuit in a much shorter time.
  - ➢ It provides tools for producing pictures not only of concrete, "real-world" objects but also of abstract, synthetic objects, such as mathematical surfaces in 3D and of data that have no inherent geometry, such as survey results.
  - ➢ It has an ability to show, moving pictures and thus, it is possible to produce animations with computer graphics.
  - ➢ With computer graphics, user can also control the animation of adjusting the speed, the portion of the total scene in view, the geometric relationship of the objects in the scene to one another, the amount of detail shown and so on.
  - ➢ The use of computer graphics is wide spread. It is used in various areas such as industry, business, government organizations, education, entertainment and most recently the home.

1.2 FRAME BUFFER

- In a raster scan system, for drawing a picture on screen, the electron beam is swept across the screen one row at a time from top to bottom. When the electron beam sweeps across each row, the beam intensity is turned ON and OFF to create a pattern of illuminated spots. The information about these spots is stored in a memory area known as '**Frame Buffer**'. The frame buffer holds the set of intensity values for all the screen points. The intensity values stored in a frame buffer are then retrieved and pointed on a screen one row at a time. This row is called as **scan line**. Each point on a screen is called as **pixel**. The pixel stands for picture element.

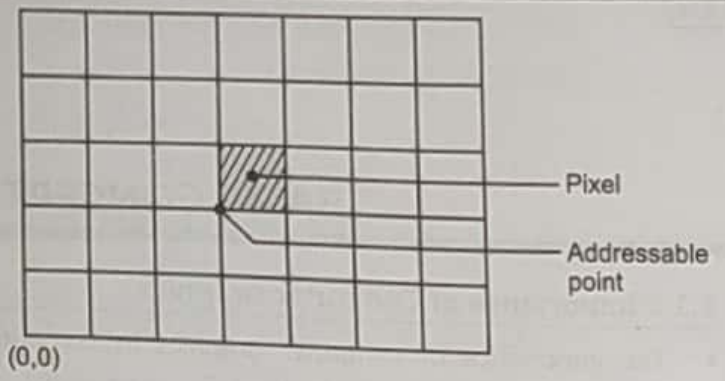
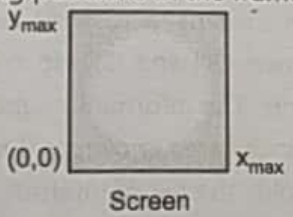


Fig. 1.1

- A pixel is addressed or identified by its lower left corner. The pixel occupies a finite area to the right and above this point. The addressing always starts at (0, 0).
- The pixel can be defined as the smallest addressable screen element. Every pixel possess a name or address. It is the smallest piece of display screen under control. The name of each pixel corresponds to the co-ordinates, which identify the points. By setting the intensity and colour of pixels, which composes screen, graphic images can be made. Line segments can be drawn by setting the intensities. The display screen can be thought of as a grid or array of pixels or matrix of discrete cells. For drawing a picture on the screen, the intensity values of pixels are placed in an array in the memory. This array is nothing but a frame buffer. Then the display devices access this frame buffer array for determining the intensity of each pixel, which is to be displayed.
- In graphic programming using 'C' languages, the put pixels function is used to plot or display a pixel.

  Example : Put pixel (100, 200, 5).
- In the above example, X-co-ordinate is 100, Y co-ordinate is 200 and third number i.e. 5 denotes the colour number. The put pixel function converts within the raster and stores intensity values at the corresponding positions in the frame buffer array.



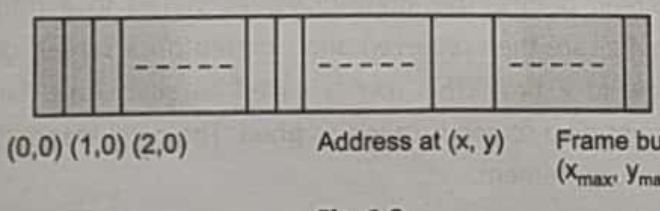


Fig. 1.2

- On a black and white system with one bit per pixel the frame buffer is commonly called as bitmap. For system with multiple bits per pixel, the frame buffer is often referred to as pixmap.

1.2.1 Black and White Frame Buffer

- If the frame buffer stores one bit pixel information then the size of each element of the frame buffer array is one bit. Hence, it is referred as bit plane. A single bit plane can be store only two values 0 and 1, thus it can yield black and white display. A single bit plane, black and white frame buffer, raster CRT graphic device is shown below in Fig. 1.3.

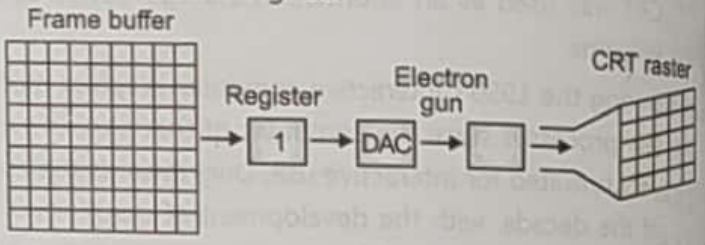
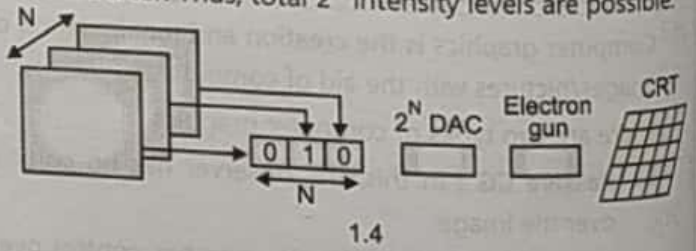


Fig. 1.3

1.2.2 N-Bit Plane Gray Level Frame Buffer

- The below figure 1.4 shows an N-bit plane gray level frame buffer. The colour of gray levels are incorporated in the screen of frame buffer raster graphics device using additional bit planes, on the CRT the intensity of each pixel is controlled by a corresponding pixel location in each of the N-bit planes. The binary value i.e. 0 or 1 frame each of the N-bit plane is stored in the register, which is interpreted as an intensity level between 0 and 2^N where 0 represents dark and 2^N represents full intensity. This is then converted into analog voltage by using DAC i.e. Digital to Analog Converter. Thus, total 2^N intensity levels are possible.



1.4

Fig. 1.4

Look-Up Table and N-Bit Plane :

- By using look-up table the number of available intensity levels are increased for achieving the modest increase in the memory. After reading the bit planes in the frame buffer, the resulting number is used as an order into the look-up table. The look-up table has 2^N entries, each of which is w-bit wide. The look-up table

can be changed or reloaded to get additional intensities.

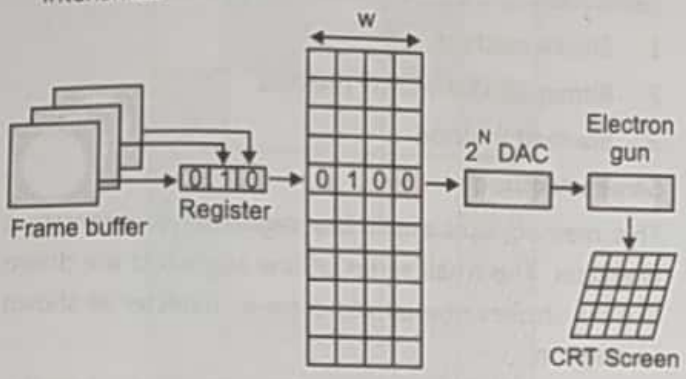


Fig. 1.5

1.2.3 Colour and 3-Bit Planes

- As shown in below figure 1.6, there are three primary colours. Thus, a simple colour frame buffer is implemented with 3-bit planes, one for each primary order. The bit plane drives an individual colour gun for each of the three primary colours in colour video. These three colours can produce $2^3 = 8$ different colours.

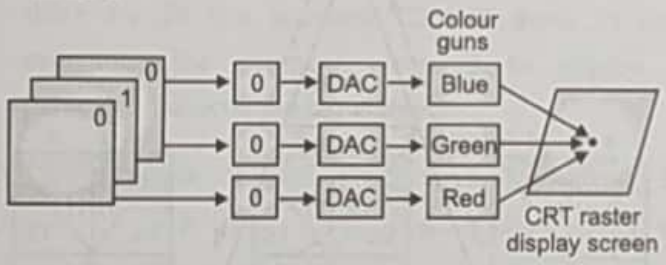


Fig. 1.6

	R	G	B
Black	0	0	0
Red	1	0	0
Green	0	1	0
Blue	0	0	1
Cyan	0	1	1
Yellow	1	1	0
White	1	1	1
Magneta	1	0	1

1.3 VECTORS

- Vectors can be defined as a directed line segment which possess magnitude as well as direction.
- If two points P_1 and P_2 are given then vector V is defined as the difference between the two point positions.

For 2-Dimension :

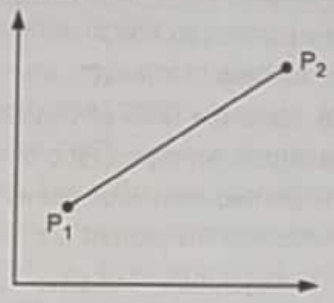


Fig. 1.7

$$V = P_2 - P_1$$
$$= (x_2 - x_1, y_2 - y_1)$$
$$= (V_x, V_y)$$

$V_x, V_y \rightarrow$ Projections of V on x and y axes.

$$|V| = \sqrt{V_2^x + V_y^2}$$

- Thus, when we consider a line segment, it has a fixed position in space. But vector does not have a fixed position in space. The vector does not tell us the starting point. It tells how far to move and in which direction.

1.4 LINE

- Two points would represent a line or edge. If the two points used to specify a line are (x_1, y_1) and (x_2, y_2) then the equation for the line is given as,

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$\therefore \quad y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$

or $y = mx + h$

where, $m = \frac{y_2 - y_1}{x_2 - x_1}$

and $b = V_1 - mx_1$

- The above equation is called the slope intercept form of the line. The slope m is the change in height $(y_2 - y_1)$ divided by the change in width $(x_2 - x_1)$ for two points on the line. The intercept b is the height at which the line crosses the y-axis.
- **Line Segment:** A line segment is a part of a line that is bounded by two distinct end points and contains every point on the line between its endpoints.

1.5 SOME IMPORTANT TERMS

- **Persistence :** It is defined as the time taken by the emitted light from the screen to decay to one-tenth of its original intensity.

- **Raster Scan Display :** In this, the beam is moved all over the screen one scan line at a time, from top to bottom and then back to top.
- **Aspect Ratio :** It is the ratio of vertical resolution to horizontal resolution. An aspect ratio of 4/5 means that a vertical line plotted with four points has the same length as a horizontal line plotted with five points.
- **Resolution :** Resolution indicates the maximum number of points that can be displayed without overlap on the CRT. It is defined as the number of points per centimeter that can be plotted horizontally and vertically.
- **Horizontal Sweep Frequency :** It is related to the number of scan lines per second. It gives the information about how much time is needed to scan one line.
- **Vertical Sweep Frequency :** It is related to the number of frames covered by the electron beam in one second.
- **Horizontal Scan Line :** It is the time duration of frequency of electron beam to move from left corner to right corner of screen or scan line.
- **Vertical Scan Line :** It is the time duration of an electron beam to move from upper left corner to bottom right corner.
- **Fluorescence :** The glow given off by the phosphor during the excitation period of the electron beam is called as fluorescence.
- **Phosphorescene :** The glow given off by the phosphor after the electron beam is removed. In other words the phosphorescence is time of fluorescence.
- **Horizontal Refresh Rate :** It is the time duration of electron beam to come from the right end of the scan line to the left end of another scan line.
- **Vertical Refresh Rate :** It is the time required for the electron beam to move from bottom right corner of scan line to the starting left corner of scan line.

1.6 CHARACTER GENERATION

- Along with lines and points, strings of characters are often displayed to label, draw and to give instructions and information to the user. Characters are always built into the graphics display device usually as hardware but sometimes through software.
- There are three primary methods for character generation :
  1. Stroke method
  2. Bitmap or Dot-matrix method
  3. Starbust method

1. Stroke Method :

- This method uses small line segments to generate a character. The small series of line segments are drawn like the strokes of a pen to form a character as shown in diagram.
- We can build our own stroke method character generator by calls to the line drawing algorithm. Here, it is necessary to decide which line segments are needed for each character and then drawing these segments using line drawing algorithm we can draw characters on the display. The stroke method supports scaling of the character. It does this by changing the length of line segments used for character drawing.

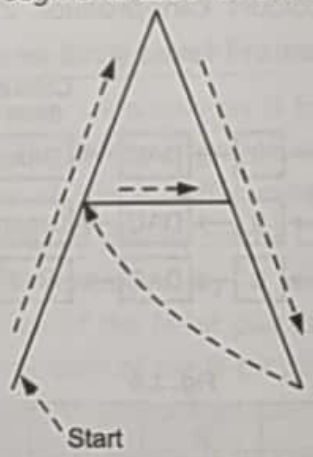


Fig. 1.8 : Stroke Method

2. Bitmap/Dot-Matrix Method :

- This method is used for character generation. It is also called the dot-matrix form.
- As shown in Fig. 1.9, a two dimensional array having columns and rows. A 5×7 array is commonly used to represent character shown in a diagram.
- However, 7×9 and 3×13 array are also used. Higher resolution devices such as inkjet printer or lazer printer may use character arrays over 100×100.
- Each dot in the matrix is a pixel. The character is placed on the screen by copying pixel values from the character array into some portion of the screen's frame buffer. The value of the pixel, controls the intensity of the pixel.

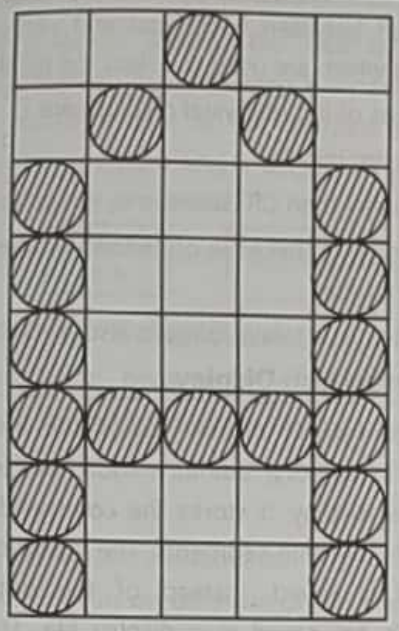


Fig. 1.9 : Character A in 5 × 8 Dot Matrix Format

3. Starbust Method :

- In this method, a fixed pattern of line segments is used to generate characters. As shown in the figure 1.10 there are 24 line segments. Out of these 24 line segments, the segments required to display a particular character are highlighted.

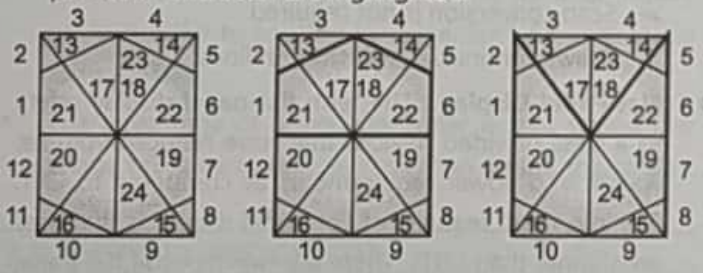


Fig. 1.10

1.7 DISPLAY DEVICES

- The display devices used in graphics are based on CRT technology. The various display devices are :
  1. DVST.
  2. Raster Refresh Graphic Display.
  3. Calligraphic Refresh Graphic Display.
  4. Plasma Panel Display.
  5. Colour CRT Monitors.
  6. Flat Panel Display.
  7. Liquid Crystal Monitors.
  8. Vector Scan Display.

1.7.1 DVST (Direct View Storage Tube)

- Conceptually, the DVST is the simplest of all CRT displays. These types of displays are no longer manufactured. The DVST contains long-persistence phosphor. The DVST can also be referred as bistable storage tube. Since the long-persistence phosphor is used hence the line or character remains visible for longer period i.e. upto one hour until erased. The electron beam intensity is increased for drawing a character or line on the display. As the intensity increases the phosphor assume its bright storage state. By flooding the entire tube with a specific voltage, the display is erased. Erase takes approximately half second. The electron beam is intensified to a point, which is just below the threshold that will cause the permanent storage and is also sufficient to brighten the phosphor.
- The features of DVST are as follows :
  - ➢ The DVST is easy to program.
  - ➢ It has flat screen.
  - ➢ The display is flicker free.
  - ➢ Refreshing is not required.
  - ➢ The hard copy produced by DVST is relatively easy, fast and inexpensive.
  - ➢ It is not suitable for animation and dynamic motion.

1.7.2 Raster Refresh Graphics Display

- The most common graphics display employing CRT is the raster scan display which is based on the television technology. Both the storage tube CRT display and the random scan refresh display are line drawing devices. The straight lines can be drawn directly from one addressable point to other addressable point. This device is a matrix of discrete cells in which each of the cells can be made bright. Thus, it is a point plotting device. Only in some several cases, it is possible to draw a straight line.
- The features of Raster refresh graphic display are :
  - ➢ Cost is low.
  - ➢ It has ability to display areas filled with solid colours.
  - ➢ Refresh process is independent of the complexity of image.
  - ➢ It controls the intensity of each dot and pixel in a rectangular matrix.

1.7.3 Calligraphic Refresh Graphic Display

- A calligraphic refresh CRT display makes the use of a very short persistence phosphor. Due to this the picture painted on the CRT must be repainted or refreshed many times in one second. The refresh rate is at least 30 times each second with a recommended rate of 40-50 times each second. The image gets flickered due to lower refresh rate.
- The features of Calligraphic refresh graphic display are:
  - ➢ The speed of communication is slow.
  - ➢ Image gets flickered.
  - ➢ The instruction pertaining to image are stored in buffer, thus selective modification is possible.
  - ➢ Generation of solid figure is difficult.
  - ➢ The calligraphic display needs two elements in addition to the Cathode Ray Tube (CRT). They are display buffer and display controller.

1.7.4 Plasma Panel Display

- A display device which stores the image but allows selective erasing is the plasma panel. It contains a gas at low pressure sandwiched between horizontal and vertical grid of fine wires. A large voltage difference between a horizontal and vertical wire will cause the gas to glow as it does in a neon street sign. A lower voltage will not start a glow but will maintain a glow once started. To set a pixel, the voltage is increased momentarily on the wires that intersect the desired point. To extinguish a pixel, the voltage on the corresponding wire is reduced until the glow cannot be maintained.
- The features of plasma panel display are :
  - ➢ They are very durable.
  - ➢ They have been used in PLATO educational system.
  - ➢ Less bulky than CRT.
  - ➢ Refreshing is not required.
  - ➢ Often used in military application.
  - ➢ Produces a very steady image, totally flicker free.
- **Liquid Crystal Monitor :** It is a very economical device. It is a flat panel display technology, hence less bulky than CRTs. In liquid crystal display, light is either transmitted or blocked, depending upon the orientation of molecules in the liquid crystal. An electrical signal can be used to change the molecular orientation, turning a pixel on or off. The material is sandwiched between horizontal and vertical grids of electrodes which are used to select the pixel.
- The features of liquid crystal displays are :
  - ➢ It is economical.
  - ➢ Less bulky than CRTs.
  - ➢ It is portable, because of its low voltage and power requirements.
  - ➢ Liquid crystal televisions are also available.

1.7.5 Vector Refresh Display

- The vector refresh display stores the image in the computer's memory, but in a more efficient manner than raster display. It stores the commands necessary for drawing the line segments. The input to the vector generator is saved, instead of the output. These commands are saved in a display file. The lines are drawn using a vector-generating algorithm. This is done on a normal cathode ray tube, so the image quickly fades. In order to present a steady image, the display must be drawn repeatedly. The vector generator must be applied to all the lines in an image fast enough to draw the entire image flicker free.
- The features of vector refresh display are :
  - ➢ Scan conversion is not required.
  - ➢ Draws continuous and smooth lines.
- **Flat Panel Display :** The term flat-panel display refers to a class of video devices that have reduced volume, weight and power requirements as compared to CRT. The important features of flat-panel display is that they are thinner than CRTs. There are two types of flat panel displays, emissive displays and non-emissive displays.
- **Emissive Displays :** They convert electrical energy into light energy. Plasma panels, thin-film electro luminescent displays and light emitting diodes are examples of emissive displays.
- **Non-Emissive Displays :** They use optical effects to convert sunlight or light from some other source into graphics patterns. Liquid crystal display is an example of non-emissive flat panel display.

1.8 DISPLAY FILE INTERPRETER

- The display file contains information necessary to construct the picture. This information is in the form of commands. The process to convert these instructions into a actual images is performed by a processor called as display file interpreter.

- In some graphical system the task of display file interpreter is performed by using a separate computer or CPU. Hence, it acts as an interface between the graphics user program and a display service, which is known as **display processor**.

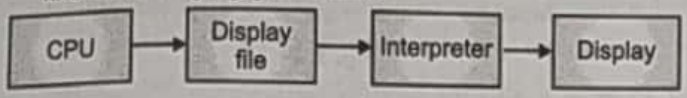


Fig. 1.11

- Every instruction possess a MOVE, LINE or PLOT commands. As shown in above Fig. 1.11 the interpreter executes the instruction and the output will be a visual image.
- A graphic program written from a particular display device will not run on different display device. Hence, the program is not possible. On the other hand, if a program has been written, generates display code then only an interpreter is needed for each device, which converts the standard display instruction into the actions of a particular device. Thus, the display device and its interpreter can be thought of as a machine on which any standard program can run. The display file instructions are actually saved in a file for later display or for transfer to another machine. Such files are called as **metafiles**.
- The advantage of using interpreter is that saving raw image takes much less storage than saving the picture itself.

1.9 DISPLAY PROCESSOR

Opcode	Command
1	Move
2	Line

- As operand is having both x and y parameters we have to separate them. We will store the x co-ordinate in x operand and y co-ordinate in y operand.
- In some graphics system, a separate computer is used to interprete the commands in the display file. Such computer is called as a **display processor**. Display processor access display file and it cycles through each command in the display file once during every refresh cycle. Fig. 1.12 shows the vector scan system with display processor.

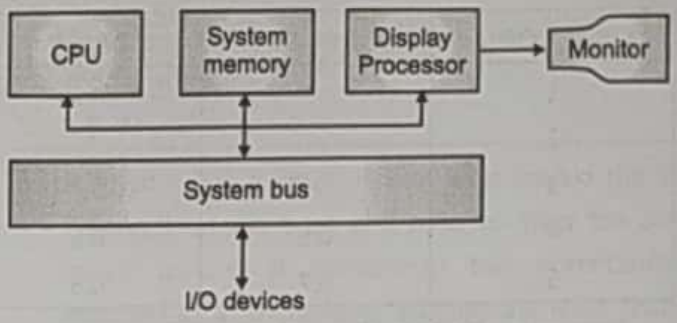


Fig. 1.12 : Vector Scan System

- In the raster scan display systems, the purpose of display processor is to free the CPU from the graphics routine task. Here, display processor is provided with separate memory. The main task of display processor is to digitize 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 known as **scan conversion**.

1.10 DISPLAY FILE STRUCTURE

- Every command of display file consists of two parts :
  1. Operation code (OPCODE).
  2. Operands.
- Operation code indicates which type of command it is i.e. either LINE or MOVE.
- Operands are the co-ordinate of a point (x, y).
- The display file is nothing but a series of above two instructions. Three separate arrays are used for storing these instructions which are as under :
  1. One array for the operation code i.e. DF – OP.
  2. Second array for the x co-ordinate i.e. DF – X.
  3. Third array for the Y co-ordinate i.e. DF – Y.
- Before processing, it is very necessary to assign meaning to the possible operation codes. Let us consider two possible instructions MOVE and LINE. Let us define an opcode of 2 to a MOVE command and an opcode of 3 to a LINE command. Then command to MOVE to position x = 0.5 and y = 0.6 would be 2, 0.5, 0.6.

  DF – OP [4] ← 2
  DF – X [4] ← 0.5
  DF – Y [4] ← 0.6
- The above statement would be stored in the fourth display file position. Suppose the value of DF – OP [5] is 3 and DF – X [5] = 0.7 and DF – Y[5] = 0.8 then the display would show a line segment from (0.5, 0.6) to (0.7, 0.8) as shown below.

DF – OP	DF – X	DF – Y
1		
2		
3		
4	0.5	0.6
5	0.7	0.8

Instruction of display file.

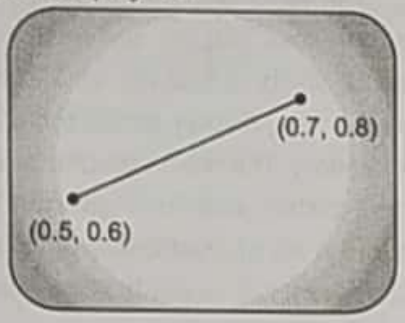


Fig. 1.13

1.11 INTERACTIVE DEVICES/INPUT DEVICES

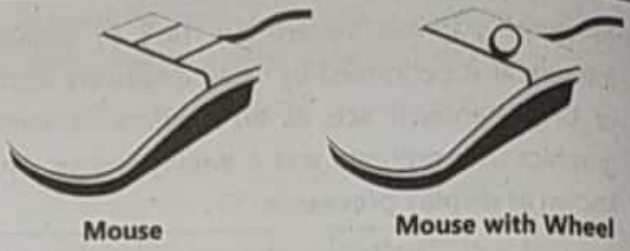
- Input device allows to communicate with computer. Using this one can feed in the information.

Most commonly used input devices are :

1. Mouse
2. Trackball
3. Touch panel
4. Light pen
5. Joystick
6. Tablets
7. Keyboard.
8. Scanner.

1.11.1 Mouse

- A mouse is a palm-sized box used to position the screen cursor. It consists of ball on the bottom connected to wheels or rollers to provide the amount and direction of move. One, two or three buttons are usually provided on the top of the mouse for signaling the execution of some operation. Now-a-days, mouse consists of one more wheel on the top to scroll the screen pages.
- A different type of mouse is optical mouse. Here, the ball is replaced by two apertures, one for optical source and other for receiving the optical ray reflected from the metallic mouse-base plate.



Mouse Mouse with Wheel

Fig. 1.14

1.11.2 Trackball

- Trackball can be seen as an inverted mouse where the ball is held inside a rectangular box. As the name implies, a trackball is a ball that can be rotated with the fingers or palm of the hand to produce screen cursor movement. The potentiometers attached to the trackball are used to measure the amount and direction of rotation.
- The trackball is a two dimensional positioning device whereas spaceball provides six degree of freedom. It does not actually move. It consists of strain gauges which measures the amount of pressure applied to the space ball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. It is usually used in three-dimensional positioning and selecting operations in virtual-reality systems.

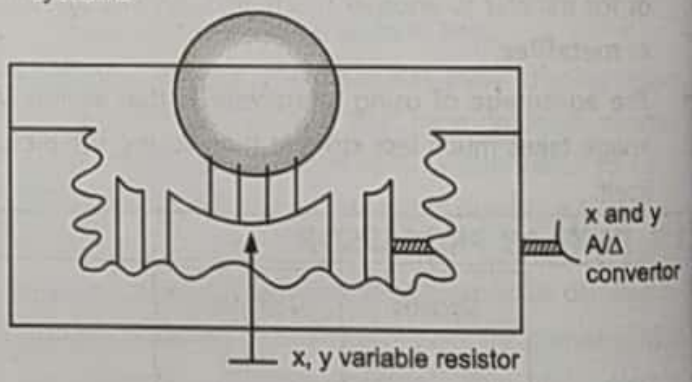


Fig. 1.15

1.11.3 Touch Panel

- Touch panels allow display objects or screen positions to be selected with a touch of finger. It is used in the selection of processing options that are represented with a graphical icons. Touch panels are of various types :

1. Optical Touch Panel

- Optical touch panels consists of a line of infrared light emitting diodes (LEDs) along one vertical edge and along one horizontal edge of the frame. The opposite vertical and horizontal edge contains light detectors

These detectors are used to record which beams are interrupted when the panel is touched.

2. **Electrical Touch Panel**

- An electrical touch panel is constructed with two transparent plates separated by a small distance. When the outer plate is touched, it is forced into contact with the inner plate. This contact creates a voltage drop across the resistive plate that is converted into the co-ordinate values of the selected screen position.

3. **Acoustical Touch Panel**

- Acoustical touch panels, uses high frequency sound waves to determine the point of contact.

1.11.4 Light Pen

- A light pen is a pointing device shaped like a pen acts as a computer input device. The tip of the light pen contains a light sensitive element which, when placed against the screen, detects the light from the screening enabling the computer to identify the reaction of the pen on the screen.
- Light pen wave the advantage of 'drawing' directly onto the screen. It allows the user to point to displayed objects, or drawn on the screen, in a similar way to a touch screen.

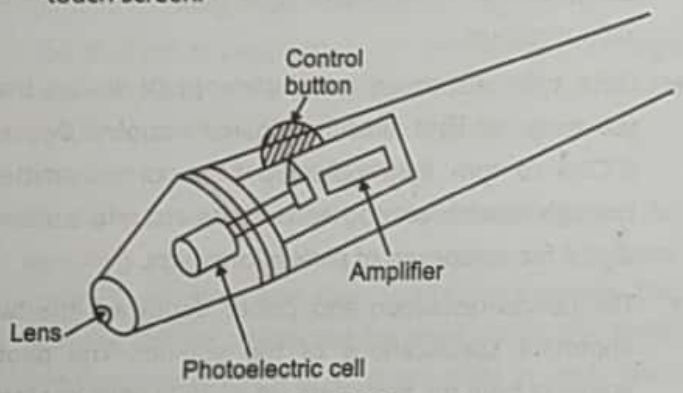
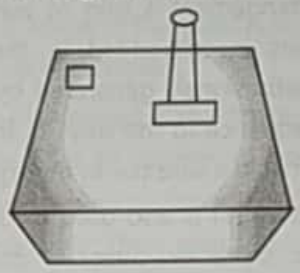


Fig. 1.16

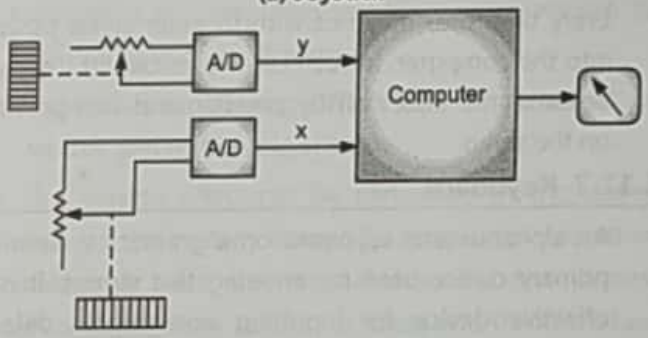
- It allows the user but with greater positional accuracy. A light pen can work with any CRT-based monitor, but not with LCD screens, projectors or other display devices.
- The light pen consist of a photoelectric cell housed in a pen like case. It works by sensing the sudden small change in brightness of a point on the screen when the electron gun refreshes that spot.
- By noting exactly where the scanning has reached at that moment, the x, y position of the pen can be resolved.
- For slow displays transistor type photo-cells such as diodes are used.

1.11.5 Joystick

- A joystick has a small vertical lever (called the stick) mounted on the base and used to steer the screen cursor around. It consists of two potentiometers attached to a single lever. Moving the lever changes the settings on the potentiometer. The left or right movement is indicated by one potentiometer and forward or back movement is indicated by other potentiometer. Thus, with a joystick both x and y co-ordinate positions can be simultaneously altered by the motion of a single lever as shown below.



(a) Joystick



(b) Internal Details of Joystick

Fig. 1.17

- Some joystick may return to their zero (center) position when released. Joystick are inexpensive and are quite commonly used where only rough positioning is needed.

1.11.6 Tablet

- For applications, such as tracing we need a device called a digitizer or a graphical tablet. It consists of a flat surface ranging in size from about 6 by 6 inches upto 48 by 72 inches or more which can detect the position of movable styles.
- The Fig. 1.18 below shows a small tablet with pen like stylus.

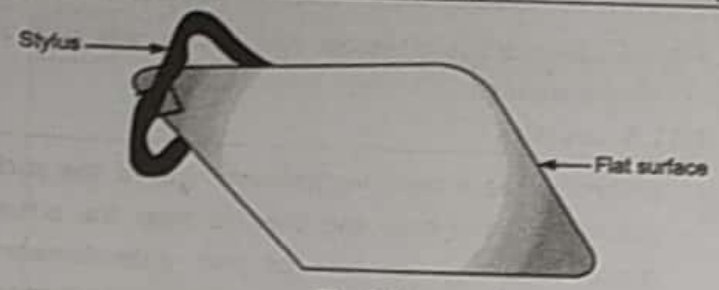


Fig. 1.18

- Different graphics tablets use different techniques for measuring position, but they all resolve the position into a horizontal and a vertical direction, which corresponds to the axes of the display.
- Most graphics tablets use an electrical sensing mechanism to determine the position of the stylus. In one such arrangement a grid of wire on 1/4 to 1/2 inches centers is embedded in the tablet surface. Electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induce an electrical signal in a wire coil in the stylus.
- The signal strength is also used to determine roughly how far the stylus or cursor is from the graphical tablet.
- Every time user may not wish to enter stylus position into the computer. In such cases user can lift the stylus or make the tablet off by pressing a switch provided on the stylus.

1.11.7 Keyboard

- An alphanumeric keyboard on a graphics system is a primary device used for entering text strings. It is an effective device for inputting non-graphic data as picture labels associated with a graphic display. Keyboards can also be provided with features to facilitate entry of screen co-ordinates, menu selections or graphic functions.
- General purpose keyboards contain cursor-control keys and function keys. Cursor-control keys can be used to select displayed objects or co-ordinate positions by positioning the screen cursor. The function keys are used to enter frequently used operations in a single keystroke. Numeric keypad is also present for inputting the numeric data.

1.11.8 Scanner

- The scanner is a device, which can be used to store graphs, photos or text available in printed form for computer processing.
- The scanners use the optical scanning mechanism to scan the information. The scanner records the gradation of gray scales or colour and stores them in the array.
- Finally, it stores the image information in a specific file format such as JPEG, GIF, TIFF, BMP and so on.
- Once the image is scanned, it can be processed or we can apply transformations to rotate, scale or crop the image using image processing softwares such as photo-shop or photo-paint. Scanners are available in variety of sizes and capabilities.

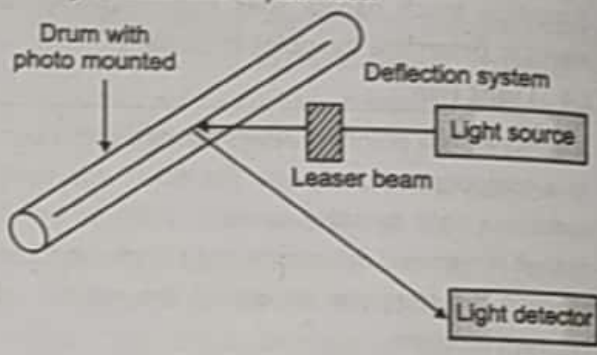


Fig. 1.19 : Photoscanner

- Above Fig. 1.19 shows the working of photo scanner.
- For coloured photographs, multiple passes are made, using filters in front of the photocell to separate out various colours.
- Other type of scanners are electro-optical devices that use arrays of light sensitive Charge Coupled Devices (CCDs) to turn light reflected from, or transmitted through artwork, photographs, slides etc. into a usable digital fire composed of pixel information.
- The optical resolution and colour depth are the two important specifications of the scanner. The photo scanners have the resolution up to 2000 units per inch. Resolution of the CCD array is 200 to 1000 units per inch which is less than the photo scanner. The colour depth is expressed in bits. It specifies the number of colours a scanner can capture.
- The scanners can also be classified into following types as per construction :
  - ➢ Flat-bed Scanner
  - ➢ Drum Scanner
  - ➢ Sheet-fed Scanner
  - ➢ Handheld Scanner.

1.11.9 Aliasing and Antialiasing

- The images which are different but posses same graphical representation are called as **alias** and phenomenon is referred as **aliasing**.
- The main cause of aliasing is the finite size of pixel. If the size of pixel get reduced then the effect of aliasing will also be reduced. The size of pixel depends upon hardware resolution. Thus, for the reduction of aliasing effect and for the improvement of visual resolution the software method is used, which is referred as **antialiasing**.
- For effective antialiasing, it is necessary to understand the reason of aliasing. The appearance of aliasing effect is due to the lines, polygon edges, and colour boundaries etc. which are continuous while a raster device is discrete. For plotting line, polygon edge etc. on the raster display device, it must be sampled at discrete locations. This will have a very unexpected result.

The two important antialiasing techniques are :

1. Super sampling or post filtering or averaging.
2. Area sampling or prefiltering.

1. Super Sampling/Post Filtering/Averaging :

- The method of sampling object characteristics at high resolution and at the same time displaying the results at low resolution is referred to as super sampling or post filtering or averaging method.
- This is a straight forward technique to increase the sampling rate by treating the screen as if it were covered with time grid then is actually available. The multiple sample points can be used across this timer grid for determination of an appropriate intensity level for each screen pixel.
- This technique can be implemented in two ways :

(i) Uniform averaging.

(ii) Averaging using weights.

2. Area Sampling/Prefiltering :

- In polygon filling and line rasterization algorithms, the intensity or colour of a pixel is identified according to the intensity or colour of a single point within the pixel area. In this method, it is assumed that the pixel is a mathematical point rather than a finite area. In area, antialiasing method, the pixel is treated as a finite area.
- In an ideal primitive the line should have zero width, but here the line has non-zero width. Hence, the line can be thought of as a rectangle of a desired thickness, which covers a portion of grid as shown in Fig. 1.20.

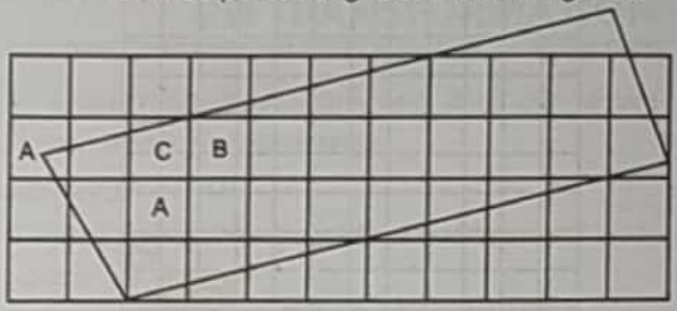
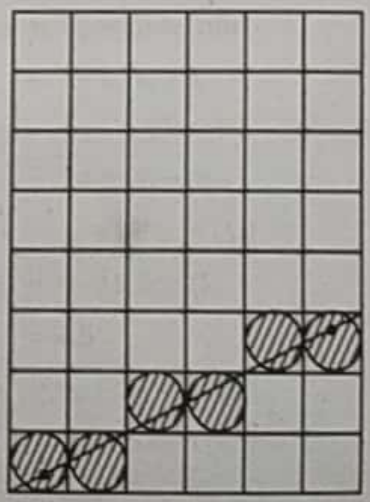


Fig. 1.20

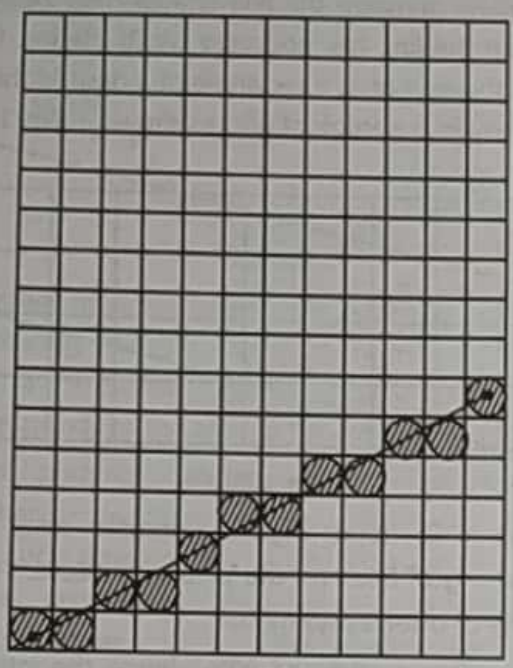
- As shown in above figure, consider pixel A. The overlapping of line and the pixel A is 100%. Thus, the intensity of pixel is maximum.
- The pixel B is overlapped 80%. Hence, the intensity of pixel would be 80% of maximum intensity. In similar manner the overlapping region of pixel C and D is 50% and 10% respectively. Thus, their intensities should be 50% and 10% of maximum intensity respectively. Antialiasing by computing overlap area is referred as Area sampling or prefiltering.

Q.1. Explain how aliasing effect can be removed in vector generation algorithm.

Ans. The aliasing effect can be minimized by increasing resolution of the raster display. By increasing resolution and making it twice the original one, the line passes through twice as many column of pixels and therefore has twice as many jags, but each jag is half as large in x and in y direction.



(a)



(b)

Fig. 1.21 : Effect of Aliasing with Increasing in Resolution

- As shown in Fig. 1.21, line looks better in twice resolution, but this improvement comes at the price of quadrupling the cost of memory, bandwidth of memory scan-conversion time. Thus, increasing resolution is an expensive method for reducing aliasing effect.
- With raster systems that are capable of displaying more than two intensity levels (colour or gray scale), we can apply antialiasing methods to modify pixel intensity. By appropriately varying the intensities of pixels along the line or object boundaries, we can smooth the edges to lessen the stair-step or jagged appearance.

EXERCISE

1. Write short note on scanner, joystick, touch panel, mouse.
2. Explain Frame Buffer and its Types.
3. Describe display devices.
4. What is Anti-aliasing.
5. Describe the methods for character Generation.
6. Define Line, Line Segment , Pixel and Frame Buffer.
7. Explain display file structure .
8. Explain Aliasing and Antialiasing.

✠ ✠ ✠

2.1 LINE DRAWING ALGORITHM

2.1.1 DDA Line Drawing Algorithm

The digital differential analyzer is a scan-conversion line algorithm based on calculating either dy or dx. We sample the line at unit intervals in one coordinate and determine corresponding integer values nearest to the line path for the other coordinate.

Algorithm :

Step 1 : Read the end points of the line (x_1, y_1) and (x_2, y_2) such that they are not equal.

Step 2 : If abs $(x_2 - x_1) \geq$ abs $(y_2 - y_1)$.

then length = abs $(x_2 - x_1)$

else length = abs $(y_2 - y_1)$

Step 3 : Initialize Δx or Δy to be equal to one raster unit.

$\Delta x = (x_2 - x_1)/length$

$\Delta y = (y_2 - y_1)/length$

Step 4 : Round the values. Sign function is used to make the algorithm work on all quadrants.

$x = x_1 + 0.5$ sign (Δx)

$y = y_1 + 0.5$ sign (Δy)

Step 5 : Plot (x,y)

Step 6 : for(int i=1; i <= length; i++)

```
{
  x = x + Δx
  y = y + Δy
        plot (integer (x), integer (y))
}
```

end

Note : The sign function is used which return 1, 0, −1 for the argument as greater than zero, equal to zero and less than zero respectively.

Advantages of DDA

- The logic is easy to understand.
- The integer arithmetic is involved.
- It is a faster method for calculating pixel position.
- It is the simplest algorithm and does not require special skills for implementation.

Disadvantages of DDA

- Floating point arithmetic in DDA algorithm is still time-consuming.
- Division logic is needed, which switches it towards hardware logic.
- Floor integer values are used in place of normal integer values, which may give different values.
- The algorithm is orientation dependent. Hence end point accuracy is poor.

SOLVED EXAMPLES

Example 2.1 : *Rasterize a line from (0, 0) to (8, 4) using DDA algorithm.*

Solution : The end points of line are (0, 0) and (8, 4).

By evaluating the steps of DDA we have,

$X_1 = 0 \quad y_1 = 0$

$X_2 = 8 \quad y_2 = 4$

If abs $(x_2 - x_1) \geq$ abs $(y_2 - y_1)$.

then length = abs $(x_2 - x_1)$

else length = abs $(y_2 - y_1)$

$\therefore$ abs $(x_2 - x_1) = (8 - 0) = 8$ and abs $(y_2 - y_1) = (4 - 0) = 4$

As abs $(x_2 - x_1) >$ abs $(y_2 - y_1)$ therefore

Length = abs $(x_2 - x_1) = 8$

Now, $\Delta x = (x_2 - x_1)/length$

$= [(8-0)/8] = 1$

$\Delta y = (y_2 - y_1)/length$

$= [(4-0)/8] = 0.5$

$x = x_1 + 0.5$ sign (Δx)

$= 0 + 0.5$ sign (1)

$= 0.5$

$y = y_1 + 0.5$ sign (Δy)

$= 0 + 0.5$ sign (1.5)

$= 0.5$

Tabulating the result for each iteration we get,

I	X	y	Plot
	0.5	0.5	(0,0)
1	1.5	1	(1,1)
2	2.5	1.5	(2,1)
3	3.5	2	(3,2)
4	4.5	2.5	(4,2)
5	5.5	3	(5,3)
6	6.5	3.5	(6,3)
7	7.5	4	(7,4)
8	8.5	4.5	(8,4)

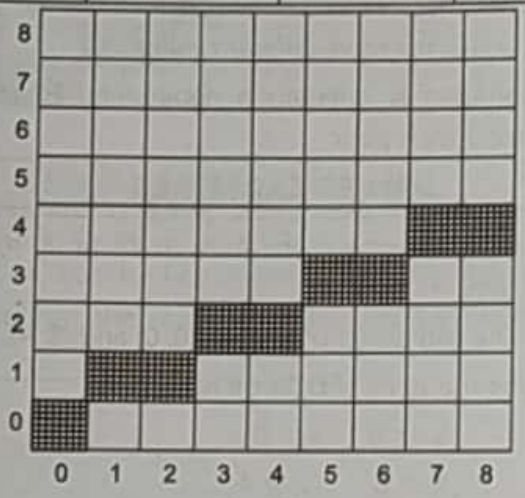


Fig. 2.1

Example 2.2 : *Interprete Digital differential analyzer (DDA) algorithm to find which are the pixels turned on for the line segment between (3,4) and (9,8).*

Solution : The end points of line are (3, 4) and (9, 8).

By evaluating the steps of DDA we have

$X_1 = 3 \quad y_1 = 4$

$X_2 = 9 \quad y_2 = 8$

If abs $(x_2 - x_1) \geq$ abs $(y_2 - y_1)$,

then length = abs $(x_2 - x_1)$

else length = abs $(y_2 - y_1)$

$\therefore$ abs $(x_2 - x_1) = (9 - 3) = 6$ and abs $(y_2 - y_1)$

$= (8 - 4) = 4$

As abs $(x_2 - x_1) >$ abs $(y_2 - y_1)$ therefore

Length = abs $(x_2 - x_1) = 6$

Now, $\Delta x = (x_2 - x_1)/\text{length}$

$= [(9-3)/6] = 1$

$\Delta y = (y_2 - y_1)/\text{length} = [(8-4)/6] = 0.6$

$x = x_1 + 0.5 \text{ sign } (\Delta x)$

$= 3 + 0.5 \text{ sign } (1)$

$= 3.5$

$y = y_1 + 0.5 \text{ sign } (\Delta y)$

$= 4 + 0.5 \text{ sign } (0.6)$

$= 4.5$

Tabulating the result for each iteration we get,

I	X	Y	Plot
	3.5	4.5	(3,4)
1	4.5	5.1	(4,5)
2	5.5	5.7	(5,5)
3	6.5	6.3	(6,6)
4	7.5	6.9	(7,6)
5	8.5	7.5	(8,7)
6	9.5	8.1	(9,8)

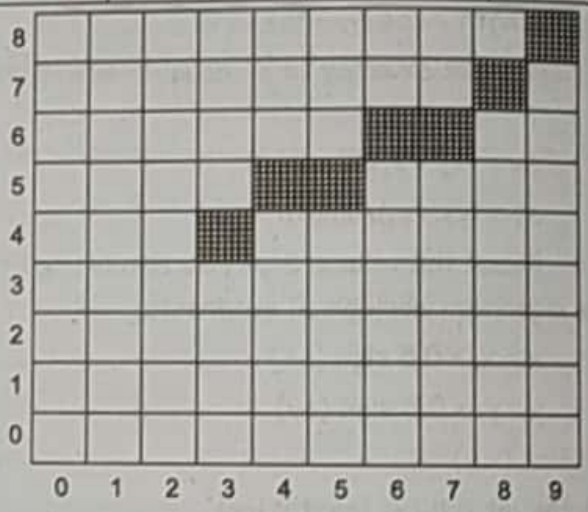


Fig. 2.2

Example 2.3 : *Explain DDA line drawing algorithm. Consider a line segment from A(2, 1) to B(7, 8) Use DDA line drawing algorithm to rasterize this line.*

Solution : For DDA line drawing algorithm explanation refer section no 2.1.1.

The end points of line are (2, 1) and (7, 8).

By evaluating the steps of DDA we have

$X_1 = 2 \quad y_1 = 1$

$X_2 = 7 \quad y_2 = 8$

If abs $(x_2 - x_1) \geq$ abs $(y_2 - y_1)$,

then length = abs $(x_2 - x_1)$

else length = abs $(y_2 - y_1)$

$\therefore$ abs $(x_2 - x_1) = (7 - 2) = 5$ and abs $(y_2 - y_1) = (8 - 1) = 7$

As $abs(y_2 - y_1) > abs(x_2 - x_1)$ therefore

Length = $abs(y_2 - y_1) = 7$

Now, $\Delta x = (x_2 - x_1)/length$

$= [(7-2)/7] = 0.7$

$\Delta y = (y_2 - y_1)/length$

$= [(8-1)/7] = 1$

$x = x_1 + 0.5 \text{ sign } (\Delta x)$

$= 2 + 0.5 \text{ sign } (0.7)$

$= 2.5$

$y = y_1 + 0.5 \text{ sign } (\Delta y)$

$= 1 + 0.5 \text{ sign } (1)$

$= 1.5$

Tabulating the result for each iteration we get,

I	X	y	Plot
	2.5	1.5	(2,1)
1	3.2	2.5	(3,2)
2	3.9	3.5	(3,3)
3	4.6	4.5	(4,4)
4	5.3	5.5	(5,5)
5	6	6.5	(6,6)
6	6.7	7.5	(6,7)

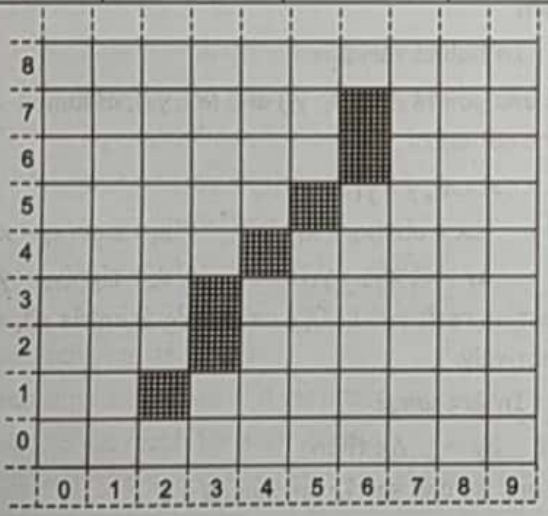


Fig. 2.3

Example 2.4 : *Explain DDA line generation algorithm. Rasterize the line segment with starting point as A(1, 0) and end point as B(5, 7).*

Solution : For DDA line drawing algorithm explanation refer section no 2.1.1

The end points of line are (1, 0) and (5, 7).

By evaluating the steps of DDA we have

$X_1 = 1 \quad y_1 = 0$

$X_2 = 5 \quad y_2 = 7$

If $abs(x_2 - x_1) \geq abs(y_2 - y_1)$,

then length = $abs(x_2 - x_1)$

else length = $abs(y_2 - y_1)$

$\therefore abs(x_2 - x_1) = (5 - 1) = 4$ and $abs(y_2 - y_1) = (7 - 0) = 7$

As $abs(y_2 - y_1) > abs(x_2 - x_1)$ therefore

Length = $abs(y_2 - y_1) = 7$

Now, $\Delta x = (x_2 - x_1)/length$

$= [(5-1)/7] = 0.5$

$\Delta y = (y_2 - y_1)/length$

$= [(7-0)/7] = 1$

$x = x_1 + 0.5 \text{ sign } (\Delta x)$

$= 1 + 0.5 \text{ sign } (0.5)$

$= 1.5$

$y = y_1 + 0.5 \text{ sign } (\Delta y)$

$= 0 + 0.5 \text{ sign } (1)$

$= 0.5$

Tabulating the result for each iteration we get,

I	X	y	Plot
	1.5	0.5	(1,0)
1	2	1.5	(2,1)
2	2.5	2.5	(2,2)
3	3	3.5	(3,3)
4	3.5	4.5	(3,4)
5	4	5.5	(4,5)
6	4.5	6.5	(4,6)

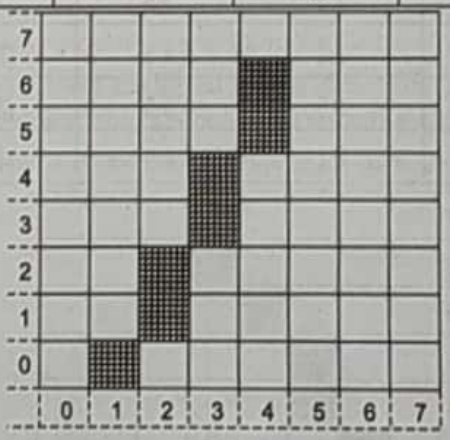


Fig. 2.4

Example 2.5 : *Explain DDA line drawing algorithm. Consider a line segment from A(0,0) to B(4,6). Use DDA line drawing algorithm to rasterize the line.*

Solution : For DDA line drawing algorithm explanation refer section no 2.1.1

The end points of line are (0, 0) and (4, 6).

By evaluating the steps of DDA we have,

$X_1 = 0 \quad y_1 = 0$

$X_2 = 4 \quad y_2 = 6$

If abs $(x_2 - x_1) \geq$ abs $(y_2 - y_1)$.

then length = abs $(x_2 - x_1)$

else length = abs $(y_2 - y_1)$

$\therefore$ abs $(x_2 - x_1)$ = (4-0) = 4 and abs $(y_2 - y_1)$ = (6-0) =6

As abs $(y_2 - y_1)$ > abs $(x_2 - x_1)$ therefore

Length = abs $(y_2 - y_1)$ = 6

Now, $\Delta x = (x_2 - x_1)/\text{length}$

$= [(4-0)/6] = 0.6$

$\Delta y = (y_2 - y_1)/\text{length}$

$= [(6-0)/6] = 1$

$x = x_1 + 0.5 \text{ sign } (\Delta x)$

$= 0 + 0.5 \text{ sign } (0.6)$

$= 0.5$

$y = y_1 + 0.5 \text{ sign } (\Delta y)$

$= 0 + 0.5 \text{ sign } (1)$

$= 0.5$

Tabulating the result for each iteration we get,

I	X	y	Plot
	0.5	0.5	(0,0)
1	1.1	1.5	(1,1)
2	1.7	2.5	(1,2)
3	2.3	3.5	(2,3)
4	2.9	4.5	(2,4)
5	3.5	5.5	(3,5)
6	4.1	6.5	(4,6)

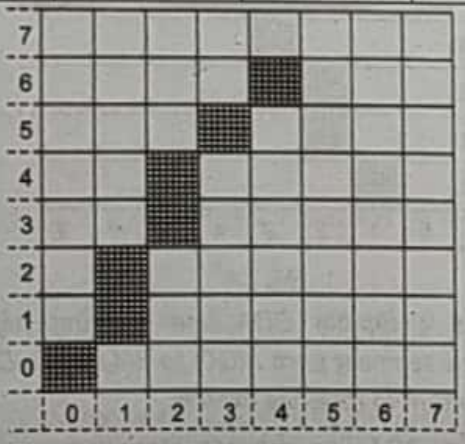


Fig. 2.5

2.1.2 Bresenham Line Drawing Algorithm

- The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that, it uses only integer calculations. Moving across the x axis in unit intervals and at each step choose between two different y coordinates.
- Bresenham algorithm uses only integer calculations such as addition, subtraction and multiplication by 2 which the computer can perform rapidly. Thus, it is an efficient method for scan converting the straight line.
- The basic principle is that, to select an optimum raster location to present a straight line. The algorithm always increments either x or y value by one unit depending on the slope of the line.
- The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called the Decision Variable or error represented as e. If e >= 0, it implies that the pixel above the line is closes to the true value. if e <=0, it implies the pixel below the line is closer to the true value.
- Thus by checking the sign of the error term, it is possible to determine the better pixel to represent the line path.

Algorithm

Step 1 : Initialize variable

The line end points are (x_1, y_1) and (x_2, y_2), assumed these are not equal.

$x = x_1, y = y_1$

$\Delta x = \text{abs}(x_2 - x_1) \qquad s_1 = \text{sign}(x_2 - x_1)$

$\Delta y = \text{abs}(y_2 - y_1) \qquad s_2 = \text{sign}(y_2 - y_1)$

Sign function returns -1 , 0, 1 as its argument is <0, =0, >0 respectively

Step 2 : Interchange

If $\Delta y > \Delta x$ then

Δx and Δy and set Interchange = 1

Else Interchange = 0

Step 3 : Initialise error $\bar{e}$ to compensate for non-zero intercept.

$\bar{e} = 2 * \Delta y - \Delta x$

Step 4 : For i = 1 to Δx

Plot (x, y)

While ($\bar{e}$ >=0)

If(Interchange=1) then,

$x = x + s_1$

else

$y = y + s_2$

End If

$\bar{e} = \bar{e} - 2\Delta x$

End While

If (Interchange=1) then

$y = y + s_2$

else

$x = x + s_1$

End if

$\bar{e} = \bar{e} + 2\Delta y$

i++

Step 5 : END

Example 2.6 : *Interpret Bresenham's algorithm to find which are pixel are turned on for the line segment between (1,2) and (7,6).*

Solution :

Step 1 : Initialise variable

The line end points are (1,2) and (7,6)

$x_1 = 1, \quad y_1 = 2$

$x_2 = 7, \quad y_2 = 6$

$x = y_1 = 1, \; y = y_1 = 2$

$\Delta x = \text{abs}(x_2 - x_1)$

$= \text{abs}(7-1) = 6$

$\Delta y = \text{abs}(y_2 - y_1)$

$= \text{abs}(6-2) = 4$

$s_1 = \text{sign}(x_2 - x_1) = 1$ as (6>0)

$s_2 = \text{sign}(y_2 - y_1) = 1$ as (4>0)

Step 2 : Interchange as , $\Delta y < \Delta x$

Interchange Δx and Δy and set Interchange =0

Tabulating the result for each iteration

I	X	Y	$\bar{e}$	Plot
	1	2	2	(1,2)
1	2	3	8	(2,3)
2	3	4	4	(3,4)
3	4	5	0	(4,5)
4	5	6	-4	(5,6)
5	6	6	4	(6,6)
6	7	7	0	(7,7)

Example 2.7 : *Consider the line from (5,5) to (13,9). Use the Bresenham's algorithm to rasterize the line.*

Solution :

Step 1 : The line end points are (5,5) and (13,9)

Initialise Variable

$x = x_1, \quad y = y_1$

$x = 5, \quad y = 5$

$\Delta x = \text{abs}(x_2 - x_1)$

$= \text{abs}(13 - 5) = 8$

$\Delta y = \text{abs}(y_2 - y_1)$

$= \text{abs}(9-5) = 4$

$s_1 = \text{sign}(x_2 - x_1) = 1$ as (8>0)

$s_2 = \text{sign}(y_2 - y_1) = 1$ as (4>0)

Step 2 : as , $\Delta y < \Delta x$

Interchange Δx and Δy and set Interchange =0

Tabulating the result for each iteration

I	X	Y	$\bar{e}$	Plot
	5	5	0	(5,5)
1	6	6	-8	(6,6)
2	7	6	0	(7,6)
3	8	7	-8	(8,7)
4	9	7	0	(9,7)
5	10	8	-8	(10,8)
6	11	9	-16	(11,9)
7	12	9	-8	(12,9)
8	13	9	0	(13,9)

2.2 CIRCLE DRAWING ALGORITHMS

- Drawing a circle on the screen is a little complex than drawing a line. There are popular algorithms for generating a circle – DDA circle algorithm, bresenham's algorithm and Midpoint Circle Algorithm.
- These algorithms are based on the idea of determining the subsequent points required to draw the circle.
- A circle is a symmetrical Figure. It has eight way symmetry as shown in Fig. 2.6.

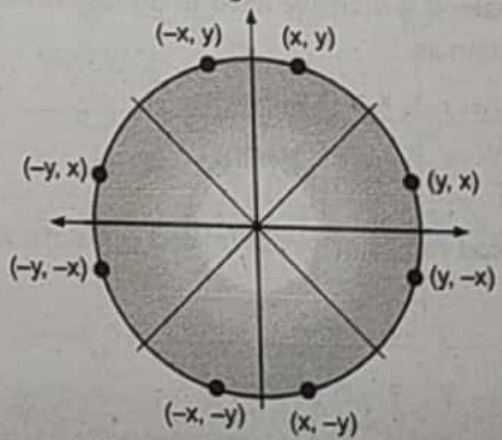


Fig. 2.6 : Symmetry of circle

The equation for a circle is :

$$X^2 + Y^2 = r^2$$

where

r is the radius of the circle

So, we can write a simple circle drawing algorithm by solving the equation for y at unit x intervals using :

$$Y = \pm\sqrt{r^2 - X^2}$$

2.2.1 DDA Circle Drawing Algorithm

The Equation of Circle is

$$X^2 + Y^2 = r^2$$

Hence , the circle equation in the form of differential equation

$$2x\,dx + 2y\,dy = 0$$

$$x\,dx + y\,dy = 0$$

$$\therefore \quad y\,dy = -x\,dx$$

Therefore , The differential equation of a circle with center as origin is

$$\frac{dy}{dx} = \frac{-x}{y}$$

From above equation we can construct a circle by using x value and y value incremental values Δx and Δy as,

$$\Delta x = \varepsilon y \text{ and}$$

$$\Delta y = -\varepsilon x$$

Where,

$$\varepsilon = 2^{-n}$$

$2^{n-1} \le r \le 2^n$ and r is radius of the circle.

Thus, we can get next pixel by applying incremental steps

$$x_{n+1} = x_n + \varepsilon y_n$$

$$y_{n+1} = y_n - \varepsilon x_n$$

But, above equations gives a spiral shape instead of a circle. To make it a circle we need to do one correction in above equation as,

$$x_{n+1} = x_n + \varepsilon y_n$$

$$y_{n+1} = y_n - \varepsilon x_{n+1}$$

Algorithm :

Step 1 : Read radius of circle (r) and calculate value of ε (epsilon).

Step 2 : $x = 0$

$y = r$

Step 3 : $x_1 = x$

$y_1 = y$

Step 4 : do

{

$X_2 = x_1 + \varepsilon y_1$

$Y_2 = y_1 - \varepsilon x_2$

plot (x_2, y_2)

}

While $((y_1 - y) < \varepsilon \,||\, (x - x_1) > \varepsilon)$ //condition is for , whether current point is starting point

Step 5 : Stop.

2.2.2 Bresenham Circle Drawing Algorithm

- Jack E. Bresenham invented this algorithm in 1962. The objective was to optimize the graphic algorithms for basic objects, in a time when computers were not as powerful as they are today.
- This algorithm is called as incremental, because the position of the next pixel is calculated on the basis of the last plotted pixel, instead of just calculating the pixels from a global formula. Such logic is faster for computers to work on and allows plotting circles without trigonometry.
- The algorithm use only integers, and that's where the strength is floating point calculations slow down the processors.

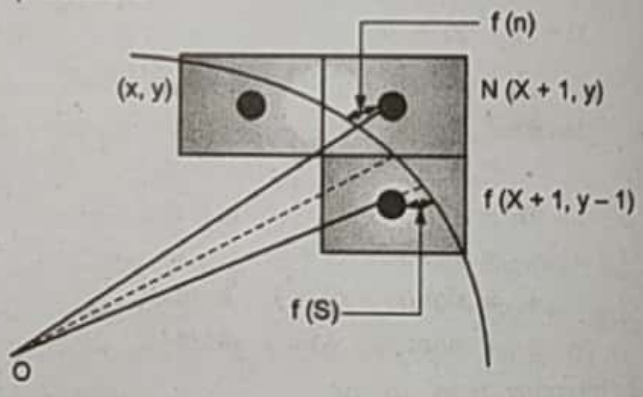


Fig. 2.7 : Bresenham circle drawing

- At any point (x,y), we have two choices – to choose the pixel on east of it, i.e. N(x+1,y) or the south-east pixel S(x+1,y-1). To choose the pixel, we determine the errors involved with both N and S which are f(N) and f(S) respectively and which ever gives the lesser error, we choose that pixel.

Let d = f(N) + f(S), where d can be called as "decision parameter",

if d<=0, then, N(x+1,y) is to be chosen as next pixel and if d>0, then, S(x+1,y-1) is to be chosen as next pixel;

- After performing derivations and putting the values of f(N) and f(S) in equation of circle, we get the value of decision parameter as

  for $d < 0$; $d = d + 4x + 6$

  for $d >= 0$; $d = d + 4(x - y) + 10$

Algorithm

Step 1 : Read radius (r) of circle.

Step 2 : initialize decision variable d=3-2r

Step 3 : x = 0 and y = r

Step 4 : do

```
{
    Plot(x,y)
    x = x + 1
    if (d<0)
      {   d = d+ 4x + 6
      }
  else if (d≥0)
      {
y = y - 1
d= d+ 4 (x - y) + 10
      }
}
while(x<y)
```

Step 5 : Plot pixels in all octants as

```
Plot (x, y)
Plot (y, x)
Plot (-y, x)
Plot (-x, y)
Plot (-x, -y)
Plot (-y, -x)
Plot (y, -x)
Plot (x, -y)
```

Step 6 : Stop.

2.2.3 Midpoint Circle Algorithm

Equation for Decision Parameter of Midpoint Circle Algorithm :

- A circle is defined as a set of points that are all at a given distance r from a center positioned at (X_c, Y_c).
- This is represented mathematically by the equation,

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad \ldots (2.1)$$

- Using equation (2.1) we can calculate the value of y for each given value of x as

$$y = y_c \pm \sqrt{r^2 - (X_c - X)^2} \quad \ldots (2.2)$$

- Thus one could calculate different pairs by giving step increments to x and calculating the corresponding value of y. But this approach involves considerable computation at each step and also the resulting circle has its pixels sparsely plotted for areas with higher values of the slope of the curve.
- Midpoint circle algorithm uses an alternative approach, where in the pixel positions along the circle are determined on the basis of incremental calculations of a decision parameter.

  Let,

$$f(x, y) = (x - x_c)^2 + (y - y_c)^2 - r^2 \quad \ldots (2.3)$$

  Thus, $f(x, y) = 0$ represents the equation of a circle.
- Further, we know from coordinate geometry, that for any point, the following holds :
  1. $f(x, y) = 0$ → The point lies on the circle.
  2. $f(x, y) < 0$ → The point lies within the circle.
  3. $f(x, y) > 0$ → The point lies outside the circle.
- In midpoint circle algorithm, the decision parameter at the k^{th} step is circle function evaluated using the coordinates of the midpoint of the two pixel centres which are the next possible pixel position to be plotted.
- Let us assume that we are giving unit increments to x in the plotting process and determining the y position using this algorithm. Assuming we have just plotted the k^{th} pixel at (X_k, Y_k), we next need to determine whether the pixel at the position (X_{k+1}, Y_k) or the one at (X_{k+1}, Y_{k-1}) is closer to the circle.

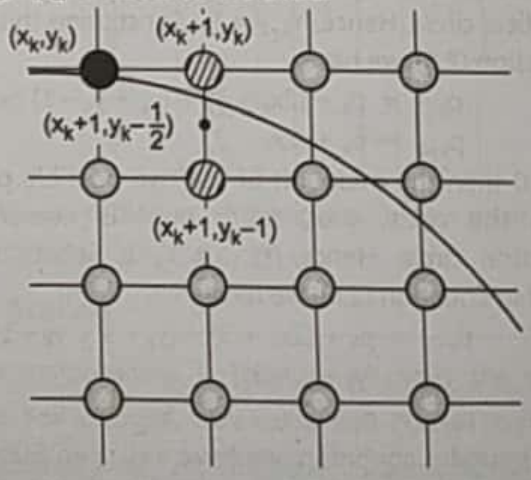


Fig. 2.8 : Decision parameter for mid point circle

Our decision parameter p_k at the k^{th} step is the circle function evaluated at the midpoint of these two pixels.

The coordinates of the midpoint of these two pixels are $(X_k+1, Y_k-1/2)$.

Thus p_k

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 \quad ...(2.4)$$

Successive decision parameters are obtained using incremental calculations, thus avoiding a lot of computation at each step. We obtain a recursive expression for the next decision parameter i.e. at the $k+1^{th}$ step, in the following manner.

Using Equation (2.4), at the $k+1^{th}$ step, we have :

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$p_{k+1} = f\left(x_{k+1} + 1, y_{k+1} - \frac{1}{2}\right)$$

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

Or, $$p_{k+1} = (x_k + 1 + 1)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2$$

Or, $$p_{k+1} = (x_k + 2)^2 + \left(y_{k+1} - \frac{1}{2}\right)^2 - r^2 \quad ... (2.5)$$

Equation (2.5) and (2.4) gives,

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

or, $$p_{k+1} = p_k + (2x_k + 3).1 + (y_{k+1} + y_k - 1)(y_{k+1} - y_k) \quad ... (2.6)$$

Now, if $p_k <= 0$, then the midpoint of the two possible pixels lies within the circle, thus north pixel is nearer to the theoretical circle. Hence, $Y_{k+1} = Y_k$. Substituing this value of in equation (2.6), we have

$$p_{k+1} = p_k + (2x_k + 3) + (y_k + y_k - 1)(y_k - y_k)$$

or, $$p_{k+1} = p_k + (2x_k + 3)$$

If $p_k > 0$ then the midpoint of the two possible pixels lies outside the circle, thus south pixel is nearer to the theorectical circle. Hence, $Y_{k+1} = Y_k - 1$. Substituting this value of in Equation (2.6), we have

$$p_{k+1} = p_k + (2x_k + 3) + (y_k - 1 + y_k - 1)(y_k - y_k - 1)$$

or, $$p_{k+1} = p_k + 2(x_k - y_k) + 5$$

For the boundry condition, we have x=0, y=r. Substituting these values in (2.4), we have

$$p_0 = (0 + 1)^2 + \left(r - \frac{1}{2}\right)^2 - r^2$$

$$= 1 + r^2 + \frac{1}{4} - r - r^2 = \frac{5}{4} - r$$

For integer values of pixel coordinates, we can approximate $P_0 = 1 - r$,

Thus we have,

If $p_k \le 0$;

$$Y_{k+1} = Y_k$$

and $$p_{k+1} = p_k + (2x_k + 3)$$

If $pk > 0$;

$$Y_{k+1} = Y_k - 1$$

and $$p_{k+1} = p_k + 2(x_k - y_k) + 5$$

Also, $$p_o = 1 - r$$

Advantage :

- The midpoint method for deriving efficient scan-conversion algorithms to draw geometric curves on raster displays in described. The method is general and is used to transform the nonparametric equation f(x,y) = 0, which describes the curve, into an algorithms that draws the curve.

Disadvantage :

- Time consumption is high.
- The distance between the pixels is not equal so we won't get smooth circle.

Eight Way Symmetry

- In the mid-point circle algorithm we use eight way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points. Assume that we have just plotted point (x_k, y_k). The next point is a choice between (x_{k+1}, y_k) and (x_{k+1}, y_{k-1}). We would like to choose the point that is nearest to the actual circle.

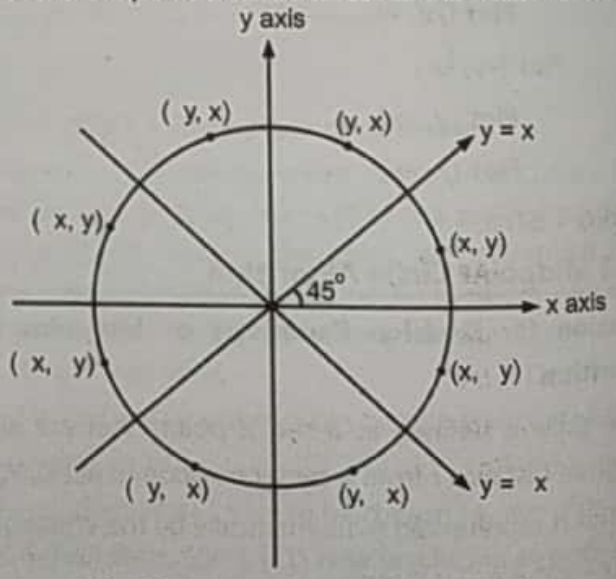


Fig. 2.9 : Eight way symmetry of circle

Algorithm

Step 1 : Set X = 0 and Y = R

Step 2 : Set P= 1-R

Step 3 : While (X < Y)

plot(X, Y)

If (P< 0) Then

X = X + 1

Y = Y

P = P+ 2X + 1

Else

X = x+1

Y = Y -1

P= P+ 2(X -Y)+ 1

End of While

Step 4 : Plot pixel (x, y) and also plot pixels in remaining seven octants as,

Plot (y, x)

Plot (-y, x)

Plot (-x, y)

Plot (-x, -y)

Plot (-y, -x)

Plot (y, -x)

Plot (x, -y)

Step 5 : Stop

2.3 2D GEOMETRIC TRANSFORMATIONS

2.3.1 Translation

- The process of changing the position of an object is called as translation. The object is translated in straight line path from one position to another.
- For example, consider point P(x, y) where x, y are the co-ordinates of original position. The point is translated to new position P'(x', y')

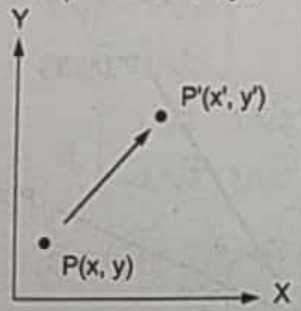


Fig. 2.10 : Translation

- Here the point P has been shifted by t_x units in X-direction similarly in t_y units in Y-direction. where t_x, t_y be the quantities by which the point P has to be shifted in x and y-direction respectively.

$$x' = x + t_x \quad \ldots (2.7)$$

$$y' = y + t_y \quad \ldots (2.8)$$

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

$$\therefore \quad P = P + T \quad \ldots (2.9)$$

The pair t_x, t_y is called translation vector.

Examples 2.8 : *Translate the polygon with co-ordinates A(2, 3), B(5, 9)and C(8, 9) by 6 units in x- direction and 3 units in y-direction.* ***[Oct 2010 4 Marks]***

Solution : From Equation (2.9)

$$P' = P + T$$

$$A' = A + T$$

$$A' = \begin{bmatrix} 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}$$

$$B' = \begin{bmatrix} 5 \\ 9 \end{bmatrix} + \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 11 \\ 12 \end{bmatrix}$$

$$C' = \begin{bmatrix} 8 \\ 9 \end{bmatrix} + \begin{bmatrix} 6 \\ 3 \end{bmatrix} = \begin{bmatrix} 14 \\ 12 \end{bmatrix}$$

Thus the co-ordinates of triangle with new position will be A'(8, 6), B'(11, 12), C'(14, 12).

Example 2.9 : *Translate the polygon A(5, 7), B(7, 11) and C(12, 15) by 4 units in x-direction and 6 units in y-direction.*

Solution : From Equation (2.9)

$$P' = P + T$$

$$A' = A + T$$

$$A' = \begin{bmatrix} 5 \\ 7 \end{bmatrix} + \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 9 \\ 13 \end{bmatrix}$$

$$B' = \begin{bmatrix} 7 \\ 11 \end{bmatrix} + \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 11 \\ 17 \end{bmatrix}$$

$$C' = \begin{bmatrix} 12 \\ 15 \end{bmatrix} + \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 16 \\ 21 \end{bmatrix}$$

Thus the co-ordinates of triangle with new position will be A'(9, 13), B'(11, 17), C'(16, 21).

2.3.2 Scaling

This transformation is used to changes the size of an object. For example, consider point P(x, y). This point is to be scaled to P' (x', y'). Then the x scaling factor is S_x and y scaling factor is S_y for the x and y co-ordinates respectively.

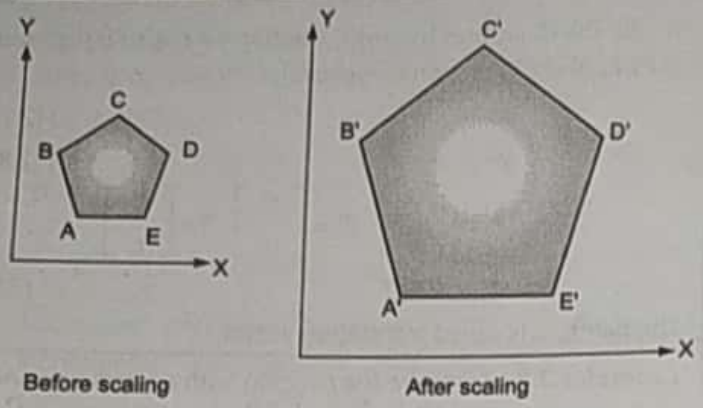


Fig. 2.11 : Scaling

To obtain the scaled object point i.e. the new point the original point i.e. old point co-ordinates is to be multiplied by the scaling factors as :

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

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

$$\therefore \quad P' = P \cdot S$$

$S_x > 1$, Increase in size

$S_x < 1$, Reduction in size

$S_x = 1$, Uniform scaling

$S_x < 1$, Reduction in size

$S_x = 1$, Uniform scaling

Example 2.10 : *Scale the polygon with coordinates A(2,5), B(7,10) and C(10,2) by 3 units in X direction and 4 units in y direction.* ***[Oct. 2010, 6 Marks]***

Solution : The scaling matrix will be,

$$S = \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}$$

The given coordinates of polygon in matrix form :

$$\begin{matrix} A' \\ B' \\ C' \end{matrix} \begin{bmatrix} x_1' & y_1' \\ x_2' & y_2' \\ x_3' & y_3' \end{bmatrix} = \begin{matrix} A \\ B \\ C \end{matrix} \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 6 & 20 \\ 21 & 40 \\ 30 & 8 \end{bmatrix}$$

Therefore the coordinates of polygon after scaling are A'(6,20), B'(21,40), C'(30,8).

Example 2.11 : *Find out the final coordinates of a figure bounded by the coordinates (1,1), (3,4), (5,7), (10,3) when scaled by two units in X direction and three units in y direction.* ***[May 2015, 6 Marks]***

Solution : The scaling matrix will be,

$$S = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

The given coordinates of polygon in matrix form :

$$\begin{matrix} A \\ B \\ C \end{matrix} \begin{bmatrix} 1 & 1 \\ 3 & 4 \\ 5 & 7 \\ 10 & 3 \end{bmatrix}$$

$$\therefore \begin{matrix} A' \\ B' \\ C' \\ D' \end{matrix} \begin{bmatrix} x_1' & y_1' \\ x_2' & y_2' \\ x_3' & y_3' \\ x_4' & y_4' \end{bmatrix} = \begin{matrix} A \\ B \\ C \end{matrix} \begin{bmatrix} 1 & 1 \\ 3 & 4 \\ 5 & 7 \\ 10 & 3 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 4 \\ 9 & 16 \\ 15 & 28 \\ 30 & 12 \end{bmatrix}$$

Therefore the coordinates of polygon after scaling are A'(3, 4), B'(9, 16), C'(15, 28),D'(30, 12).

2.3.3 Rotation

The transformation in which an object is rotated along the circular path is called rotation of an object about origin. In this case the object can be rotated by a given angle in either clockwise or anticlockwise direction.

Consider point P(x, y). Let the angle of rotation be θ°.

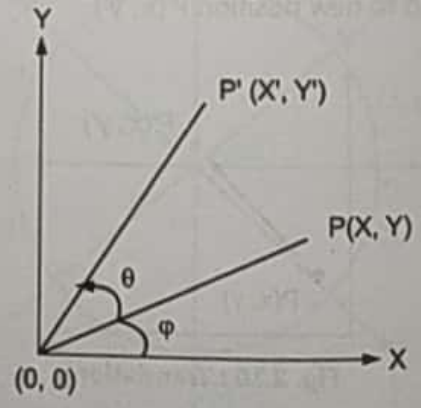


Fig. 2.12 : Rotation

The original coordinates of points is given as

$$\therefore \quad x = r\cos\phi$$
$$y = r\sin\phi \qquad \ldots (2.10)$$

we can express the transformed coordinates in terms of angles θ and φ as

$$x' = r\cos(\phi + \theta)$$
$$= r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi + \theta)$$
$$= r\cos\phi\sin\theta + r\sin\phi\cos\theta \qquad \ldots (2.11)$$

Substituting equation (2.10) into equation (2.11), we get

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

So we can write the rotation equation in matrix form as

$$P'' = R \cdot P$$

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

The positive value of rotation angle define anticlockwise or counterclockwise rotation where negative value of rotation angle define clockwise rotation of an object.

Therefore for clockwise rotation, we get the matrix as

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

Example 2.12 : *A point (5,4) is rotated anticlockwise by an angle of 45. Find rotation matrix and the resultant point.*

[Oct. 2010, 6 Marks]

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

$\theta = 45°$

$$R = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$P' = [5 \;\; 4] \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$= \left[\frac{5}{\sqrt{2}} - 2/\sqrt{2} \quad \frac{5}{\sqrt{2}} + 4/\sqrt{2} \right]$$

$$= [\, 1/\sqrt{2} \quad 9/\sqrt{2} \,]$$

Example 2.13 : *Consider a polygon with vertices A(10' 10)' B(15' 15) and C(20, 10). Obtain the following rotations of the polygon about the origin :*

(i) Counterclockwise π

(ii) Clockwise π/2

(iii) Counterclockwise 5π/4

(iv) Clockwise 3π/4.

Solution : Counterclockwise π

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

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

$$= \begin{bmatrix} \cos\pi & \sin\pi \\ -\sin\pi & \cos\pi \end{bmatrix}$$

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

$$P' = P \cdot R$$

$$P' = \begin{bmatrix} 10 & 10 \\ 15 & 10 \\ 20 & 10 \end{bmatrix} = \begin{bmatrix} -10 & -10 \\ -15 & -10 \\ -20 & -10 \end{bmatrix}$$

(ii) Clockwise π/2

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

$$R = \begin{bmatrix} \cos\pi/2 & \sin\pi/2 \\ -\sin\pi/2 & \cos\pi/2 \end{bmatrix}$$

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

$$P' = P \cdot R$$

$$P' = \begin{bmatrix} 10 & 10 \\ 15 & 10 \\ 20 & 10 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -10 & 10 \\ -10 & 15 \\ -10 & 20 \end{bmatrix}$$

(iii) Counterclockwise 5π/4

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

$$R = \begin{bmatrix} \cos 5\pi/4 & \sin 5\pi/4 \\ -\sin 5\pi/4 & \cos 5\pi/4 \end{bmatrix} = \begin{bmatrix} -0.7 & -0.7 \\ 0.7 & -0.7 \end{bmatrix}$$

$$P' = P \cdot R$$

$$P' = \begin{bmatrix} 10 & 10 \\ 15 & 10 \\ 20 & 10 \end{bmatrix} \begin{bmatrix} -0.7 & -0.7 \\ 0.7 & -0.7 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -14 \\ -3.5 & -17.5 \\ -7 & -21 \end{bmatrix}$$

(iv) Clockwise $3\pi/4$

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

$$R = \begin{bmatrix} \cos 3\pi/4 & \sin 3\pi/4 \\ -\sin 3\pi/4 & \cos 3\pi/4 \end{bmatrix} = \begin{bmatrix} -0.7 & 0.7 \\ -0.7 & -0.7 \end{bmatrix}$$

$$P' = P \cdot R$$

$$P' = \begin{bmatrix} 10 & 10 \\ 15 & 10 \\ 20 & 10 \end{bmatrix} \begin{bmatrix} -0.7 & -0.7 \\ -0.7 & -0.7 \end{bmatrix}$$

$$= \begin{bmatrix} -14 & 0 \\ -17.5 & 3.5 \\ -21 & 7 \end{bmatrix}$$

2.4 REFLECTION

This type of transformation generates the mirror image of an object. To carry out the reflection it is necessary to define the axis of reflection.

1. **Reflection About x-axis :** Consider point P(x, y)

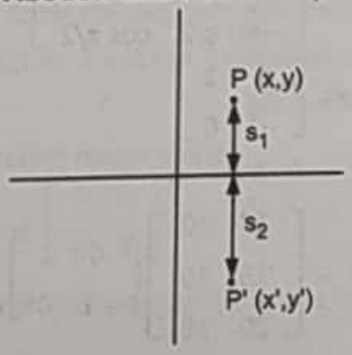


Fig. 2.13

Here P' is the mirror image of P.

$\therefore$ $S_1 = S_2$

$x' = x$

$y' = -y$

In matrix representation,

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

2. **Reflection About y-axis :** Consider point P(x, y)

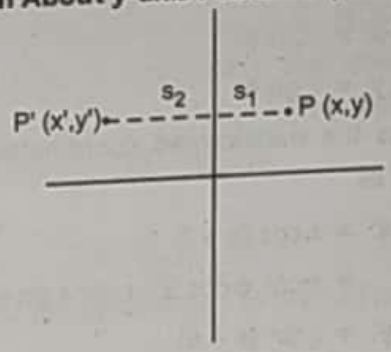


Fig. 2.14

Here P' is the mirror image of P.

$\therefore$ $S_1 = S_2$

$x' = -x$

$y' = y$

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

Similarly, the transformation matrix for y = x axis will be,

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

and for y = –x axis the transformation matrix will be,

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

2.5 POLYGON

- So far we have discussed about the line and circle. The earlier display devices such as plotters, DVSTs, vector refresh displays were line – drawing devices. However the raster displays can display solid objects and patterns too. Thus, we must know the fundamental aspects of graphics primitives. The basic surface primitive is polygon. The polygon is a figure with many sides. A polygon can be represented as a group of connected edges, which forms a closed figure. The line segments which form the boundary of polygon are called as edges or sides. The end point of the edges or sides of the polygon are called as vertices.

Example : Triangle – It is a polygon having three edges and three vertices.

Fig. 2.15

However, polygon must be any shape as shown below in Fig. 2.16.

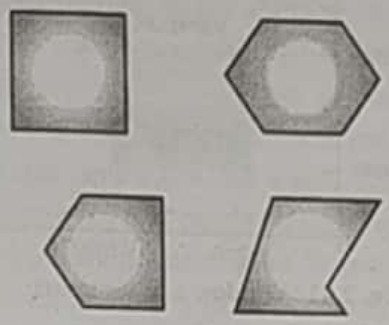


Fig. 2.16

- A polygon is a chain of connected line segments. It is specified by giving the vertices (nodes), P_0, P_1, P_2, ... and so on. The first vertex is called the initial or starting point and the last vertex is called the final or terminal point as shown below. When starting point of any polyline is same i.e. when polyline is closed, then it is called polygon.

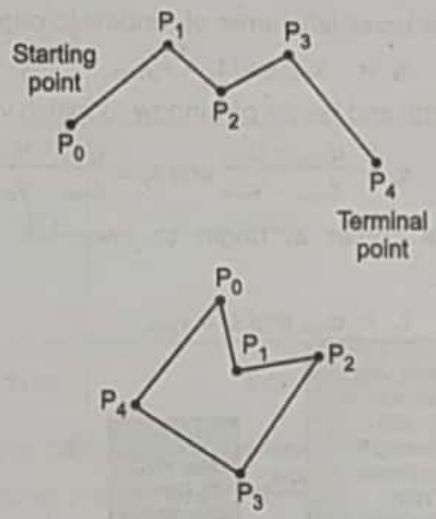


Fig. 2.17

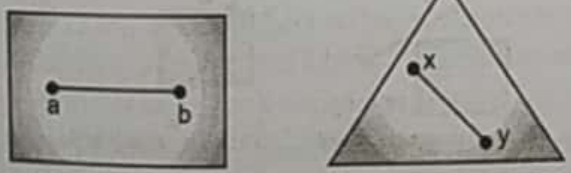
2.5.1 Types of Polygon

There are two types of polygon :

1. Convex Polygon
2. Concave Polygon

- A polygon is said to be convex, if the line joining any two interior points of the polygon lies completely inside the polygon.

Example : Triangle, Square, Rectangle etc.



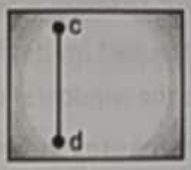


Fig. 2.18

- A polygon is said to be concave if the line joining any two interior points of the polygon does not lie completely within the polygon.

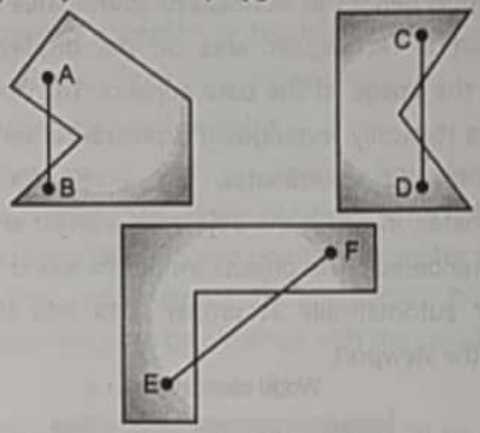
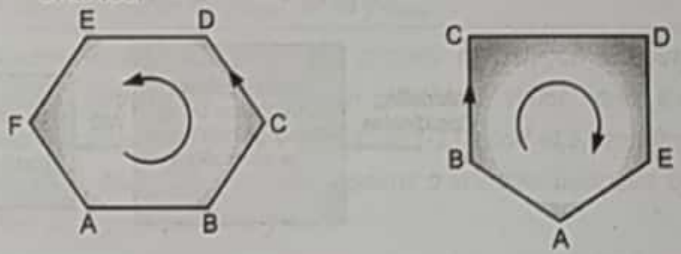


Fig. 2.19

- A normal convention of writing the polygon as P vertices $P_1, P_2, \ldots, P_N$.
- A polygon is said to be positively oriented if visiting of all the vertices in the given order produces a counter clockwise circuit, otherwise it is said to be negatively oriented.



Positive Orientation **Negative Orientation**

Fig. 2.20

2.6 WINDOWING AND CLIPPING

2.6.1 Concept of Window and Viewport

- The space in which objects are described is called world coordinate.
- World coordinates used Cartesian XY coordinate system used in mathematics.

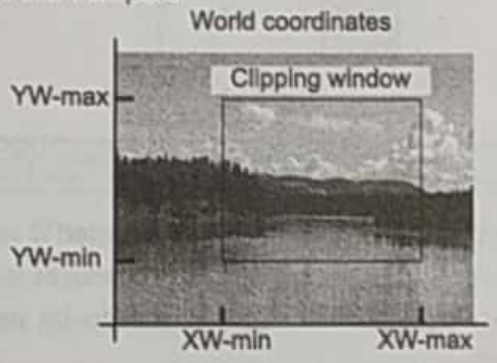
Window

- Defining a window in these world coordinate called "world window" or simply "window" in computer graphics

- The world "window" specifies which part of the world should be drawn, the part inside the window should be drawn and outside the window should be clipped.
- "Window" refers to the area in "world space" or "world coordinates" that you want to project onto the screen.
- The window is often centered around the origin.

Viewport

- A viewport defines in normalized coordinates (u, v). A viewport is rectangular area on the display device where the image of the data appears. The viewport is an area (typically rectangular) expressed in rendering-device-specific coordinates, e.g. pixels for screen coordinates, in which the objects of interest are going to be rendered. The objects inside the world window appear automatically at proper sizes and locations inside the viewport.



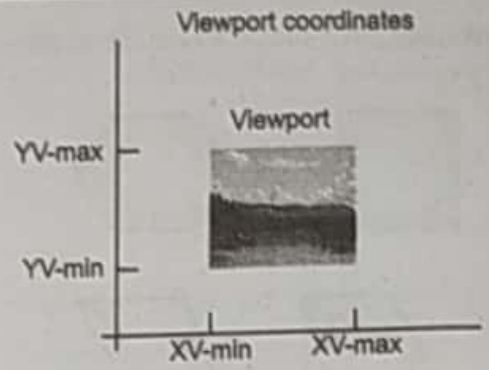


Fig. 2.21 : Window and viewport

2.6.2 Viewing Transformations

- Moving objects from their 3D locations to their positions in a 2D view called viewing transformation.
- Need to transform points from "world" view (window) to the screen view (viewport).
- Maintain relative placement of points (usually).
- Can be done with a translate-scale-translate sequence.
- The window-to-viewport transform is

1. **Translate** lower-left corner of window to origin.

$$t_x = -x_{min} \text{ and } t_y = -y_{min}$$

2. **Scale** width and height of window to match viewport's.

$$s_x = \frac{u_{max} - u_{min}}{x_{max} - x_{min}} \text{ and } s_y = \frac{v_{max} - v_{min}}{y_{max} - y_{min}}$$

3. **Translate** corner at origin to lower-left corner in viewport.

$$t_x = u_{min} \text{ and } t_y = v_{min}$$

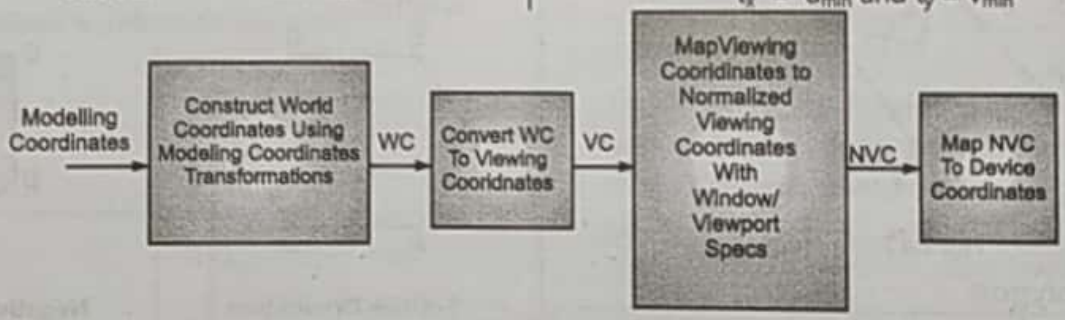


Fig. 2.22 : Translating world coordinate to device coordinate

2.7 LINE CLIPPING

- When a window is "placed" on the world, only certain objects and parts of objects can be seen. Points and lines which are outside the window are "cut off" from view. This process of "cutting off" parts of the image of the world is called Clipping.
- In clipping, we examine each line to determine whether or not it is completely inside the window, completely outside the window, or crosses a window boundary.
- **If inside** the window, **the line is displayed**. If **outside** the window, the lines and points are **not displayed**. If **a line crosses** the boundary, we must determine the point of intersection and **display only the part which lies inside the window**. Refer Fig. 2.23.

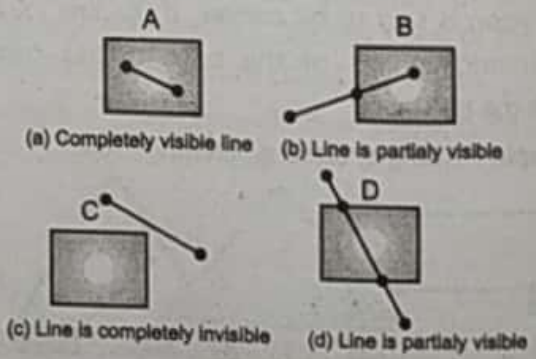


Fig. 2.23

2.7.1 Cohen Sutherland Method

- The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.
- According to cohen Sutherland method, the window is divided in total 9 regions.
- Cohen Sutherland uses 4-bits of code. These 4 bits represent the Top, Bottom, Right, and Left of the region as shown in the following Fig. 2.24.

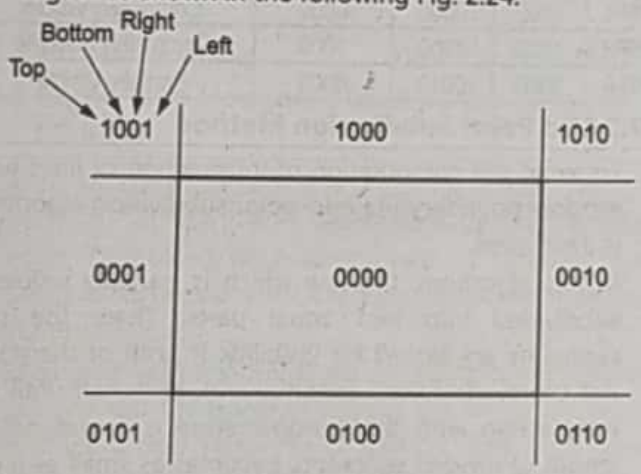


Fig. 2.24 : Cohen sutherland region codes

There are three possibilities for the line :

1. Line can be completely inside the window (This line will be completely removed from the region).
2. Line can be completely outside the window (This line will be completely removed from the region).
3. Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

Step 1 : Assign a region code for each end points.

Step 2 : If both end points have a region code 0000 then accept this line.(i.e. Line is completely visible)

Step 3 : If any one end point or both end points are not 0000 then, perform the logical AND operation for both region codes or for both end points.

Step 3.1 : If the result is not 0000, then reject the line completely.(i.e. the line is completely outside)

Step 3.2 : Else clip the line.

Step 3.2.1 : select the endpoints of the line that is outside the window.

Step 3.2.2 : Find the intersection point at the window boundary (base on region code).

Step 3.2.3 : Replace endpoint with the intersection point and update the region code.

Step 3.2.4 : Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 : Repeat step 1 for other lines.

Finding out Intersection Point

Intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation. For a line with end points coordinates (x_1, y_1) and (x_2, y_2), they coordinate of the intersection point with a vertical boundary can be obtained with the calculation,

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

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

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

Similarly, if we are looking for the intersection with a horizontal boundary, the x coordinate can be calculated as

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

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

***Example 2.14** : Explain the process of polygon clipping using Sutherland Hodgeman Method. What are the intersecting point for line P_1 joining (-1,0) and (4,5) and line P_2 (3,1) and (6,2) if clipped against a window bounded by line x=0, y=0 and x=5, y=3.*

Solution : For Sutherland Hodgman Method refer 2.8.1.

The window with given line segment will look like this,

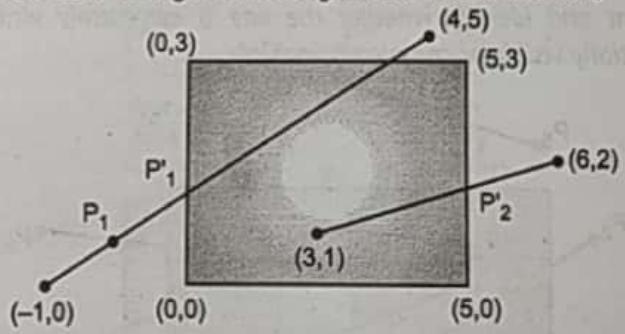


Fig. 2.25

To find P'_1 and P'_2, first we have to find slope m

$$m_1 = \frac{y_2 - y_1}{x_2 - x_1} = \frac{5 - 0}{4 - (-1)} = \frac{5}{5} = 1$$

$$m_2 = \frac{y_2 - y_1}{x_2 - x_1} = \frac{2-1}{6-3} = \frac{1}{3} = 0.33$$

For vertical line intersection, x intersection will be XW_{min} hence $x_{int} = 0$

For y intersection

$$y_{int} = y_1 + m(x - x_1)$$

Where

$$x \text{ value} = 0$$

$$= 0 + 1(0 - (-1)) = 0 + 1 = 1$$

Hence for line P_1 the vertical intersection P'_1 is (0, 1).

Similarly,

For horizontal line intersection y intersection will be y_{max}

$$y_{int} = 3$$

For x intersection

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

where,

$$y = Y_{wmax} = (-1) + \frac{(3-0)}{1} = 2$$

Hence for line P'_1 the horizontal intersection will be (2, 3).

For line P_2 there is no horizontal intersection.

For vertical intersection

$$x_{int} = X_{wmax}$$

$$x_{int} = 5$$

For Y intersection

$$y_{int} = y_1 + m(x - x_1) = 1 + (0.3)(5 - 1)$$

$$= 1 + 1.2 = 2.2$$

Hence for line P_2 the vertical intersection point is (5, 2.2).

Example 2.15 : *Consider the clipping window and the lines shown in below Fig. 2.26 find the region codes for each end point and identify whether the line is completely visible, partially visible or completely invisible.*

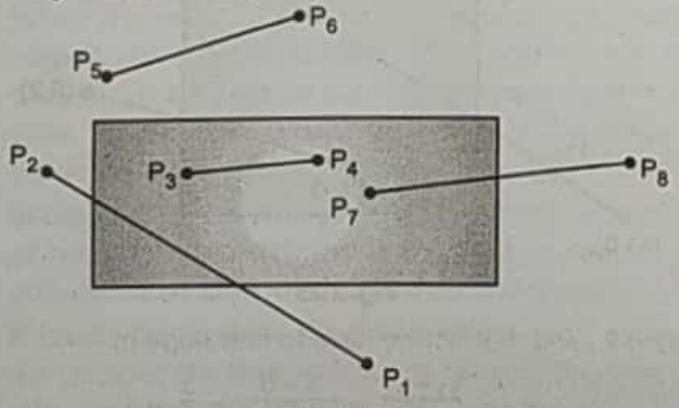


Fig. 2.26

Solution :

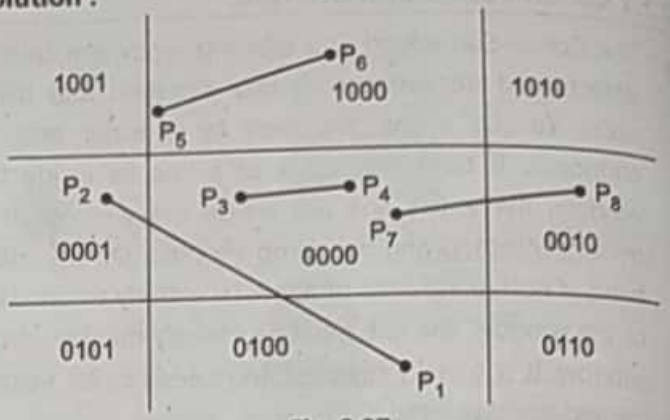


Fig. 2.27

The above Fig. 2.27 shows clipping window with region codes, the end points are ANDed to identify visibility.

Line	End Points		AND Result	Visibility
P1P2	0100	0001	0000	Partially Visible
P3P4	0000	0000	0000	Completely visible
P5P6	1000	1000	1000	Completely Invisible
P7P8	0000	0010	0000	Partially Visible

2.7.2 Mid Point Subdivision Method

- To avoid the computation of intersection of lines with window boundary the mid-point subdivision algorithm is developed.
- In this algorithm, the line which is partially visible is subdivided into two equal parts. Then, the two segments are tested for visibility. If both of them can not be rejected then continue with each half until the intersection with the window edge is found or the length of divided segments becomes as small as it can be treated as a point. Then the visibility of the point is then checked to determine whether it is inside or outside the window.

Algorithm

Step 1 : Assign a region code for each end points.

Step 2 : If both end points have a region code 0000 then accept this line.(i.e. Line is completely visible)

Step 3 : If any one end point or both end points are not 0000 then, perform the logical AND operation for both region codes or for both end points.

Step 3.1 : If the result is not 0000, then reject the line completely.(i.e. the line is completely outside)

Step 3.2 : If step 2 and 3.1 both are not satisfied, line is partially visible.

Step 4 : Divide partially visible line segments to equal parts and repeat steps 3 through 5 for both sub divided line segments until you get completely visible and completely invisible line segments.

Step 5 : Stop.

2.8 POLYGON CLIPPING

- To clip polygons, we need to modify the line-clipping algorithm discussed in the previous section. A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segments refer Fig. 2.28 and 2.29 depending on the orientation of the polygon to the clipping window.
- What we really want to display is a bounded area after clipping,
- Thus only line clipping algorithm will not work for polygon clipping as it generates disconnected edges of polygon as shown in Fig. 2.29.

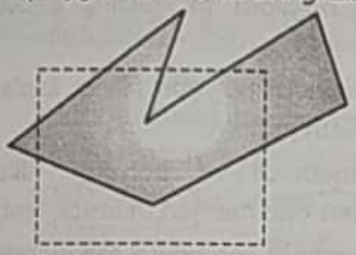


Fig. 2.28 : Before clipping

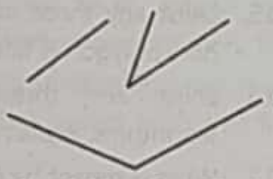


Fig. 2.29 : After clipping

2.8.1 Sutherland Hodgman Method for Clipping Method

- We can correctly clip a polygon by processing the polygon boundary as a whole against each window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn.
- Beginning with the initial set of polygon vertices, we can first clip the polygon against the left boundary to produce a new sequence of vertices. The new set of vertices can then passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. 2.31.
- At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.
- The Sutherland- Hodgman's algorithm clips the original polygon against the simple window edge for obtaining intermediate polygon, by generating list of vertices. It then re-enters the procedure with intermediate polygon, which was generated in previous step and next window edge in the way, when it re-entered with last window edge it generates the list of vertices, which gives a clipped polygon.

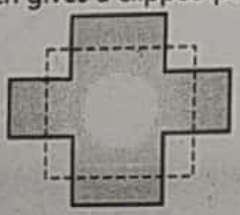


Fig. 2.30 : Original polygon

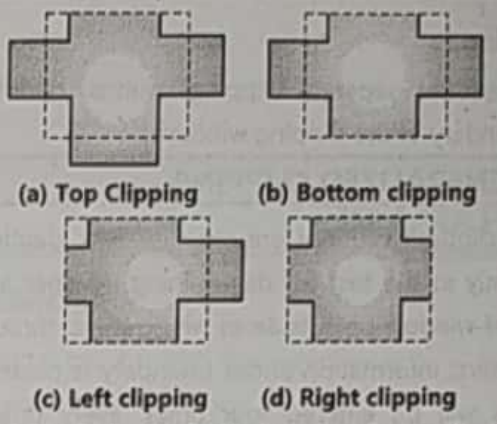


Fig. 2.31 : Sutherland – Hodgman polygon clipping

There are four possible cases when processing vertices in sequence around the perimeter of a polygon. As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests :

- If the first vertex is outside the window boundary and the second vertex is inside, both the intersection point of the polygon edge with the window boundary and the second vertex is added to the output vertex list.
- If both input vertices are inside the window boundary, only the second vertex is added to the output vertex list.
- If the first vertex is inside the window boundary and the second vertex is outside, only the edge intersection with the window boundary is added to the output vertex list.
- If both input vertices are outside the window boundary, nothing is added to the output list. These four cases are illustrated in Fig. 2.32 for successive pairs of polygon vertices. Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

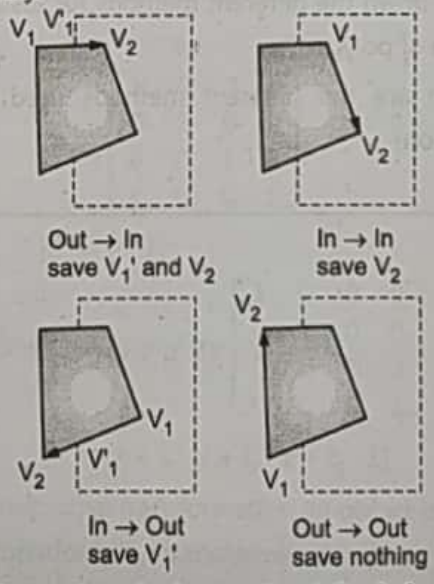


Fig. 2.32 : Processing of edges against left window boundary

Limitation :

- It requires separate clipping routine, one for each boundary of the clipping window.

2.9 GENERALIZED CLIPPING

All the clipping routines are more or less identical. They differ only in the test for determining whether a point is inside of window or outside of window and through their parameters, information about boundary is passed. These routines will be entered four times. Every time with a different boundary specified by its parameters. The routine is generalized so that it can clip along any line including horizontal and vertical boundaries. This can clip along rectangular windows parallel to the axis along the arbitrary lines i.e. window sides may be at any angles. The algorithm is a recursive language that can be used to clip along an arbitrary convex plane.

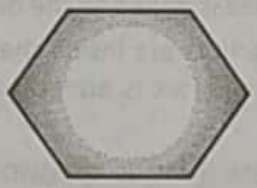


Fig. 2.33 : Windows with eight clipping points

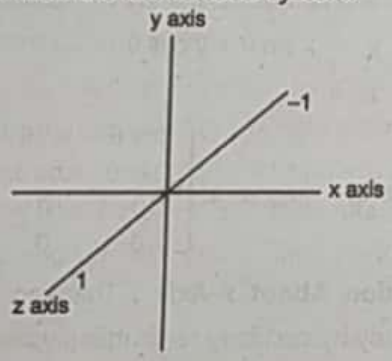
EXERCISE

1. What is polygon and Polyline?
2. What are the different types of polygon?
3. Which are the various approaches used to represent polygon?
4. Represent the polygon using display file structure?
5. Write down the different methods for testing a pixel inside of polygon?
6. Which are the different methods used for filling polygon?
7. Explain boundary fill method for polygon?
8. What is mean by Seed pixel and what is mean by four connected region and eight connected region?
9. Compare Seed fill, edge fill method and flood fill ?
10. What is the need of scan line algorithm?
11. Enlist any three polygon filling algorithms. Explain even-odd method of inside test.
12. Explain even-odd method of inside test.
13. What is inside test? Explain even odd method in detail.
14. Explain the different methods for testing a pixel inside a polygon.
15. Enlist any three methods of polygon filling. Explain how polygon is filled with pattern.
16. Enlist any three methods of polygon filling algorithms. Explain even-odd method of inside test.
17. What is meant by Coherence and how it can increase the efficiency of scan line polygon filling.
18. Write algorithm to fill the polygon area using flood fill method.
19. Write flood fill algorithm.
20. Explain scanline algorithm with example.
21. Describe 3D viewing transformations.
22. Explain concept of viewing parameters with an example.
23. Explain Cohen-Sutherland outcode algorithm with example.
24. Write and explain Cohen-Sutherland line clipping algorithm.
25. Write Cohen-Sutherland line clipping algorithm.
26. Write and explain with an example Cohen-Sutherland line clipping algorithm.
27. Explain Sutherland-Hodgman clipping algorithm with example.

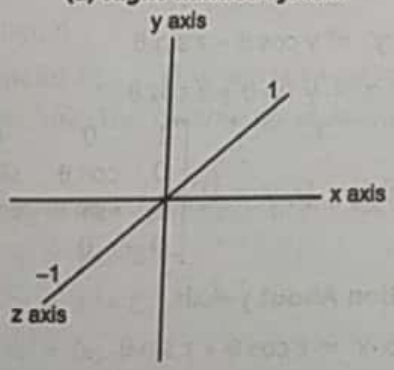
3.1 3-D TRANSFORMATION

Introduction

- Some graphics applications are two-dimensional such as graphs, charts, certain maps etc. But we live in three-dimensional world and deal with many design applications which describe three-dimensional objects. In three-dimensional space we shall extend the transformations to allow translation and rotation, but the viewing surface is only two-dimensional, we must consider ways of projecting the object onto this flat surface to form the image.
- The 3D co-ordinate system is divided into two types :
  1. Right-handed system.
  2. Left-handed system.
- If the thumb of the right hand points in the positive z direction as one curls the fingers of the right hand from x into y, then the coordinates are called a right-handed system. If the thumb points in the negative z direction then it is left handed-system.



(a) Right-handed system



(b) Left-handed system

Fig. 3.1

3.1.1 Translation

- Consider point p with the coordinates (x, y, z). To shift this point to new position p'(x', y', z').

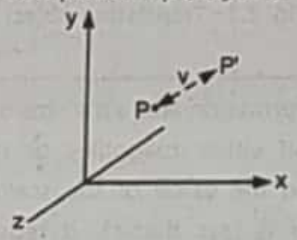


Fig. 3.2 : Translating point

- As shown in above Fig. 3.2 the shifting and direction of the translation is now defined by vector $v = ai + bj + ck$. Thus,

$$x' = x + a$$
$$y' = y + b$$
$$z' = z + c$$

where, a, b, c are the translation factors in x, y and z directions respectively.

The matrix representation will be,

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

or

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

$$P' = P \cdot T$$

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

$$= [x + t_x\ \ y + t_y\ \ z + t_z\ \ 1]$$

Like two dimensional transformation an object is translated in three-dimensions by transforming each vertex of the object.

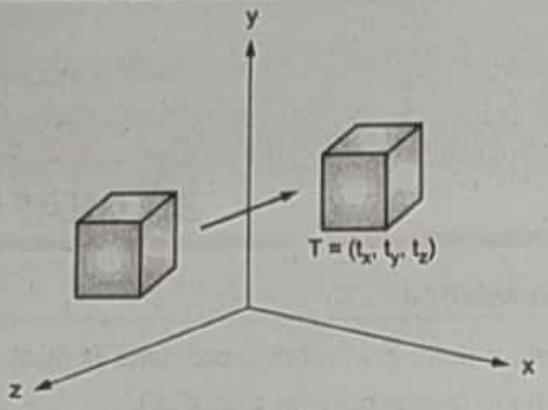


Fig. 3.3 : Translating object

3.1.2 Scaling

- Scaling transformation alters the size of the object. This transformation either magnifies or reduces the size depending on the value of the scaling factor. If the scaling factor is less than 1, it reduces and if it is greater than 1 it magnifies.
- Consider point P(x, y, z) which is to be scaled by S_x, S_y, S_z. Then the new coordinates will be,

$$x' = x \cdot S_x$$
$$y' = y \cdot S_y$$
$$z' = z \cdot S_z$$

The scaling matrix will be,

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The scaling transformation is done with respect to origin i.e. the origin is kept fixed.

3.1.3 Rotation

- In 2D transformation the rotation was prescribed by the angle of rotation and the point of rotation. But in case of 3D rotation, the angle of rotation as well as the axis of rotation need to be mentioned. There are three axis, so the rotation can take place about any of these axis, i.e. about x-axis, y-axis and z-axis respectively.
- Three-dimensional transformation matrix for each coordinate axes rotations with homogeneous coordinates are –

➢ **Rotation About z-Axis :** Let P be the point object in xy plane P(x, y, 0). Rotate it by an angle θ° in counterclockwise direction. The resultant point will be P'(x', y', 0).

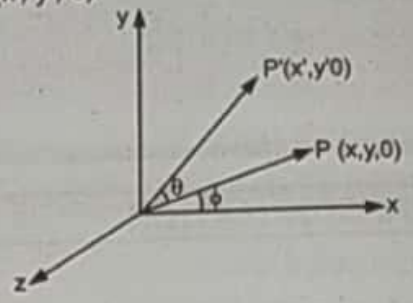


Fig. 3.4

As shown in Fig. 3.4,

$$x = r\cos\phi \quad \ldots (3.1)$$
$$y = r\sin\phi \quad \ldots (3.2)$$
$$x' = r\cos(\theta + \phi)$$
$$y' = r\sin(\theta + \phi)$$
$$x' = r\cos\theta\cos\phi - r\sin\theta\sin\phi$$
$$y' = r\sin\theta\cos\phi + r\cos\theta\sin\phi$$

Put the values of r cos ϕ and r sin ϕ from equations (3.1) and (3.2).

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

The resulting transformation will be,

$$Rz \Rightarrow x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = 0$$

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

➢ **Rotation About x-Axis :** This can be obtained similarly by circularly re-shuffling y and z.

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

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

➢ **Rotation About y-Axis :**

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

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

All the above rotation matrix are for rotation in counterclockwise direction. To obtain the rotation matrix in clockwise direction, change the sign of 't sin θ'.

$$\therefore \quad Rz = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ry = \begin{bmatrix} \cos\theta & 0 & +\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2 ROTATION ABOUT AN ARBITRARY AXIS

- Any line in a space can be used as axis of rotation. For deriving the transformation matrix for rotation by an angle θ° about any arbitrary line in a space, the following transformation must be carried out in a sequence.
  - ➢ **Translation :** Perform translation, so that the line will coincide with origin.
  - ➢ **Rotation :** Perform rotation to align with one of the co-ordinate axis. For example if the line is to be aligned with z-axis then first rotate it about x-axis to bring it in x-z plane and then rotate it about y-axis to align it with z-axis. Then perform rotation about z-axis.
  - ➢ **Retranslation :** Then apply inverse translation to bring the line and coordinates to their original orientation.
- Consider point P(x, y, z) which is to be rotated about an arbitrary line. The parametric equation for the line are –

$$x = x_1 + A_t$$
$$y = y_1 + B_t$$
$$z = z_1 + C_t$$

where, $A = (x_2 - x_1)$
$B = (y_2 - y_1)$
$C = (z_2 - z_1)$

$x_1, y_1, z_1 \rightarrow$ Points on the line
A, B, C → Direction vectors

The first step is to translate the line to bring it in the origin. The translation matrix will be,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix}$$

The inverse transformation will be,

$$T_{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{bmatrix}$$

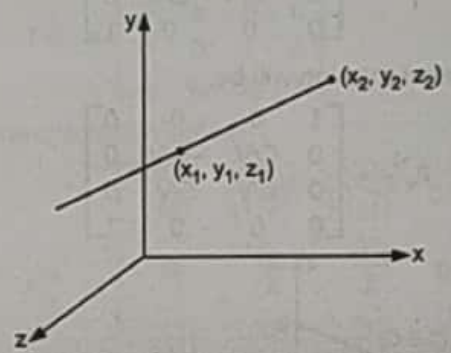


Fig. 3.5 : Before translation the position of line

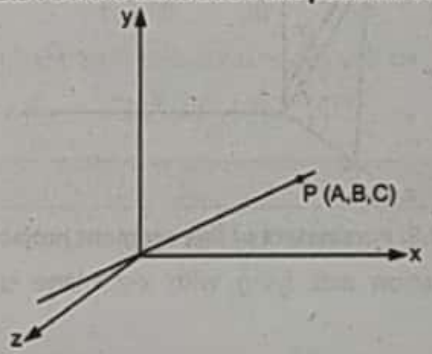


Fig. 3.6 : After translation the position of line

- The second step is the rotation of line about x-axis to bring the line in x-z plane, for this the angle of rotation by which the line is to be rotated must be computed. For this, project a point P(A, B, C) in y-z plane.
  Let P' be the point p in y-z plane.

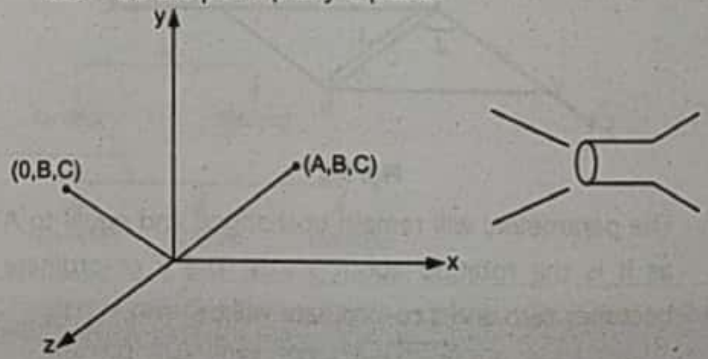


Fig. 3.7 : Projection of a line segment on yz plane

- The coordinates of P' are (0, B, C). The length of segment OP' = $\sqrt{B_2 + C_2}$. The angle of rotation about x-axis is,

$$\cos I = \frac{C}{\sqrt{B_2 + C_2}} \qquad \sin I = \frac{B}{\sqrt{B_2 + C_2}}$$

Put, $\sqrt{B_2 + C_2} = V$

$\therefore \quad \cos I = C/V \qquad \sin I = B/V$

- Now rotation matrix about x-axis, so that arbitrary axis will be in xz plane, the line segment's shadow will lie in z-axis.

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & B/V & 0 \\ 0 & -B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The inverse rotation will be,

$$R_x^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C/V & -B/V & 0 \\ 0 & B/V & C/V & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

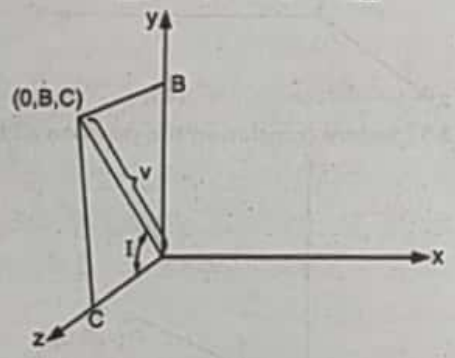


Fig. 3.8: Parameters of line segment projection

- The rotation axis lying with x-z plane is shown in Fig. 3.8

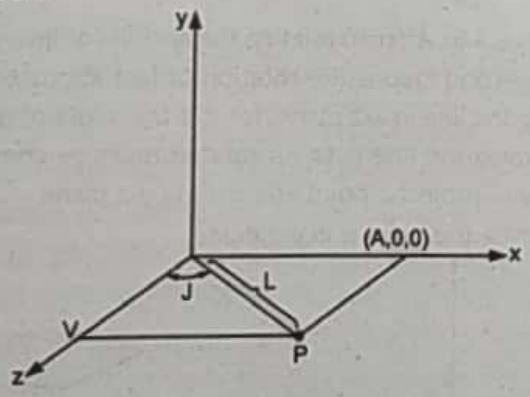


Fig. 3.9

- The parameters will remain unchanged and equal to A as it is the rotation about x-axis. The y co-ordinate becomes zero and z co-ordinate will be,

$$z = \sqrt{R_2 + C_2} = V$$

- The co-ordinates of point P are P(A, 0, 0), the length op will be $\sqrt{A_2 + B_2 + C_2}$.

Put, $\sqrt{A_2 + B_2 + C_2} = L$

$\therefore \quad (op) = L$

- Now perform the rotation of line about y-axis by an angle J to make it align with z-axis.
- As shown in above Fig. 3.9 is an angle between segment op and z-axis.

$$\cos J = V/L \qquad \sin J = A/L$$

- The rotation matrix about y-axis will be,

$$R_y = \begin{bmatrix} V/L & 0 & A/L & 0 \\ 0 & 1 & 0 & 0 \\ -A/L & 0 & V/L & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The inverse transformation will be,

$$R_y^{-1} = \begin{bmatrix} V/L & 0 & A/L & 0 \\ 0 & 1 & 0 & 0 \\ -A/L & 0 & V/L & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Now after performing rotation about y-axis the line will get aligned with z-axis. Then perform the rotation about z-axis by an angle θ. The matrix will be,

$$R_z = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Then apply inverse transformation in sequence R_y^{-1}, R_x^{-1}, T_{-1} to rotate the axis to their original position. The resultant transformation matrix will be,

$$RS = [T]\,[R_x]\,[R_y]\,[R_z]\,[R_y]^{-1}\,[R_x]^{-1}\,[T^{-1}]$$

3.3 REFLECTION

- The reflection in 3D transformation is similar to the concept of 2D transformation. In this case the reference plane i.e. plane about which the reflection is to be taken must be known. Thus, there are three standard reflections about xy plane, xz plane and yz plane. Each plane reference implies that those co-ordinates will remain same which are constituting that plane.

3.3.1 Reflection with Respect to Any Plane

- It is necessary to reflect an object through a plane other than x = 0 (yz-plane), y = 0 (xz-plane) or z = 0 (xy-plane). Procedure to achieve such a reflection i.e. reflection about any plane can be given as follows :

- Translate a known point P0, that lies in the reflection plane to the origin of the co-ordinate system.
- Rotate the normal vector to the reflection plane at the origin until it is coincident with the z axis, this makes the reflection plane z = 0 coordinate plane i.e. xy-plane.
- Reflect the object through z = 0 (xy-plane) co-ordinate plane.
- Perform the inverse transformation to those given above to achieve the result.

- Let $P_0(x_0, y_0, z_0)$ be the given known point. Translate this point to the origin by using corresponding translation matrix.

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

The normal vector will be,

$$N = h_1 I + h_2 J + h_3 K$$

$$|N| = \sqrt{n_1^2 + n_2^2 + n_3^2}$$

$$\lambda = \sqrt{n_2^2 + n_3^2}$$

- To match this vector with z-axis, so that the plane of reflection will be parallel to xy plane, the same procedure will be used as used in rotation.

$$R_{xy} = \begin{bmatrix} \frac{\lambda}{|N|} & 0 & \frac{n_1}{|N|} & 0 \\ \frac{-n_1 n_2}{\lambda |N|} & \frac{n_3}{\lambda} & \frac{n_2}{|N|} & 0 \\ \frac{-n_1 n_3}{\lambda |N|} & \frac{-n_2}{\lambda} & \frac{n_3}{|N|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For reflection about xy plane,

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

The inverse translation will be,

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

The inverse rotation will be,

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\lambda}{|N|} & \frac{-n_2 n_2}{\lambda |N|} & \frac{-n_1 n_3}{\lambda |N|} & 0 \\ 0 & \frac{n_3}{\lambda} & \frac{-n_3}{\lambda} & 0 \\ \frac{n_1}{|N|} & \frac{n_2}{|N|} & \frac{n_3}{|N|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

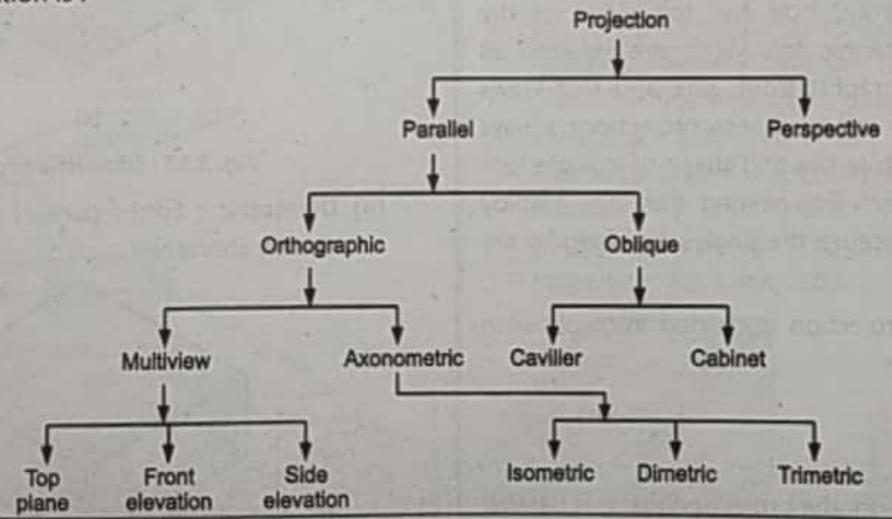
∴ Resultant transformation matrix will be,

$$R_S = [T]\,[R_{xy}]\,[R_e]\,[R_{xy}]^{-1}\,[T]^{-1}$$

3.4 PROJECTION

- The process of representing a three dimensional object or scene into two dimensional medium is referred as projection.

Hierarchy of projection is :



- The plane geometric projections of objects are formed by the intersection of lines referred as projectors with a plane called the projection plane. Projectors are nothing but lines form an arbitrary point called as

center of projection. In three dimensional space, if the center of projection is located at a finite point, then the result is a perspective projection. If the projectors are parallel and the center of projection is located at infinity then the result is a parallel projection.

3.4.1 Parallel Projection

- This technique is used in drawing or drafting for producing scale drawings of three dimensional objects. This method is very useful for obtaining the accurate views of the various sides of an object. It also preserves the relative dimensions of objects. But the drawback of parallel projection is that, it does not give a realistic representation of the appearance of three dimensional object.

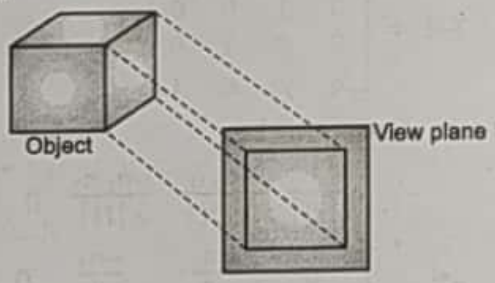


Fig. 3.10

- In parallel projection, z coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.
- Parallel projection is further classified into two types :
  1. Orthographic projection
  2. Oblique projection

1. Orthographic Projection :

- In orthographic projection, the direction of projection is perpendicular to the projection plane. It is used in the projection of front, side and top views of the object. The orthographic top views are referred as "planes" and orthographic front, side and rear views are referred as "elevations". These projections always show the correct or true size and shape of a single face or plane of an object. Engineering drawings employ these projections because the angles and lengths are accurately depicted.

  The orthographic projection is divided into following types :

  (i) Multi-view.

  (ii) Axonometric.

- In multi-view projection, the projection plane is parallel to the principal plane. It is categorized into three types viz. Three views, Auxiliary views and Sectional views.

- In Axonometric projection, the projection plane is not parallel to the principle plane. This projection can display more than one face of an object.
- The axonometric projection is further classified as :

  (i) Isometric.

  (ii) Diametric.

  (iii) Trimetric projection.

  (i) Isometric : Projections are aligned in such a way that all the edges will appear shortened by same distance. Assume object as cube then form all side looks like square, but using isometric ortho projection. We can see more than one face. All principal axes are shortened equally, so that relative proportions are maintained.

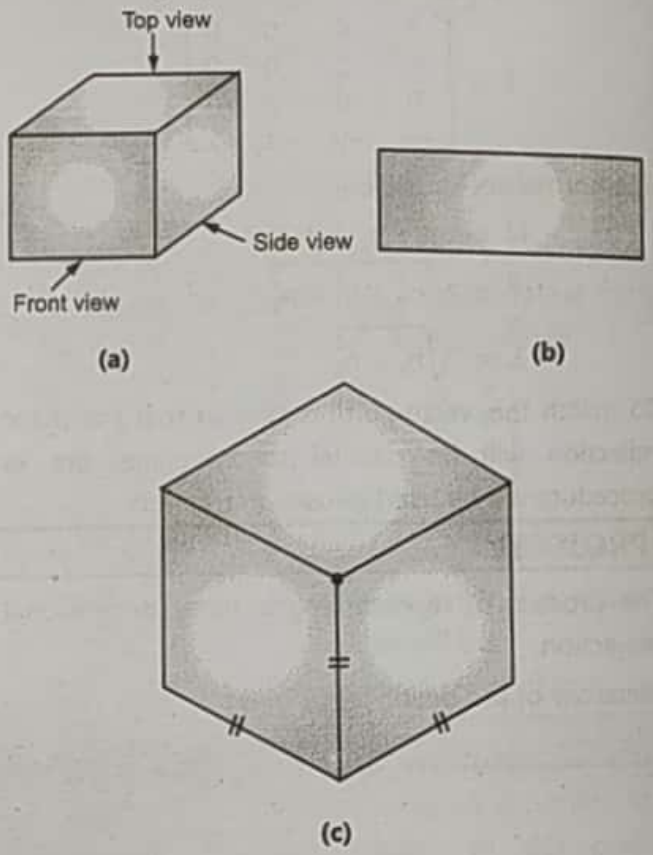


Fig. 3.11 : Isometric projection

(ii) Diametric : Edges parallel to only two axes are equally shortened.

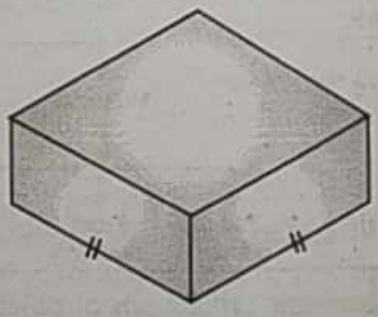


Fig. 3.12 : Diametric projection

(iii) Trimetric : None of the three edges are equally shortened.

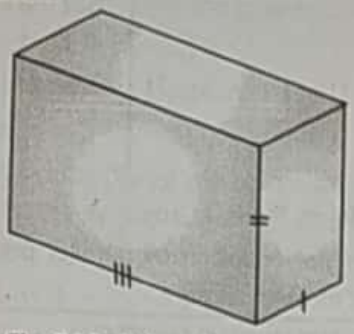


Fig. 3.13 : Trimetric projection

The most commonly used axonometric orthographic projection is the isometric projection. It can be generated by aligning the view plane so that it intersects each coordinate axis in which the object is defined at the same distance from the origin.

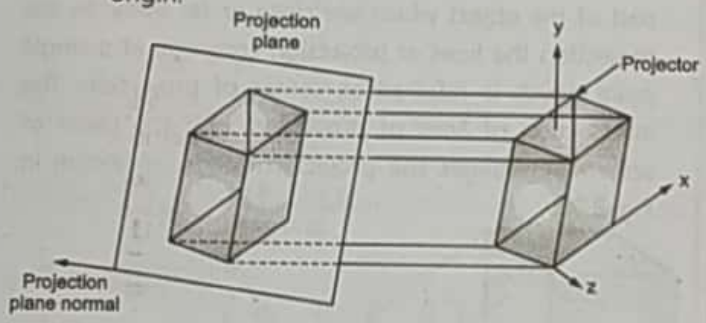


Fig. 3.14

As shown in Fig. 3.14 the isometric projection is obtained by aligning the projection vector with the cube diagonal. It uses an useful property that all the principle axes are equally foreshortened, allowing measurements along the axes to be made to the same scale hence the name is for equal, metric for measure.

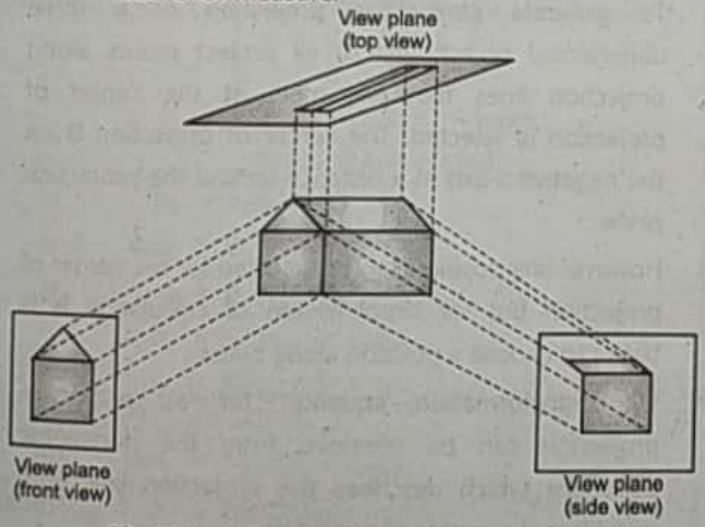


Fig. 3.15 : Orthographic parallel projection

2. Oblique Projection :

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.

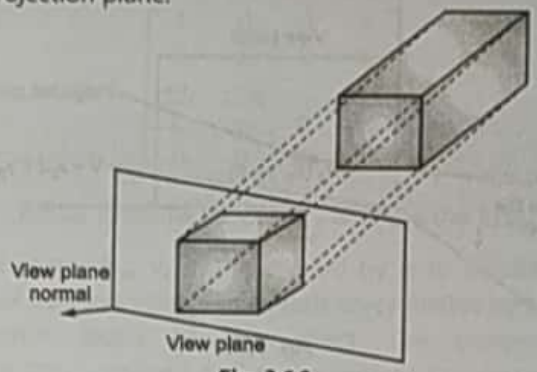


Fig. 3.16

- As shown in Fig. 3.16, the view plane normal and direction of projection are not the same.

The oblique projection is further classified into :

(i) Cavalier projection,

(ii) Cabinet projection.

- For the cavalier projection the direction of projection makes a 45° angle with the view plane. As a result, the projection of a line perpendicular to the view plane has the same length as the line itself i.e. there is no foreshortening. Fig. 3.17 shows cavalier projection of a unit cube with $\alpha = 45°$ and $\alpha = 30°$.

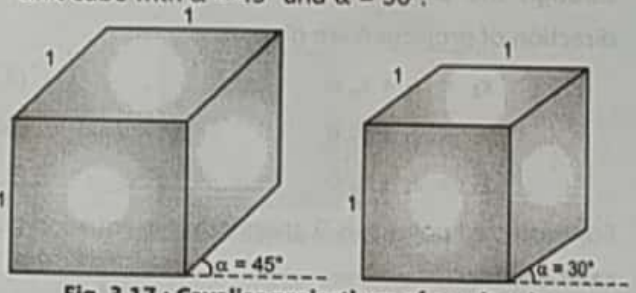


Fig. 3.17 : Cavalier projections of a unit cube

- In the cabinet projection, the direction of projection makes an angle of arc tan (2) = 63.4 with the view plane. For this angle, lines perpendicular to the viewing surface are projected at one half their actual length. Cabinet projections appear more realistic than cavalier because of the reduction in the length of perpendiculars. Fig. 3.18 below shows the cabinet projections for a unit cube.

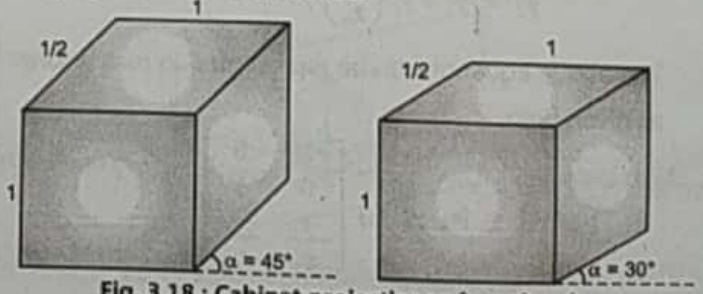


Fig. 3.18 : Cabinet projections of a unit cube

3.4.2 General Equation of Parallel Projection

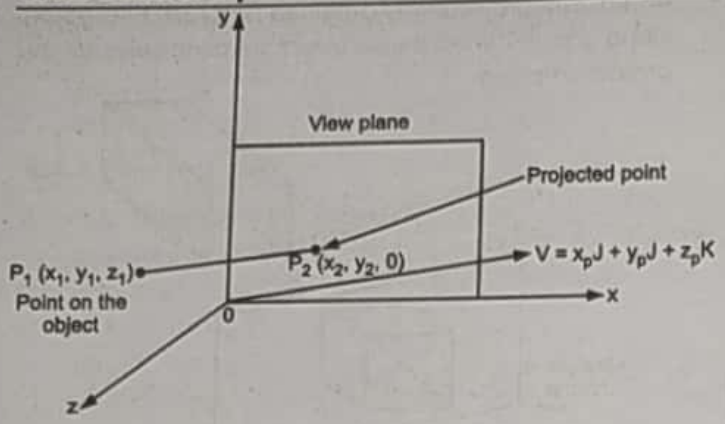


Fig. 3.19

- In a general parallel projection, any direction may be selected for the lines of projection. Suppose that the direction of projection is given by the vector $[x_p, y_p, z_p]$ and that the object is to be projected onto the xy plane. If the point on the object is given as (x_1, y_1, z_1) then the projected point (x_2, y_2) can be determined as given below :
- The equations in the parametric form for a line passing through the projected point (x_2, y_2, z_2) and in the direction of projection are given as –

$$x_2 = x_1 + x_p u \quad \ldots (3.3)$$

$$y_2 = y_1 + y_p u \quad \ldots (3.4)$$

$$z_2 = z_1 + z_p u \quad \ldots (3.5)$$

For projected point z_2 is 0, therefore the equation (3.5) can be written as,

$$0 = z_1 + z_p u$$

$$u = \frac{-z_1}{z_p}$$

Substituting the value of u in equations (3.3),(3.4),

$$x_2 = x_1 + x_p \left(\frac{-z_1}{z_p}\right)$$

$$y_2 = y_1 + y_p \left(\frac{-z_1}{z_p}\right)$$

The above equation can be represented in matrix form as given below :

$$[x_2, y_2] = [x_1 \; y_1 \; z_1] \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \frac{-x_p}{z_p} & \frac{-y_p}{z_p} \end{bmatrix}$$

or in homogeneous co-ordinates.

$$[x_2, y_2, z_2 \; 1] = [x_1 \; y_1 \; z_1 \; 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-x_p}{z_p} & \frac{-y_p}{z_p} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

i.e. $P_2 = P_1 \cdot P_{arv}$ ... (3.6)

This is the general equation of parallel projection on xy plane in matrix form.

3.5 PERSPECTIVE PROJECTION

- In perspective projection, if the object is far away from the viewer then it appears smaller and it appears larger if the object is nearer to the viewer. This helps the viewer in determining depth cue. The depth cue is an indication of which portion of the image correspond to part of the object which are close or far away. In this projection the lines of projection converge at a single point which is referred as center of projection. The intersection of lines of projection with the plane of screen determines the projected image, as shown in Fig. 3.20.

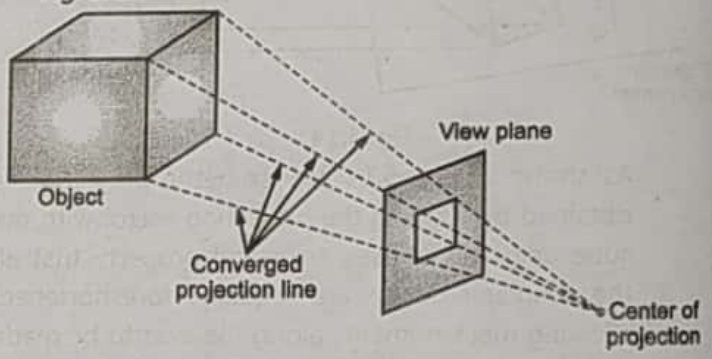


Fig. 3.20

- To generate perspective projection of a three dimensional object, first of all project points along projection lines that will meet at the center of projection is selected. The center of projection is on the negative z-axis at a distance behind the projection plane.
- However any position can be selected for the center of projection but for simplification of calculation it is better to choose a position along z-axis.
- The transformation equation for a perspective projection can be obtained from the parametric equations which describes the projection line from point P to the center of projection.

- If the center of projection is at (x_c, y_c, z_c) and point on the object is (x_1, y_1, z_1) then the projection ray will be,

$$x_1 = x_c + (x_1 - x_c)\,u$$
$$y = y_c + (y_1 - y_c)\,u$$
$$z = z_c + (z_1 - z_c)\,u$$

- The projected point (x_2, y_2) will be the point where this line intersects the xy plane. For this intersection point $z = 0$,

$$z = z_c + (z_1 - z_c)\,u$$
$$0 = z_c + (z_1 - z_c)\,u$$
$$\therefore \quad u = \frac{-z_c}{z_1 - z_c}$$
$$\therefore \quad x_2 = x_c + (x_1 - x_c)\left(\frac{-z_c}{z_1 - z_c}\right)$$
$$x_2 = x_c - z_c\frac{x_1 - x_c}{z_1 - z_c}$$
$$y_2 = y_c - z_c\frac{y_1 - y_c}{z_1 - z_c}$$
$$x_2 = \frac{x_c z_1 - x_1 z_c}{z_1 - z_c}$$
$$y_2 = \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}$$

In the matrix form :

$$P = \begin{bmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{bmatrix}$$

Consider point (x_1, y_1, z_1), in homogeneous coordinates it is $[x_1\omega_1 \;\; y_1\omega_1 \;\; z_1\omega_1 \;\; \omega_1]$

$$[x_2\omega_2 \;\; y_2\omega_2 \;\; z_2\omega_2 \;\; \omega_2] = [x_1\omega_1 \;\; y_1\omega_1 \;\; z_1\omega_1 \;\; \omega_1] \begin{bmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{bmatrix}$$
$$= [-x_1\omega_1 z_c + z_1\omega_1 x_c \;\; -y_1\omega_1 z_c + z_1\omega_1 y_c \;\; 0 \;\; z_1\omega_1 - z_c\omega_1]$$
$$\therefore \quad \omega_2 = z_1\omega_1 - z_c\omega_1$$
and $\quad z_2\omega_2 = 0$
$$\therefore \quad z_2 = 0$$
$$x_2\omega_2 = -x_1\omega_1 z_c + z_1\omega_1 x_c$$
$$\Rightarrow \quad x_2 = \frac{x_c z_1 - x_1 z_c}{z_1 - z_c}$$
And $\quad y_2\omega_2 = -y_1\omega_1 z_c + z_1\omega_1 y_c$
$$\Rightarrow \quad y_2 = \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}$$

The resulting point (x_2, y_2) is then the correctly projected point.

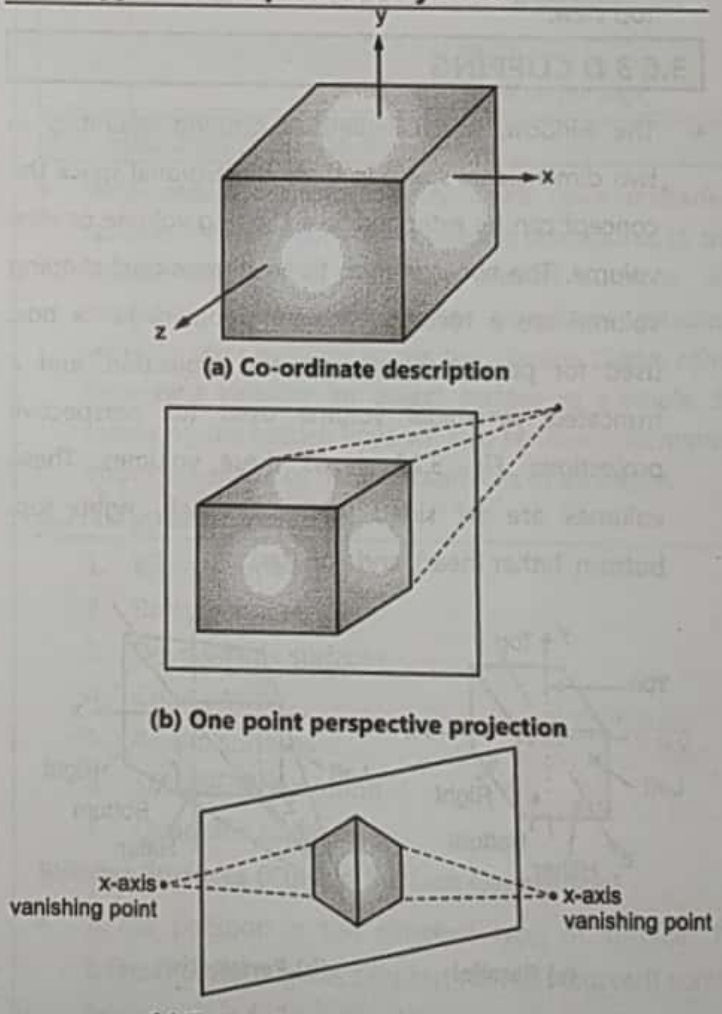
$\therefore$ The projection transformation is,

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-x_c}{z_c} & \frac{-y_c}{z_c} & 0 & \frac{-1}{z_c} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The change is the factor of $-\frac{1}{z_c}$. Because the first three coordinates $(x_\omega, y_\omega, z_\omega)$ is divided by ω to obtain the actual position, changing all four co-ordinates by some common factor has no effect. The perspective projection is defined as that the center of projection is located at the origin and the view plane is positioned at $z = d$. Thus, the transformation matrix is given by,

$$P_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.5.1 Types of Perspective Projection



(a) Co-ordinate description

(b) One point perspective projection

(c) Two point perspective projection

Fig. 3.21

- The perspective projections of any set of parallel lines which are not parallel to the projection plane converge to a vanishing point.

One-Point Perspective:

- This projection is based on single vanishing point. Object is placed so that one of its surface is parallel to the plane of projection.

Two-Point:

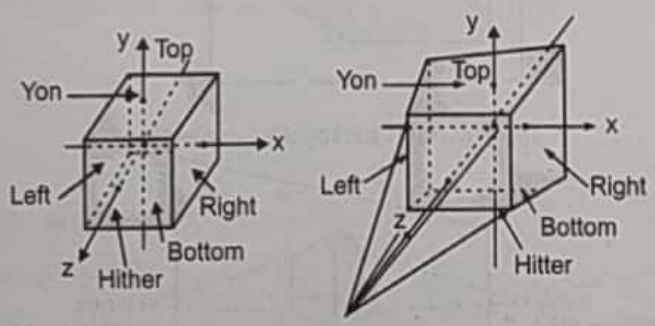
- In this projection, two surfaces of the object have vanishing points. It is used to draw the same objects as one-point.

Three-Point:

- The three point perspective is achieved by positioning the object so that none of its axes are parallel to the plane of projection. It is usually used for building from top view.

3.6 3 D CLIPPING

- The window, which served as clipping boundary in two-dimensional space In three dimensional space the concept can be extended to a clipping volume or view volume. The two common three dimensional clipping volume are a rectangular parallelepiped, i.e. a box, used for parallel or axonometric projection, and a truncated pyramidal volume used for perspective projections. Fig. 3.22 shows these volumes. These volumes are six sided with sides: left, right, top, bottom, hither (near), and yon (far).



(a) Parallel (b) Perspective

Fig. 3.22

- The two-dimensional concept of region codes can be extended to three dimensions by considering six sides and 6-bit code instead of four sides and 4-bit code. Like two-dimension, we assign the bit positions in the region code from right to left as

  Bit 1 = 1, if the end point is to the left of the volume.

  Bit 2 = 1, if the end point is to the right of the volume.

  Bit 3 = 1, if the end point is the below the volume.

  Bit 4 = 1, if the end point is above the volume.

  Bit 5 = 1, if the end point is in front of the volume.

  Bit 6 = 1, if the end Point is behind the volume.

- Otherwise, the bit is set to zero. As an example, a region code of 101000 identifies a point as above and behind the view volume, and the region code 000000 indicates a point within the view volume.
- A line segment can be immediately identified as completely within the view volume if both endpoints have a region code of 000000. If either endpoint of a line segment does not have a region code of 000000, we perform the logical AND operation on the two endpoint codes. If the result of this AND operation is nonzero then both endpoints are outside the view volume and line segment is completely invisible, On the other hand. If the result of AND operation is zero then line segment may be partially visible. In this case, it is necessary to determine this intersection of the line and the clipping volume.
- We have seen that determining the end point codes for a rectangular parallelepiped clipping volume is a straight forward extension of the two dimensional algorithm.
- However the perspective clipping volume shown in Fig. 3.23 (b) requires some additional processing. As shown in the Fig. 3.23 (a), the line connecting the center of projection and the center of the perspective clipping volume coincides with the z-axis in a right handed co-ordinate system.

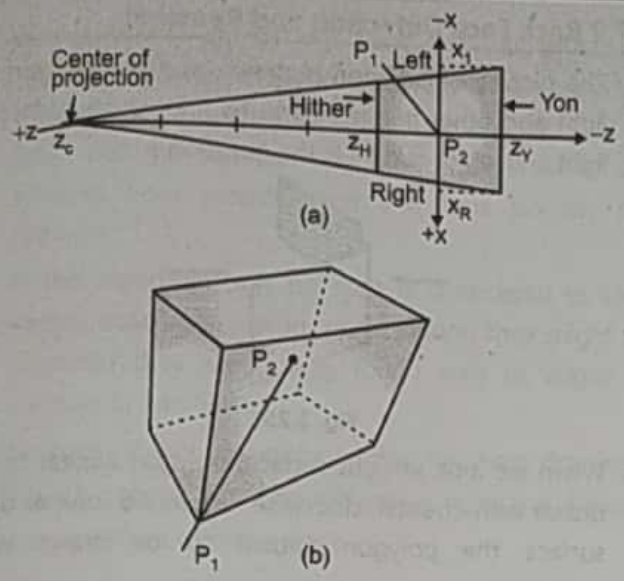


Fig. 3.23

- Fig. 3.23 shows a top view of the perspective clipping volume. The equation of the line which represents the right hand plane in this view can be given as

$$x = \frac{z - z_C}{z_Y - z_C} x_R = z\,\alpha_1 + \alpha_2$$

where, $\alpha_1 = \frac{x_R}{z_Y - z_C}$ and $\alpha_2 = -\alpha_1 z_C$

- This equation of right hand plane can be used to determine whether a point is to the right, on or to the left of the plane i.e., outside the volume, on the right hand plane, or inside the volume. Substituting the x and y coordinates of a point P into $x - z\,\alpha 1 - \alpha 2$ gives the following results.

$f_R = x - z\,\alpha_1 - \alpha_2$

> 0 if P is to the right of the right plane.

$= 0$ if P is on the right plane.

< 0 if P is to the left of the right plane.

Similarly, we can derive. the test functions for left, top, bottom. hither and yon planes. Table 3.1. show the test functions.

Table 3.1 : Test functions for six planes of clipping volume

Plane	Test functions with results
Right	$f_R = x - z\,\alpha_1 - \alpha_2$ > 0 if P is to the right of the right plane. $= 0$ if P is on the right plane. < 0 if P is to the left of the right plane. where $\alpha_1 = \frac{x_R}{z_Y - z_C}$ and $\alpha_2 = -\alpha_1 z_C$
Left	$f_L = x - z\,\beta_1 - \beta_2$ < 0 if P is to the left of the left plane. $= 0$ if P is on the left plane. > 0 if P is to the right of the left plane. where $\beta_1 = \frac{x_L}{x_Y - z_C}$ and $\beta_2 = -\beta_1 z_C$
Top	$f_T = y - z\,\gamma_1 - \gamma_2$ > 0 if P is above the top plane. $= 0$ if P is on the top plane. < 0 if P is below the top plane. where $\gamma_1 = \frac{y_T}{x_Y - z_C}$ and $\gamma_2 = -\gamma_1 z_C$
Bottom	$f_B = y - z\,\delta_1 - \delta_2$ < 0 if P is below the bottom plane. $= 0$ if P is on the bottom plane. > 0 if P is above the bottom plane. where $\delta_1 = \frac{y_B}{x_Y - z_C}$ and $\delta_2 = -\delta_1 z_C$
Hither	$f_H = z - z_H$ > 0 if P is in front of the hither plane. $= 0$ if P is on the hither plane. < 0 if P is behind the hither plane.
Yon	$f_Y = z - z_Y$ < 0 if P is behind the yon plane. $= 0$ if P is on the yon plane. > 0 if P is in front of the yon plane.

3.7 HIDDEN SURFACES

- When objects are to be displayed with color or shaded surface, we apply surface– rendering procedures to the visible surfaces so that the hidden surfaces are obscured. Some visible surface algorithms establish visibility pixel by pixel across the viewing plane, other determine visibility for object surface as a whole. By removing the hidden lines we also remove information about the shape of the back surfaces of an object.

3.7.1 Types of Surfaces

1. Bilinear surfaces
2. Ruled surfaces
3. Developable surfaces
4. Coon patch
5. Sweep surfaces
6. Surface of revolution
7 Quadratic surfaces

Bilinear Surfaces (Curved Surface Generation)

- A flat polygon is the simplest type of surface. The bilinear surface is the simplest non flat (curved) surface because it is fully defined by means of its four corner points. It is discussed here because its four boundary

curves are straight lines and because the coordinates of any point on this surface are derived by linear interpolations. Since this patch is completely defined by its four corner points, it cannot have a very complex shape. Nevertheless it may be highly curved. If the four corners are coplanar, the bilinear patch defined by them is flat. Let the corner points be the four distinct points P00, P01, P10, and P11. The top and bottom boundary curves are straight lines and are easy to calculate.

They are

$$P(u, 0) = (P10 - P00)u + P00 \text{ and } P(u, 1)$$
$$= (P11 - P01)u + P01.$$

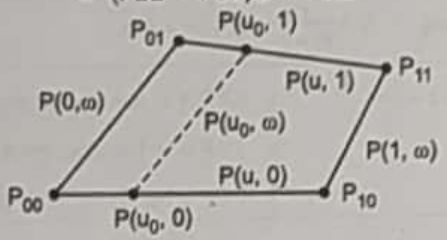


Fig. 3.24

- To linearly interpolate between these boundaries curves, we first calculate two corresponding points P(u0, 0) and P(u0, 1), one on each curve, then connect them with a straight line P(u0, w). The two points are P(u0, 0) = (P10 – P00)u0 + P00 and P(u0,1) = (P11 – P01)u0 + P01, and the straight segment connecting them is P(u0, w) = (P(u0, 1) – P(u0, 0))w + P(u0, 0) = [(P11 – P01)u0 + P01 –(P10 – P00)u0 + P00]w+ (P10 – P00)u0 + P00.

Bezier Surfaces

- To create a Bezier surface, we blend a mesh of Bezier curves using the blending function

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

Where j and k are points in parametric space and $P_{x,y}$ represents the location of the knots in real space. The Bezier functions specify the weighting of a particular knot. They are the Bernstein coefficients.

- By controlling light intensity in different directions.
- The directionality is provided by $\cos^n p$, p is angle from central direction warn model can be used for spot lighting.

3.7.2 Back Face Defection and Removal

- The picture of polygon is drawn as one side pointed light and other painted dark. To find which surface is light and which is dark is the question.

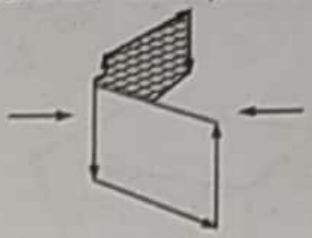


Fig. 3.25

- When we look at light surface, polygon appear to be drawn with counter clockwise. When we look at dark surface the polygon appear to be drawn with clockwise.

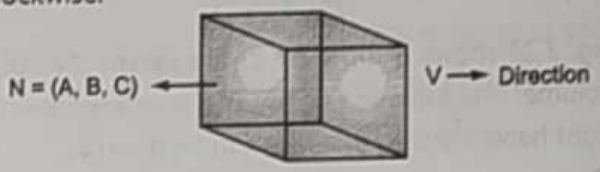


Fig. 3.26

The direction of light face is

$$NV > 0$$

N – Normal vector to polygon surface with Cartesian components (A, B, C)

V – Vector in the viewing direction

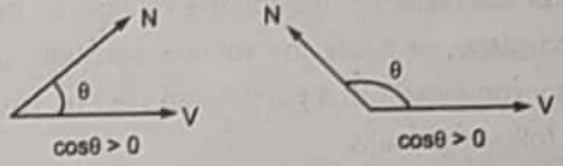


Fig. 3.27

- Cosine angle between to rectors
- If the dot product is positive, polygon faces towards viewer. Otherwise it faces away.

3.7.3 Back Face Removal Algorithm

- Hidden-line removal is a costly process hence it is advisable to apply easy tests to simplify the problem as much as possible before undertaking a thorough analysis.
- Back-face removal is a simple test which can be performed to eliminate most of the faces which cannot be seen.
- This test identifies surfaces which face away from the viewer. The back of the object cannot be visible because bulk of the object is in the way.

- This does not completely solve the hidden-surface problem because still the front face of an object is obscured by a second object or by another part of itself. But the test can remove roughly half of the surfaces from consideration and thus simplify the problem.
- In this algorithm only polygon is considered as lines cannot obscure anything and although they might be obscured they are usually found only as edges of surfaces of an object.
- Because of this, polygons suffice for most drawings. Now, a polygon has two surfaces, a front and a black, just as a piece of paper does.
- We might picture our polygons with one side painted light and the other painted dark. But the question is "how to find which surface is light or dark" ? When we are looking at the light surface, the polygon will appear to be drawn with counter clockwise pen motions and when we are looking at the dark surface the polygon will appear to be drawn with clockwise pen motions as shown below in Fig. 3.28.

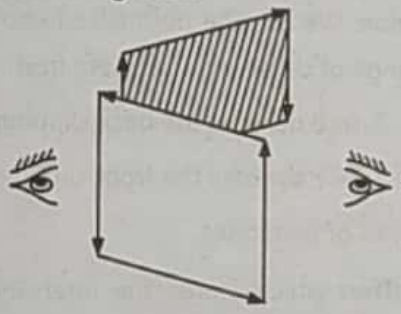


Fig. 3.28 : Drawing Directions

- Let us assume that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to the air, the dark faces meet the material inside the object.
- This means that when we look at an object face from the outside, it will appear to be drawn counter clockwise as shown in below figure Fig. 3.29.

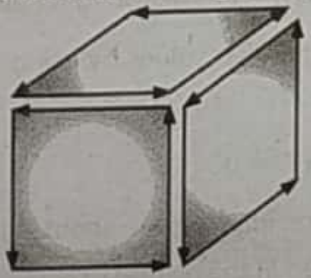


Fig. 3.29 : Exterior Surfaces are Coloured Light

- If a polygon is visible, the light surface should face towards us and the dark surface should face away from us. Therefore, if the direction of the light face is pointing towards the viewer, the face is visible (a front face), otherwise the face is hidden (a back face) and should be removed.
- The direction of the light face can be identified by examining the light.

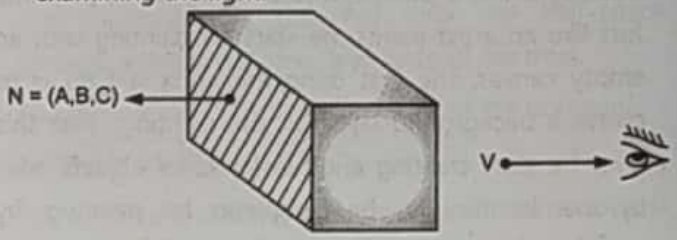


Fig. 3.30

$$N \cdot V > 0$$

Where, N : Normal vector to the polygon surface with Cartesian components (A, B, C)

V : A vector in the viewing direction from the eye or camera position

- We know that the dot product of two vectors gives the product of the lengths of the two vectors times the cosine of the angle between them. This cosine factor is important to us because if the vectors are in the same direction ($0 \le \theta < \pi/2$) then the cosine is positive and the overall dot product is positive. However, if the directions are opposite ($\pi/2 < \theta \le \pi$) then the cosine and the overall dot product is negative.

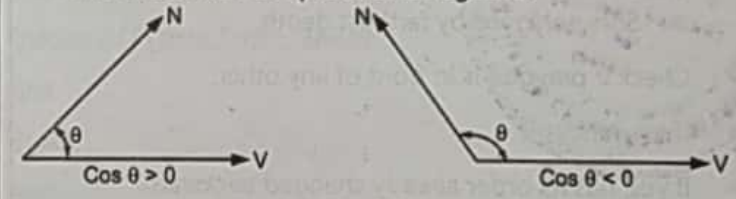


Fig. 3.31

Cosine Angles Between Two Vectors :

- If the dot product is positive, then the polygon faces towards the viewer otherwise it faces away and should be removed.
- In case, if object description has been converted to projection co-ordinates and our viewing direction is parallel to the viewing Z_V axis then $V = (0, 0, V_Z)$ and

$$V \cdot N = V_Z C$$

So that we only have to consider the sign of C, the Z component of the normal vector N. Now, if the Z

component is positive then the polygon faces towards the viewer, if negative it faces away.

3.7.4 Depth Sorts Painter's Algorithm

- The techniques used by these algorithms are image space and object space.
- The name of this algorithm is Painter's because it's working is like a painter who creating an oil painting. Just like an artist paints, he start his painting with an empty canvas, the first thing the artist will do is to create a background layer for the painting, after this layer he start creating another layers of objects one-by-one. In this way he completes his painting, by covering the previous layer partially or fully according to the requirement of the painting.
- This algorithm is basically used to paint the polygons in the view plane by considering their distance from the viewer. The polygons which are at more distance from the viewer are painted first. After that the nearer polygons are started painted on or over more distant polygons according to the requirement.
- In this algorithm the polygons or surfaces in the scene are firstly scanned or then painted in the frame buffer in the decreasing distance from view point of the viewer starting with the polygons of maximum depth or we can say minimum z-value.
  - Sort polygons by farthest depth.
- Check if polygon is in front of any other.
- If no, render it.
- If yes, has its order already changed backward?
  1. If no, render it.
  2. If yes, break it apart.
- Which polygon is in front? Our strategy: apply a series of tests.
  1. First tests are cheapest
  2. Each test says poly1 is behind poly2, or maybe.
- If min z of poly1 > max z poly2 ———1 in back.
- The plane of the polygon with smaller z is closer to viewer than other polygon.

(a,b,c)*(x,y,z) > =d.

- The plane of polygon with larger z is completely behind other polygon.
- Check whether they overlap in image
  1. Use axial rectangle test.
  2. Use complete test.

Problem Cases: Cyclic and Intersecting Objects

- Solution : split polygons

Advantages of Painter's Algorithm

- Simple
- Easy transparency

Disadvantages

- Have to sort first.
- Need to split polygons to solve cyclic and intersecting objects.

3.7.5 Depth Buffer-(Z) Algorithm

- Here, the depth of the surface is given by the coordinates. Algorithm compares the depth of each pixel position. We use the normalized coordinates. So that the range of depth(z) values vary from 0 to 1.

  Z = 0 denotes the back clipping plane.

  Z = 1 denotes the front clipping plane.

We use two types of memories.

1. **Frame Buffer:** which stores the intensity values for each pixel position and
2. **Z–Buffer:** which stores the depth of each pixel (x,y) position.

- Algorithm keeps track of the minimum depth value.

Algorithm

- Initialize depth (x,y)=0

  Frame buffer (x,y) = I background for all (x,y)
- Compute the z–Buffer values by using the equation of the plane.

  Ax + By + Cz + D = 0

  [here, we store information about all the polygonal surface included in the picture.]

The pixels are scanned by the scaline incremental method.

$Z = -1/C\ (Ax+By+D)$ i.e. for any pixel position (Xk,Yk) the depth

(Xk,Yk) = Zk

$Zk = -1/C(Axk+Byk+C)$ The next pixel position is at (Xk+1, Yk) or (Xk,Yk–1)

$Zk+1 = -1/C(A(xk+1)+Byk+D)$

$Zk+1 = -1/C(A(xk+1)+Byk+D) - A/C$

$Zk+1 = -Zk-A/C$ calculates the values of depth recursively.

- Similarly, the depth values down the edges of intersection of the polygon surface and the scanline are given as calculated as follows:

Let $y = mx+c$ is the example of the left most intersecting edge then from (Xk,Yk)→(X', Yk–1) along the edge gives,

$Yk = mXk+C\ 1= m(Xk-X')$

$Yk-1 = mX'+C\ X'= Xk -1/m$

The depth value then becomes,

$Zk+1 = -1/C\ (A\ (Xk-1/m) + Byk-1 + D)$

$Zk+1 = -1/C(AXk + Byk - B+D)+ (A/M)/C$

$Zk+1 = -1/C(AXk + Byk +D)+B/C+ (A/M)/C$

$Zk+1 = Zk+ ((A/M)+B)/C$

- If the calculated depth value is Zk+1 and if at (x,y) pixel position

$Z >$ calculated depth (x,y) Then,

Set the depth value as

depth(x,y) = z and frame buffer (x,y)= I surface.

- Repeat steps 2 and 3, till all the polygonal surfaces are processed.

3.7.6 Binary Space Partitioning

- Suitable for a static group of 3D polygon to be viewed from a number of view points.
- Based on the observation that hidden surface elimination of a polygon is guaranteed if all polygons on the other side of it as the viewer is painted first, then itself, then all polygons on the same side of it as the viewer.

1. The Algorithm First Build the BSP Tree:

- A root polygon is chosen (arbitrarily) which divides the region into 2 half–spaces (2 nodes => front and back).
- A polygon in the front half–space is chosen which divides the half–space into another 2 half spaces.
- The subdivision is repeated until the half–space contains a single polygon (leaf node of the tree).
- The same is done for the back space of the polygon.

2. To Display a BSP Tree:

- See whether the viewer is in the front or the back half–space of the root polygon.
- If front half–space then first display back child (subtree) then itself, followed by its front child / subtree.
- The algorithm is applied recursively to the BSP tree.

BSP Algorithm

```
Procedure DisplayBSP(tree: BSP_tree)
Begin
If tree is not empty then
If viewer is in front of the root then
Begin
DisplayBSP(tree.back_child)
displayPolygon(tree.root)
DisplayBSP(tree.front_child)
End
Else
Begin
DisplayBSP(tree.front_child)
displayPolygon(tree.root)
DisplayBSP(tree.back_child)
End
End
```

Discussion

- Back face removal is achieved by not displaying a polygon if the viewer is located in its back half–space.
- It is an object space algorithm (sorting and intersection calculations are done in object space precision).

- If the view point changes, the BSP needs only minor re-arrangement.
- A new BSP tree is built if the scene changes.
- The algorithm displays polygon back to front (Depth-sort).

3.7.7 Area Subdivision (Warnock) Algorithms

- The area-subdivision method takes advantage of area coherence in a scene by locating those view areas that represent part of a single surface. The total viewing area is successively divided into smaller and smaller rectangles until each small area is simple, i.e. it is a single pixel, or is covered wholly by a part of a single visible surface or no surface at all.
- The procedure to determine whether we should subdivide an area into smaller rectangle is:

1. We first classify each of the surfaces, according to their relations with the area:

- **Surrounding Surface :** A single surface completely encloses the area.
- **Overlapping Surface :** A single surface that is partly inside and partly outside the area.
- **Inside Surface :** A single surface that is completely inside the area.
- **Outside Surface :** A single surface that is completely outside the area. To improve the speed of classification, we can make use of the bounding rectangles of surfaces for early confirmation or rejection that the surfaces should be belong to that type.

2. Check the result from 1, that, if any of the following condition is true, then, no subdivision of this area is needed.

- All surfaces are outside the area.
- Only one surface is inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.
- For cases b and c, the color of the area can be determined from that single surface.
- Fig. 3.32 shows scan-line method for surface removal
- Active edge list for scan-line 1 – AD, BC, EH, FG edges.
- Active edge list for scan-line 2 – AD, EH, BC, FG

S_1 is ON – Along the scan-lines between edges AD and BC.

S_2 is ON – EH and FG

Along scan-line 2 : S_1 is ON AD and EH edge

Flag for both surfaces – EH and DC edge ON.

Hence, depth of S_1 is less than S_2 so, intensities of surface S_1 are loaded into frame buffer.

Scan-Line :

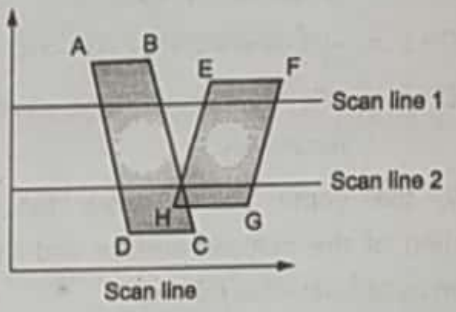


Fig. 3.32

- Fig. 3.32 shows scan-line method for surface removal.
- Active edge list for scan-line 1.

AD, BC, EH, FG edges

- Active edge list for scan-line 2.

AD, EH, BS, FG

S_1 is ON – along the scan-lines between edges AD and BC.

S_2 is ON – EH and FG.

S_1 is ON AD and EH edge

Flag for both surfaces ON – EH and DC.

Hence, depth of S_1 is less than S_2 so, intensities of surface S_1 are loaded into frame buffer.

3.8 GENERATION OF SOLIDS (SOLID MODELING)

- Solid modeling (or modelling) is a consistent set of principles for mathematical and computer modeling of three-dimensional solids. Solid modeling is distinguished from related areas of geometric modeling and computer graphics by its emphasis on physical fidelity. Together, the principles of geometric and solid modeling form the foundation of 3D-computer-aided design and in general support the creation, exchange, visualization, animation, interrogation, and annotation of digital models of physical objects.

- The use of solid modeling techniques allows for the automation of several difficult engineering calculations that are carried out as a part of the design process. Simulation, planning, and verification of processes such as machining and assembly were one of the main catalysts for the development of solid modeling. More recently, the range of supported manufacturing applications has been greatly expanded to include sheet metal manufacturing, injection molding, welding, pipe routing, etc. Beyond traditional manufacturing, solid modeling techniques serve as the foundation for rapid prototyping, digital data archival and reverse engineering by reconstructing solids from sampled points on physical objects, mechanical analysis using finite elements, motion planning and NC path verification, kinematic and dynamic analysis of mechanisms, and so on. A central problem in all these applications is the ability to effectively represent and manipulate three-dimensional geometry in a fashion that is consistent with the physical behavior of real artifacts. Solid modeling research and development has effectively addressed many of these issues, and continues to be a central focus of computer-aided engineering.

3.9 MATHEMATICAL FOUNDATIONS

- The notion of solid modeling as practiced today relies on the specific need for informational completeness in mechanical geometric modeling systems, in the sense that any computer model should support all geometric queries that may be asked of its corresponding physical object. The requirement implicitly recognizes the possibility of several computer representations of the same physical object as long as any two such representations are consistent. It is impossible to computationally verify informational completeness of a representation unless the notion of a physical object is defined in terms of computable mathematical properties and independent of any particular representation. Such reasoning led to the development of the modeling paradigm that has shaped the field of solid modeling as we know it today.
- All manufactured components have finite size and well behaved boundaries, so initially the focus was on mathematically modeling rigid parts made of homogeneous isotropic material that could be added or removed. These postulated properties can be translated into properties of subsets of three-dimensional Euclidean space. The two common approaches to define solidity rely on point-set topology and algebraic topology respectively. Both models specify how solids can be built from simple pieces or cells.

3.9.1 Solid Representation Schemes

- Based on assumed mathematical properties, any scheme of representing solids is a method for capturing information about the class of semi-analytic subsets of Euclidean space. This means all representations are different ways of organizing the same geometric and topological data in the form of a data structure. All representation schemes are organized in terms of a finite number of operations on a set of primitives. Therefore, the modeling space of any particular representation is finite, and any single representation scheme may not completely suffice to represent all types of solids. For example, solids defined via combinations of regularized boolean operations cannot necessarily be represented as the sweep of a primitive moving according to a space trajectory, except in very simple cases. This forces modern geometric modeling systems to maintain several representation schemes of solids and also facilitate efficient conversion between representation schemes.
- Below is a list of common techniques used to create or represent solid models. Modern modeling software may use a combination of these schemes to represent a solid.

Primitive Instancing

- This scheme is based on notion of families of object, each member of a family distinguishable from the other by a few parameters. Each object family is called a generic primitive, and individual objects within a family are called primitive instances. For example, a family of bolts is a generic primitive, and a single bolt specified by a particular set of parameters is a primitive instance. The distinguishing characteristic of pure parameterized instancing schemes is the lack of means for combining instances to create new structures which represent new and more complex objects. The other main drawback of this scheme is the difficulty of

writing algorithms for computing properties of represented solids. A considerable amount of family-specific information must be built into the algorithms and therefore each generic primitive must be treated as a special case, allowing no uniform overall treatment.

Spatial Occupancy Enumeration

- This scheme is essentially a list of spatial cells occupied by the solid. The cells, also called voxels are cubes of a fixed size and are arranged in a fixed spatial grid (other polyhedral arrangements are also possible but cubes are the simplest). Each cell may be represented by the coordinates of a single point, such as the cell's centroid. Usually a specific scanning order is imposed and the corresponding ordered set of coordinates is called a spatial array. Spatial arrays are unambiguous and unique solid representations but are too verbose for use as 'master' or definitional representations. They can, however, represent coarse approximations of parts and can be used to improve the performance of geometric algorithms, especially when used in conjunction with other representations such as constructive solid geometry.

Cell Decomposition

- This scheme follows from the combinatoric (algebraic topological) descriptions of solids detailed above. A solid can be represented by its decomposition into several cells. Spatial occupancy enumeration schemes are a particular case of cell decompositions where all the cells are cubical and lie in a regular grid. Cell decompositions provide convenient ways for computing certain topological properties of solids such as its connectedness (number of pieces) and genus (number of holes). Cell decompositions in the form of triangulations are the representations used in 3d finite elements for the numerical solution of partial differential equations. Other cell decompositions such as a Whitney regular stratification or Morse decompositions may be used for applications in robot motion planning.

Boundary Representation

- In this scheme a solid is represented by the cellular decomposition of its boundary. Since the boundaries of solids have the distinguishing property that they separate space into regions defined by the interior of the solid and the complementary exterior according to the Jordan-Brouwer theorem discussed above, every point in space can unambiguously be tested against the solid by testing the point against the boundary of the solid. Recall that ability to test every point in the solid provides a guarantee of solidity. Using ray casting it is possible to count the number of intersections of a cast ray against the boundary of the solid. Even number of intersections correspond to exterior points, and odd number of intersections correspond to interior points. The assumption of boundaries as manifold cell complexes forces any boundary representation to obey disjointedness of distinct primitives, i.e. there are no self-intersections that cause non-manifold points. In particular, the manifoldness condition implies all pairs of vertices are disjoint, pairs of edges are either disjoint or intersect at one vertex, and pairs of faces are disjoint or intersect at a common edge. Several data structures that are combinatorial maps have been developed to store boundary representations of solids. In addition to planar faces, modern systems provide the ability to store quadrics and NURBS surfaces as a part of the boundary representation. Boundary representations have evolved into a ubiquitous representation scheme of solids in most commercial geometric modelers because of their flexibility in representing solids exhibiting a high level of geometric complexity.

Surface Mesh Modeling

- Similar to boundary representation, the surface of the object is represented. However, rather than complex data structures and NURBS, a simple surface mesh of vertices and edges is used. Surface meshes can be structured (as in triangular meshes in STL files or quad meshes with horizontal and vertical rings of quadrilaterals), or unstructured meshes with randomly grouped triangles and higher level polygons.

Constructive Solid Geometry

- Constructive Solid Geometry (CSG) is a family of schemes for representing rigid solids as Boolean constructions or combinations of primitives via the regularized set operations discussed above. CSG and boundary representations are currently the most important representation schemes for solids. CSG representations take the form of ordered binary trees

where non-terminal nodes represent either rigid transformations (orientation preserving isometries) or regularized set operations. Terminal nodes are primitive leaves that represent closed regular sets. The semantics of CSG representations is clear. Each subtree represents a set resulting from applying the indicated transformations/regularized set operations on the set represented by the primitive leaves of the subtree. CSG representations are particularly useful for capturing design intent in the form of features corresponding to material addition or removal (bosses, holes, pockets etc.). The attractive properties of CSG include conciseness, guaranteed validity of solids, computationally convenient Boolean algebraic properties, and natural control of a solid's shape in terms of high level parameters defining the solid's primitives and their positions and orientations. The relatively simple data structure and elegant recursive algorithms have further contributed to the popularity of CSG.

Sweeping

- The basic notion embodied in sweeping schemes is simple. A set moving through space may trace or sweep out volume (a solid) that may be represented by the moving set and its trajectory. Such a representation is important in the context of applications such as detecting the material removed from a cutter as it moves along a specified trajectory, computing dynamic interference of two solids undergoing relative motion, motion planning, and even in computer graphics applications such as tracing the motions of a brush moved on a canvas. Most commercial CAD systems provide (limited) functionality for constructing swept solids mostly in the form of a two dimensional cross section moving on a space trajectory transversal to the section. However, current research has shown several approximations of three dimensional shapes moving across one parameter, and even multi-parameter motions.

Implicit Representation

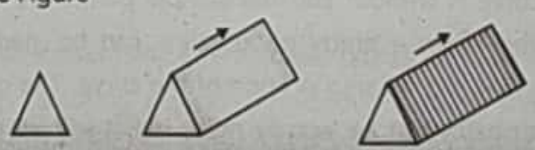
- A very general method of defining a set of points X is to specify a predicate that can be evaluated at any point in space. In other words, X is defined implicitly to consist of all the points that satisfy the condition specified by the predicate. The simplest form of a predicate is the condition on the sign of a real valued function resulting in the familiar representation of sets by equalities and inequalities. For example, if $f = a_x + b_y + c_z + d = 0$, the conditions $f(p) = 0$, $f(p) > 0$, and $f(p) < 0$ represent, respectively, a plane and two open linear half spaces. More complex functional primitives may be defined by boolean combinations of simpler predicates. Furthermore, the theory of R-functions allow conversions of such representations into a single function inequality for any closed semi analytic set. Such a representation can be converted to a boundary representation using polygonization algorithms, for example, the marching cubes algorithm.

Parametric and Feature-Based Modeling

- Features are defined to be parametric shapes associated with attributes such as intrinsic geometric parameters (length, width, depth etc.), position and orientation, geometric tolerances, material properties, and references to other features. Features also provide access to related production processes and resource models. Thus, features have a semantically higher level than primitive closed regular sets. Features are generally expected to form a basis for linking CAD with downstream manufacturing applications, and also for organizing databases for design data reuse. Parametric feature based modeling is frequently combined with constructive binary solid geometry (CSG) to fully describe systems of complex objects in engineering.

Sweep Representations:

- Sweep representations are used to construct three dimensional objects from two dimensional shape. There are two ways to achieve sweep: Translational sweep and Rotational sweep. In translational sweeps, the 2D shape is swept along a linear path normal to the plane of the area to construct three dimensional object. To obtain the wireframe representation we have to replicate the 2D shape and draw a set of connecting lines in the direction of shape, as shown in the Figure



(a) 2D shape (b) 3D object (c) Wire frame representation

Fig. 3.33: Translational sweep

- In rotational sweeps, the 2D shape is rotated about an a axis of rotation specified in the plane of 2D shape to produce three dimensional object. This is illustrated in Fig. 3.34.

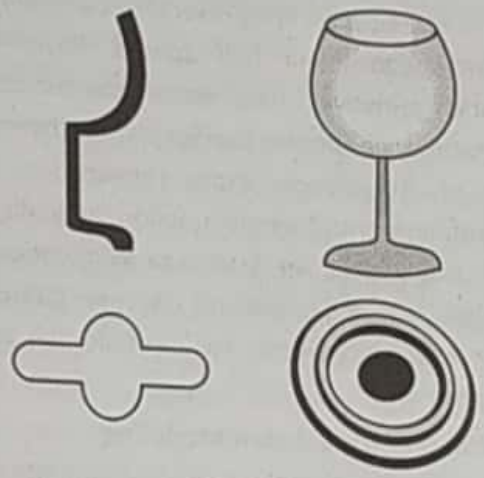


Fig. 3.34

- In general we can specify sweep constructions using any path. For translation we can vary the shape or size of the original 2D shape along the sweep path. For rotational sweeps, we can move along a circular path through any angular distance from 0° to 360°. These sweeps whose generating area or volume changes in size, shape or orientation as they are swept and that follow an arbitrary curved trajectory are called general sweeps. General sweeps are difficult to model efficiently for example, the trajectory and object shape may make the swept object intersect itself, making volume calculations complicated. Furthermore, general sweeps do not always generate solids. For example, sweeping a 2D shape in its own plane generates another 2D shape.

3.10 INTERPOLATION

- In this technique the curve is approximated by a number of small straight line segments. It is possible to draw an approximation to a curve, if an array of sample points are known. Then it can be guessed what the curve should look like between the sample points. If the curve is smooth and the sample points are close together, then a pretty good guess can be made to determine the missing portion of the curve. The guess will probably not be exactly right, but it will be close enough for appearances.

The process is :

- Fill in the portions of the unknown curve with pieces of known curve which pass through the nearby sample points. Since the known and unknown curves share these sample points in a local region, it is assumed that in this region, the two curves look pretty much alike.

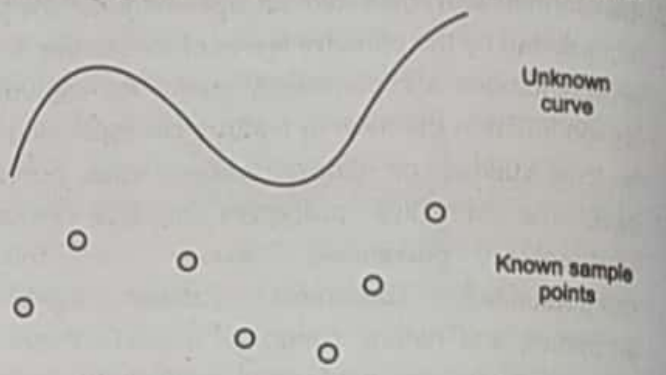


Fig. 3.35

- Fit a portion of the unknown curve with a curve that is known.

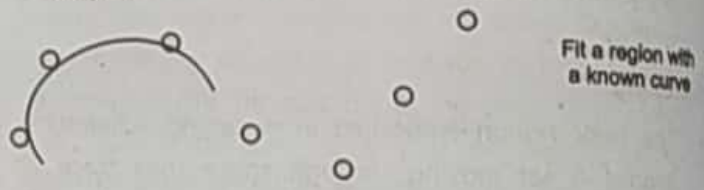


Fig. 3.36

- Now, fill in a gap between the sample points by finding the co-ordinates of point along the known approximating curve.

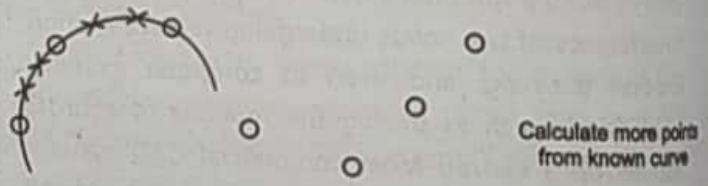


Fig. 3.37

- Connect these points with line segments.

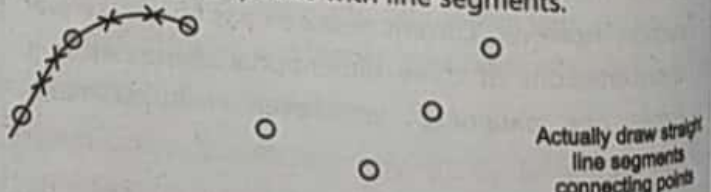


Fig. 3.38

- To invent a function which can be used for interpolation, consider a polynomial curve that will pass through n sample points.

$(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots\ldots, (x_n, y_n, z_n)$

- We will construct the function as the sum of terms, one term for each sample point. These functions can be given as,

$$f_x(u) = \sum_{i=1}^{n} x_i B_i(u)$$

$$f_y(u) = \sum_{i=1}^{n} y_i B_i(u)$$

$$f_z(u) = \sum_{i=1}^{n} z_i B_i(u)$$

- The function $B_i(u)$ is called **'bending function'**. For each value of parameter u, blending function determines how much the i^{th} sample point affects the position of curve. In other words, we can say that each sample points tries to pull the curve in its direction and the function $B_i(u)$ gives the strength of the pull. If for some value of u, $B_i(u) = 1$ for unique value of i (i.e. $B_i(u) = 0$ for other values of i), then i^{th} sample point has complete control of the curve and the curve will pass through i^{th} sample point. For different value of u, some other sample point may have complete control of the curve. In such case, the curve will pass through that point as well. In general, the blending functions control to each of the sample points in turn for different value of u. Let's assume that the first sample point (x_1, y_1, z_1) has complete control when u = −1, the second when u = 0, the third when u = 1 and so on i.e. when $u = -1 \Rightarrow B_1(u) = 1$ and 0 for u = 0, 1, 2, ..., n − 2.

When $u = 0 \Rightarrow B_2(u) = 1$ and 0 for u = −1, 1, ..., n − 2

:

:

:

When $u = (n - 2) \Rightarrow B_n(u) = 1$ and 0 for u = −1, 0, ..., (n − 1)

To get $B_1(u) = 1$ at u = −1 and 0 for u = 0, 1, 2, ..., n − 2, the expression for $B_i(u)$ can be given as,

$$B_1(u) = \frac{u(u-1)(u-2)\ldots[u-(n-2)]}{(-1)(-2)\ldots(1-n)}$$

where denominator term is a constant used. In general form i^{th} bending function which is 1 at u = i − 2 and 0 for other integers can be given as,

$$B_i(u) = \frac{(u+1)(u)(u-1)\ldots[u-(i-3)][u-(i-1)\ldots[u-(i-2)]}{(i-1)(i-2)(i-3)\ldots(1)(-1)(i-n)}$$

- The approximation of the curve using above expression is called **Lagrange interpolation.**
- From the above expression blending functions for four sample points can be given as,

$$B_1(u) = \frac{u(u-1)(u-2)}{(-1)(-2)(-3)}$$

$$B_2(u) = \frac{(u+1)(u-1)(u-2)}{1(-1)(-2)}$$

$$B_3(u) = \frac{(u+1)u(u-2)}{(2)(1)(-1)}$$

$$B_4(u) = \frac{(u+1)u(u-1)}{(3)(2)(1)}$$

- Using above blending functions, the expression for the curve passing through sampling points can be realized as follows :

$$x = x_1B_1(u) + x_2B_2(u) + x_3B_3(u) + x_4B_4(u)$$

$$y = y_1B_1(u) + y_2B_2(u) + y_3B_3(u) + y_4B_4(u)$$

$$z = z_1B_1(u) + z_2B_2(u) + z_3B_3(u) + z_4B_4(u)$$

- It is possible to get intermediate points between two sampling points between two sampling points by taking values of u between the values of u related to the two sample points under consideration. For example, we can find the intermediate points between second and third sample points for which values of u are 0 and 1 respectively; by taking value of u between 0 and 1, this is shown below.

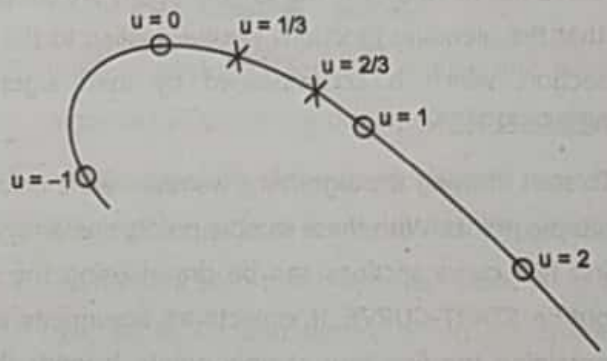


Fig. 3.39

- The subsequent intermediate points can be obtained by repeating the same procedure. Finally, the points obtained by this procedure are joined by small straight line segments to get the approximated curve.

3.11 INTERPOLATING ALGORITHMS

- To implement the curve-drawing program, the same blending function values are needed for each section of the curve that is drawn. If each section is approximated by three straight-line segments then each section will require the blending function values for u at 0, 1/3, 2/3 and 1 as shown below.

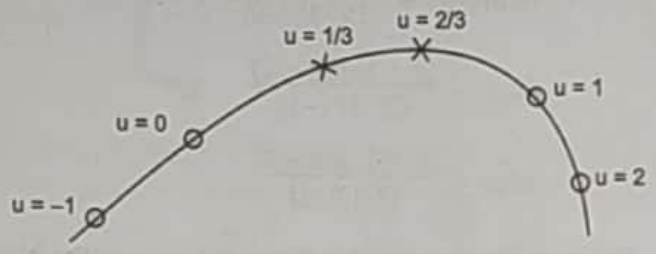


Fig. 3.40

- These values are calculated once and saved in an array for use in drawing each curve section.
- The first algorithm SET-SMOOTH allows the user to specify how many straight-line segments should be used to complete a section of the curve. This number will tell the u values at which to calculate and save the blending function values. The blending function values needed for the first and last sections of the curve are also calculated and stored in arrays.
- The second algorithm MAKE-CURVE multiplies the sample points and blending function which generates points on the approximation curve. These points are then connected by line segments. After a section of the curve has been drawn, the sample points are shifted so that the blending functions can be applied to the next section, which is accomplished by third algorithm NEXT-SECTION.
- To start drawing the algorithm, we require the first four sample points. With these sample points and arrays the first two curve sections can be drawn using the next routine START-CURVE. It expects an arguments array containing the first four sample points. It loads these points into another array and then the pen is positioned at the first sample point and then algorithm is used to draw the first two sections of the curve. The sample points are then shifted to prepare for drawing the next curve section.

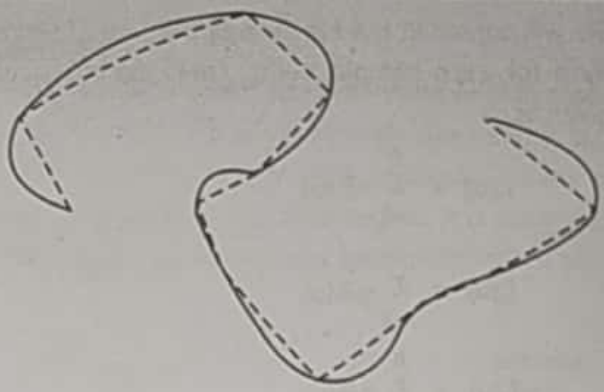


Fig. 3.41 : Interpolation smoothing

- Once the curve has been started new sections can be added one at a time. For each new sample point, a new section of the curve can be drawn. Since we are adding the fourth sample point while interpolating between the second and third sample points, the section of the curve being drawn always lags one sample point behind the points entered. The routine PUT-IN-SM is used to place sample points in arrays. The curve may be extended as desired by repeated calls to a routine CURVE-ABS but when we are ready to end it, we must process the last section. This can be done by using the END-CURVE routine, which takes as an argument the last point on the curve.

3.12 INTERPOLATING POLYGONS

- The blending function can be used to round the sides of a polygon. It is easier to deal with a polygon since no special initial or final section occurs. We just step around the polygon, smooth out each side by replacing it with several small line segments. We start with a polygon that has only few sides and end up with a polygon which has many more sides and appears smoother as shown below.

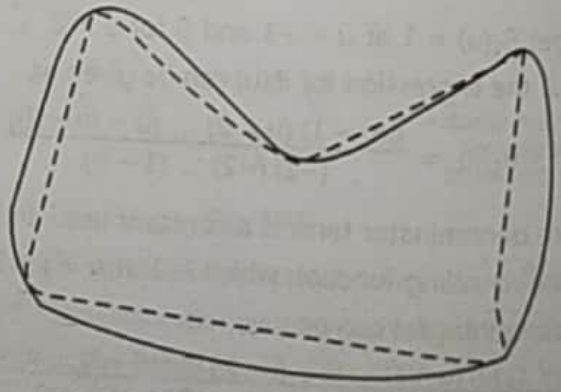


Fig. 3.42 : Smoothing of a polygon

EXERCISE

1. Obtain the 3-D transformation matrices for :
   (i) Translation,
   (ii) Scaling,
   (iii) Rotation about an arbitrary axis.
2. Give the classification of perspective parallel projection.
3. Explain parallel projection in detail with transformation matrix.
4. Derive the 3D primitive transformation for the following rotation :
   Rotate object about z-axis such that x-axis passes through a point $P(x_p, y_p, 0)$ in x-y plane.
5. Consider the square A(1, 0), B(0, 0), (0, 1), D(1, 1). Rotate the square ABCD by 45° anticlockwise about point A(1, 0).
6. Perform a 45° rotation of triangle A(0, 0), B(1, 1), C(5, 2).
   (i) About the origin,
   (ii) About P(−1, −1).
7. Explain with example, 3-D viewing transformation.
8. What is the concept of vanishing point in perspective projection.
9. Explain classification of parallel projection in detail. Discuss applications of parallel projections.
10. Consider the square A(2, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square anticlockwise direction followed by reflection about x-axis.
11. Explain the following 3-D transformation :
    (i) Rotation about all co-ordinate axis
    (ii) Rotation about any arbitrary axis.
12. Derive the general equation of parallel projection onto a given view plane in the direction of given projector.
13. Consider the square P(0, 0), Q(0, 10), R(10,10), S(10, 0). Rotate the square about fixed point. R(10, 10) by an angle 45° (anticlockwise) followed by scaling by 2 units in X direction and 2 units in Y direction.
14. What are parallel and perspective projections? Give classification of both.
15. Explain inverse transformation. Derive the matrix for inverse transformation and what is the concept of homogeneous co-ordinates.
16. Explain various steps to perform rotation about x-axis, y-axis and z-axis in 3D.
17. Explain :
    (i) 3-D co-ordinate system,
    (ii) 3-D primitives.
18. Explain the 3-D viewing process with various 3-D viewing parameters.
19. Show that the transformation matrix of reflection about a line y = x is equivalent to reflection relative to x-axis followed by anticlockwise rotation of 90°.
20. Drive transformation matrix for perspective projection.
21. Magnify the triangle with vertices A(0, 0), B(1, 1), C(5, 2) to twice its size as well as rotate it by 45°. Derive the translation matrices.
22. What is necessary for 3D clipping and windowing algorithm? Explain any one of 3D clipping algorithm.
23. A 3D cube of dimensions (length, breadth and height) 2 units each is placed in a 3D anti-clockwise axis system such that one of its vertex "A" is at origin (i.e. (0, 0, 0)) and vertex "F" in 3D space. Apply necessary transformation such that vertex F becomes the origin. Give complete mathematical formulation. Draw initial and final state of the cube.
24. Explain reflection about line?
25. What is mean by Transformation?

26. Compete homogeneous co-ordinate system and normalized co-ordinate system?
27. What is the need of homogeneous co-ordinates system ?
27. What is composite transformation?
28. Explain the inverse transformation?
29. How can you perform homogeneous co-ordinates for scaling?
30. What are the different steps for rotation about an arbitrary axis?
31. What is mean by shear and reflection?

✠ ✠ ✠

UNIT IV

GRAPHICAL USER INTERFACE

4.1 X-WINDOWS

- The X Window System (X11) is an open source, cross platform, client-server computer software system that provides a GUI in a distributed network environment.
- Used primarily on Unix variants, X versions are also available for other operating systems. Features of the X window system include network transparency, the ability to link to different networks, and customizable graphical capabilities. The X window system was first developed in 1984, as part of project Athena, collaboration between Stanford University and MIT. X.Org Foundation, an open group, manages the development and standardization of the X window system.
- The X Window System is also known simply as X, X11 or X Windows.
- The X Window System (X11, or simply X) is a windowing system for bitmap displays. X provides the basic framework for a GUI environment: drawing and moving windows on the display device and interacting with a mouse and keyboard. X does not mandate the user interface – this is handled by individual programs. As such, the visual styling of X-based environments varies greatly; different programs may present radically different interfaces.
- X is an architecture-independent system for remote graphical user interfaces and input device capabilities. Each person using a networked terminal has the ability to interact with the display with any type of user input device.
- In its standard distribution it is a complete, albeit simple, display and interface solution which delivers a standard toolkit and protocol stack for building graphical user interfaces on most Unix-like operating systems and OpenVMS, and has been ported to many other contemporary general purpose operating systems.
- X provides the basic framework, or primitives, for building such GUI environments: drawing and moving windows on the display and interacting with a mouse, keyboard or touch screen. X does not mandate the user interface; individual client programs handle this. Programs may use X's graphical abilities with no user interface. As such, the visual styling of X-based environments varies greatly; different programs may present radically different interfaces.
- Unlike earlier display protocols, X was specifically designed to be used over network connections rather than on an integral or attached display device. X features network transparency, which means an X program running on a computer somewhere on a network (such as the Internet) can display its user interface on an X server running on some other computer on the network. The X server is typically the provider of graphics resources and keyboard/mouse events to X clients, meaning that the X server is usually running on the computer in front of a human user, while the X client applications run anywhere on the network and communicate with the user's computer to request the rendering of graphics content and receive events from input devices including keyboards and mice.
- The fact that the term "server" is applied to the software in front of the user is often surprising to users accustomed to their programs being clients to services on remote computers. Here, rather than a remote database being the resource for a local app, the user's graphic display and input devices become resources made available by the local X server to both local and remotely hosted X client programs who need to share the user's graphics and input devices to communicate with the user.
- X's network protocol is based on X command primitives. This approach allows both 2D and (through extensions like GLX) 3D operations by an X client application which might be running on a different

computer to still be fully accelerated on the X server's display. For example, in classic OpenGL (before version 3.0), display lists containing large numbers of objects could be constructed and stored entirely in the X server by a remote X client program, and each then rendered by sending a single glCallList(which) across the network.

4.1.1 Client Server Model in X

- The client/server model in X system works in reverse to typical client/server model, where the client runs on the local machine and asks for services from the server. In X system, the server runs on the local machine and provides its display and services to the client programs. The client programs may be local or remotely exist over different networks, but appear transparently.
- X is used in networks of interconnected mainframes, minicomputers, workstations, and X Terminals. X window system consists of a number of interacting components, including:
  - ➢ **X Server:** Manages the display and input hardware. It captures command-based and graphics-based inputs from input hardware and passes it to the client application that requested it. It also receives inputs from the client applications and displays the output under guidance from windows manager. The only component that interacts with hardware is X server. This makes it easier to recode it as per the requirements of different hardware architectures.
  - ➢ **Windows Manager:** Is the client application that manages client windows. It controls the general operations of the window system like geometry, appearance, coordinates, and graphical properties of X display. Window manager can change the size and position of windows on the display and reshuffle windows in a window stack.
  - ➢ **X Client:** Is an application program that communicates with X server using X protocol. Xterm, Xclock, and Xcalc are examples of X clients. X manages its windows in a hierarchal structure. The shaded area that fills the entire screen is the root window. X client application windows are displayed on top of the root window and are often called the children of the root.

4.1.2 X Window System Protocols and Architecture

The X Client-Server Model and Network Transparency

- X is based on a client-server model. An X server program runs on a computer with a graphical display and communicates with various client programs. The server accepts requests for graphical output (windows) and sends back user input (keyboard, mouse).
- In X Window, the server runs on the user's computer, while the clients may run on a different machine. This is the reverse of the common configuration of client-server systems, where the *client* runs on the user's computer and the server runs on a remote computer. This reversal often confuses new X users. The X Window terminology takes the perspective of the program, rather than the end-user or the hardware: the remote programs connect to the X server display running on the local machine, and thus act as clients; the local X display accepts incoming traffic, and thus acts as a server.

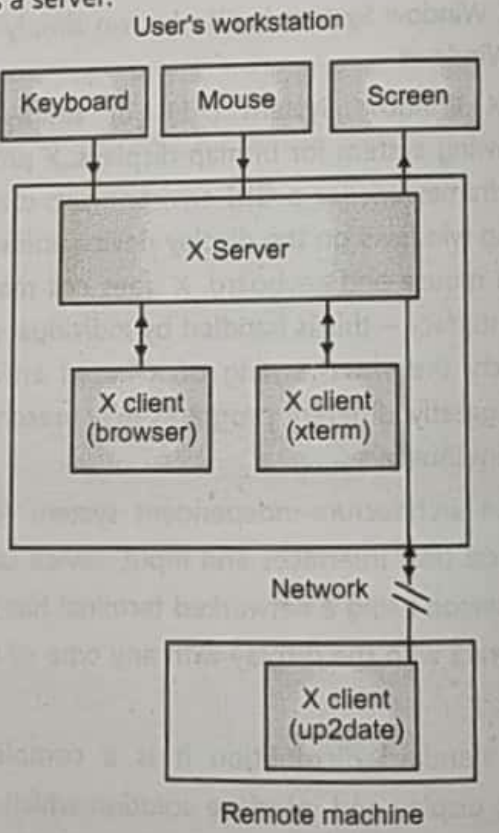


Fig. 4.1: X Client-Server example

Design Principles

Early principles of X as follows :

- Do not add new functionality unless an implementer cannot complete a real application without it.

- It is as important to decide what a system is not as to decide what it is. Do not serve all the world's needs; rather, make the system extensible so that additional needs can be met in an upwardly compatible fashion.
- The only thing worse than generalizing from one example is generalizing from no examples at all.
- If a problem is not completely understood, it is probably best to provide no solution at all.
- If you can get 90 percent of the desired effect for 10 percent of the work, use the simpler solution. (See also Worse is better.)
- Isolate complexity as much as possible.
- Provide mechanism rather than policy. In particular, place user interface policy in the clients' hands.

The first principle was modified during the design of X11 to: *"Do not add new functionality unless you know of some real application that will require it."* X has largely kept to these principles since.

X Window Core Protocol

- Communication between server and clients is done by exchanging packets over a network channel. The connection is established by the client, which sends the first packet. The server answers by sending back a packet stating the acceptance or refusal of the connection, or with a request for a further authentication. If the connection is accepted, the acceptance packet contains data for the client to use in the subsequent interaction with the server.
- After connection is established, four types of packets are exchanged by the client and the server over the channel:
  1. **Request:** The client requests information from the server or requests it to perform an action.
  2. **Reply:** The server responds to a request. Not all requests generate replies.
  3. **Event:** The server sends an event to the client, e.g., keyboard or mouse input, or a window being moved, resized or exposed.
  4. **Error:** The server sends an error packet if a request is invalid. Since requests are queued, error packets generated by a request may not be sent immediately.
- The X server provides a set of basic services. The client programs realize more complex functionalities by interacting with the server.

Attributes and Properties

- Every window has a predefined set of attributes and a set of properties, all stored in the server and accessible to the clients via appropriate requests. Attributes are data about the window, such as its size, position, background colour, etc. Properties are pieces of data that are attached to a window. Contrary to attributes, properties have no meaning at the level of the X Window core protocol. A client can store arbitrary data in a property of a window.
- A property is characterized by a name, a type, and a value. Properties are similar to variables in imperative programming languages, in that the application can create a new property with a given name and of a given type and store a value in it. Properties are associated to windows: two properties with the same name can exist on two different windows while having different types and values.
- Properties are mostly used for inter-client communication. For example, the property named `WM_NAME` is used for storing the name for the window; window managers typically read this property and display the name of the window at the top of it.
- The properties of a window can be shown using the `xprop` program. In particular, `xprop -root` shows the properties of the root window, which include the X resources (parameters of programs).

Events

- Events are packets sent by the server to the client to communicate that something the client may be interested in has happened. A client can request the server to send an event to another client; this is used for communication between clients. For example, when a client requests the text that is currently selected, an event is sent to the client that is currently handling the window that holds the selection.
- The content of a window may be destroyed in some conditions (for example, if the window is covered). Whenever an area of destroyed content is made visible, the server generates an `Expose` event to notify the client that a part of the window has to be drawn.
- Other events are used to notify clients of keyboard or mouse input, of the creation of new windows, etc.
- Some kinds of events are always sent to client, but most kinds of event are sent only if the client

previously stated an interest in them. This is because clients may only be interested in some kind of events. For example, a client may be interested in keyboard-related event but not in mouse-related events.

Colour Modes

- The way colors are handled in the X Window Systems sometimes confuse users, and historically several different modes has been supported. Most modern applications use TrueColor (24-bit color, 8 bits for each of red, green and blue), but old or specialist applications may require a different colour mode. Many commercial specialist applications use PseudoColor.
- The X11 protocol actually uses a single 32-bit unsigned integer for representing a single colour in most graphic operations, called a *pixel value*. When transferring primary colors intensity, a 16 bits integer is used for each colour component. The following representations of colors exist; not all of them may be supported on a specific device.
  - ➢ **DirectColor:** A pixel value is decomposed into separate red, green, and blue subfields. Each subfield indexes a separate colormap. Entries in all colormaps can be changed.
  - ➢ **TrueColor:** Same as DirectColor, except that the colormap entries are predefined by the hardware and cannot be changed. Typically, each of the red, green, and blue colormaps provides a (near) linear ramp of intensity.
  - ➢ **GrayScale:** A pixel value indexes a single colormap that contains monochrome intensities. Colormap entries can be changed.
  - ➢ **StaticGray:** Same as GrayScale, except that the colormap entries are predefined by the hardware and cannot be changed.
  - ➢ **PseudoColor** (Chunky): A pixel value indexes a single colormap that contains colour intensities. Colormap entries can be changed.
  - ➢ **StaticColor:** Same as PseudoColor, except that the colormap entries are predefined by the hardware and cannot be changed.

4.2 INTRODUCTION TO OPENGL

- In recent years OpenGL has become a worldwide standard for 3D computer graphics programming. It's very widely used: in industry, in research laboratories, in computer games – and for teaching computer graphics.
- OpenGL is often called an "Application Programming Interface" (API): the interface is a collection of routines that the programmer can call, along with a model of how the routines work together to produce graphics. The programmer "sees" only the interface, and is therefore shielded from having to cope with the specific hardware or software idiosyncrasies on the resident graphics system.
- OpenGL is at its most powerful when drawing images of complex three dimensional (3D) scenes, as we shall see. It might be viewed as overkill for simple drawings of 2D objects. But it works well for 2D drawing, too, and affords a *unified* approach to producing pictures.

What is OpenGL?

- OpenGL has it origins in the earlier GL ("Graphics Library") system which was invented by Silicon Graphics Inc. as the means for programming their high-performance specialized graphics workstations. As time went on, people became interested in porting GL to other kinds of machine, and in1992 a variation of
- GL–called OpenGL–was announced. Unlike GL, OpenGL was specifically designed to be platform-independent, So it would work across a whole range of computer hardware – not just Silicon Graphics machines.
- The combination of OpenGL's power and portability led to its rapid acceptance as a standard for computer graphics programming. OpenGL itself isn't a programming language, or a software library. It's the specification of an Application Programming Interface (API) for computer graphics programming. In other words, OpenGL defines a set of functions for doing computer graphics.

4.2.1 OpenGL Architecture

- The Fig. 4.2 shown below gives an abstract, high-level block diagram of how OpenGL processes data. In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to

be drawn, and others control how the objects are handled during the various processing stages.

- As shown by the first block in the diagram, rather than having all commands proceeds immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time. The *evaluator* stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, *per-vertex operations and primitive assembly*; OpenGL processes geometric primitives- points, line segments and polygons all of which are described by vertices.
- Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.
- Rasterization produces a series of frame buffer addresses and associated values using a two dimensional description of a point, line segment or polygon. Each fragment so produced is fed into the last stage, per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer.

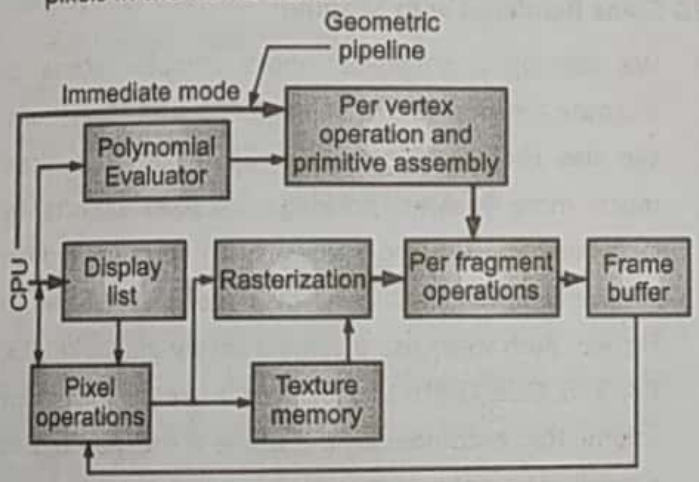


Fig. 4.2 : Architecture of OpenGL

- These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.
- Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The result of this stage is either stored as texture memory, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.
- All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

OpenGL Features

- It provides 3D geometric objects, such as lines, polygons, triangle meshes, spheres, cubes, quadric surfaces, NURBS curves and surfaces; It provides 3D modeling transformations, and viewing functions to create views of 3D scenes using the idea of a virtual camera;
- It supports high-quality rendering of scenes, including hidden-surface removal, multiple light sources, material types, transparency, textures, blending, fog;
- It provides display lists for creating graphics caches and hierarchical models. It also supports the interactive "picking" of objects;
- It supports the manipulation of images as pixels, enabling frame-buffer effects such as antialiasing, motion blur, depth of field and soft shadows.

OpenGL Drawing Primitives

OpenGL supports various basic primitives' types:

1. GL_POINTS - single points
2. GL_LINES - pairs of vertices are a line segment
3. GL_LINE_STRIP - polyline
4. GL_LINE_LOOP - closed polyline
5. GL_TRIANGLES - triples indicate a triangle
6. GL_TRIANGLE_STRIP- linked strip of triangles
7. GL_TRIANGLE_FAN - linked fan of triangles
8. GL_QUADS - quadruples indicate a quadrilateral
9. GL_QUAD_STRIP - linked strip of quadrilaterals
10. GL_POLYGON - simple convex polygon

Polygon Issues

OpenGL only correctly displays polygons that are

- **Simple:** edges cannot cross
- **Convex:** All points on line segment between two points in a polygon are also in the polygon
- **Flat:** all vertices are in the same plane
- Triangles satisfy all conditions

Attributes

Attributes are part of the OpenGL state and determine the appearance of objects.

- Color (points, lines, polygons)
- Size and width (points, lines)
- Stipple pattern (lines, polygons)
- Polygon mode(Display as filled: solid color or stipple pattern and Display edges).

OpenGL Library

OpenGL provides a simple collection of utilities, called the GL Utility Library or GLU in short.

- gl - basic OpenGL functions. #include <GL/gl.h>
- glu - OpenGL Utility Library. Encapsulates frequently used combinations of gl calls plus setup inside its functions. #include <GL/glu.h>
- glut - OpenGL Utility Toolkit. Encapsulates window management functions via a set of window-system independent functions. #include <GL/glut.h>
- glx - OpenGL Extension to the X Window System. Wrappers for X functions. #include <X11/Xlib.h> and #include <GL/glx.h>

OpenGL State

- OpenGL is a state machine.
- OpenGL functions are of two types
  1. Primitive generating: Can cause output if primitive is visible.
  2. How vertices are processed and appearances of primitive are controlled by the state.
- State changing
  1. Transformation functions
  2. Attribute functions

Typical GL Program Structure

- Configure and open a window.
- Initialize GL state .
- Register callback functions (Render , Resize, Events)
- Enter infinite event processing loop.

OpenGL as a Renderer

- Geometric primitives points, lines and polygons .
- Image Primitives images and bitmaps separate pipeline for images and geometry linked through texture mapping.
- Rendering depends on state colors, materials, light sources, etc.
- As mentioned, OpenGL is a library for rendering computer graphics. Generally, there are two operations that you do with OpenGL:
  1. Draw something
  2. Change the state of how OpenGL draws
- OpenGL has two types of things that it can render: geometric primitives and image primitives. *Geometric primitives* are points, lines and polygons. *Image primitives* are bitmaps and graphics images (i.e. the pixels that you might extract from a JPEG image after you've read it into your program.) Additionally, OpenGL links image and geometric primitives together using *texture mapping*, which is an advanced topic we'll discuss this afternoon. The other common operation that you do with OpenGL is *setting state*. "Setting state" is the process of initializing the internal data that OpenGL uses to render your primitives. It can be as simple as setting up the size of points and color that you want a vertex to be, to initializing multiple map levels for texture mapping.

3D Scene Rendered with Shading

- We develop a somewhat more complex scene to illustrate further the use of modeling transformations. We also show how easy OpenGL makes it to draw much more realistic drawings of solid objects by incorporating shading, along with proper hidden surface removal. Two views of a scene are shown in Fig. 4.3. Both views use a camera set by gluLookAt(2.3, 1.3, 2, 0, 0.25, 0, 0.0,1.0,0.0). Part a uses a large view volume that encompasses the whole scene; part b uses a small view volume that encompasses only a small portion of the scene, thereby providing a close-up view.
- The scene contains three objects resting on a table in the corner of a "room". Each of the three walls is made by flattening a cube into a thin sheet, and moving it into position. (Again, they look somewhat unnatural due to the use of a parallel projection.) The "jack" is composed of three stretched spheres oriented at right angles plus six small spheres at their ends.

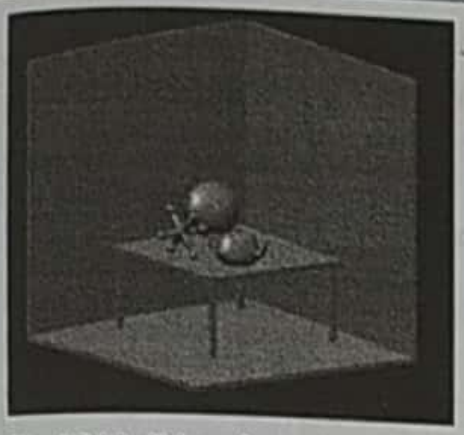


Fig. 4.3 (a) : Using a large view volume

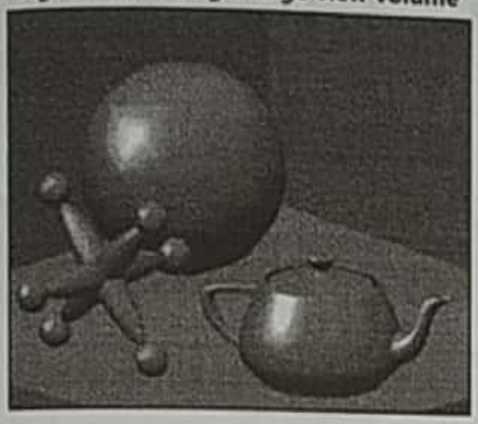


Fig. 4.3 (b) : Using a small view volume

4.2.2 GLUT Basics

Application Structure

- Configure and open window
- Initialize OpenGL State
- Register input callback functions
  - ➢ Render
  - ➢ Resize
  - ➢ Input: Keyboard, Mouse, etc
- Enter event processing loop

Here's the basic structure that we will be using in our applications. This is generally what you'd do in your own OpenGL applications. The steps are:

1. Choose the type of window that you need for your application and initialize it.
2. Initialize any OpenGL state that you don't need to change every frame of your program. This might include things like the background color, light positions and texture maps.
3. Register the callback functions that you'll need. Callbacks are routines you write that GLUT calls when a certain sequence of events occurs, like the window needing to be refreshed, or the user moving the mouse. The most important callback function is the one to render your scene, which we'll discuss in a few slides.
4. Enter the main event processing loop. This is where your application receives events, and schedules when callback functions are called.

Sample Program

```
void main( int argc, char** argv )
{
    int mode = GLUT_RGB|GLUT_DOUBLE;
    glutInitDisplayMode( mode );
    glutCreateWindow( argv[0] );
    init();
    glutDisplayFunc( display );
    glutReshapeFunc( resize );
    glutKeyboardFunc( key );
    glutIdleFunc( idle );
    glutMainLoop();
}
```

- Here is an example of the main part of a GLUT based OpenGL application. This is the model that we'll use for most of our programs in the course. The glutInitDisplayMode() and glutCreateWindow() functions compose the window configuration step. We then call the init() routine, which contains our one-time initialization. Here we initialize any OpenGL state and other program variables that we might need to use during our program that remain constant throughout the program's execution. Next, we register the callback routines that we're going to use during our program. Finally, we enter the event processing loop, which interprets events and calls our respective callback routines.

GLUT Callback Functions

Routine to call when something happens

- Window resize or redraw
- User input
- Animation

"Register" callbacks with GLUT

```
glutDisplayFunc( display );
glutIdleFunc( idle );
glutKeyboardFunc( keyboard );
```

GLUT uses a callback mechanism to do its event processing. Callbacks simplify event processing for the

application developer. As compared to more traditional event driven programming, where the author must receive and process each event, and call whatever actions are necessary, callbacks simplify the process by defining what actions are supported, and automatically handling the user events. All the author must do is fill in what should happen when. GLUT supports many different callback actions, including:

- glutDisplayFunc() - called when pixels in the window need to be refreshed.
- glutReshapeFunc() - called when the window changes size.
- glutKeyboardFunc() - called when a key is struck on the keyboard.
- glutMouseFunc() - called when the user presses a mouse button on the mouse.
- glutMotionFunc() - called when the user moves the mouse while a mouse button is pressed.
- glutPassiveMouseFunc() - called when the mouse is moved regardless of mouse button state.
- glutIdleFunc() - a callback function called when nothing else is going on. Very useful for animations.

GLU, GLUT Package in OpenGL

- An application programmer sees OpenGL as a single library providing a set of functions for graphical input and output. In fact, it's slightly more complicated than that. He support libraries: GLU and GLUT a key feature of the design of OpenGL function.
- GLUT provides the facilities for interaction that OpenGL lacks. It provides functions for managing windows on the display screen, and handling input events from the mouse and keyboard. It provides some rudimentary tools for creating Graphical User Interfaces (GUIs).
- It also includes functions for conveniently drawing 3D objects like the platonic solids, and a teapot. All GLUT function names start with "glut". Fig. 4.4 shows the relationships between OpenGL, GLU, and GLUT. As we can see it is helpful to think of "layers" of software where each layer calls upon the facilities of software in a lower layer.

The toolkit supports:

- Multiple windows for OpenGL rendering.
- Callback driven event processing.
- Sophisticated input devices.
- An 'idle' routine and timers.
- A simple, cascading pop-up menu facility.
- Utility routines to generate various solid and wire frame objects.
- Support for bitmap and stroke fonts.

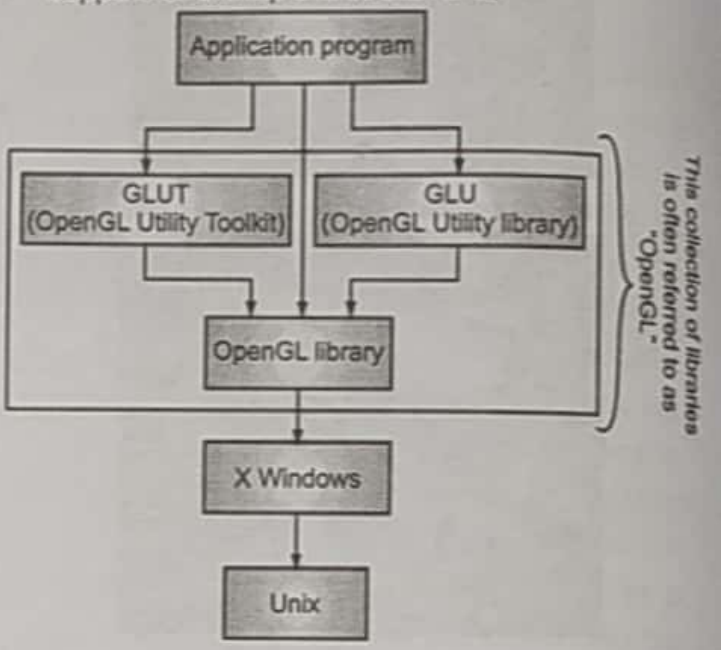


Fig. 4.4: What is commonly called "OpenGL" is actually a set of three libraries: OpenGL itself, and the supporting libraries GLU and GLUT

Write a program to draw a rectangle in OpenGL.

```
* simple.c second version*/
/* This program draws a white rectangle on a
blackbackground.*/
#include <glut.h> /* glut.h includes gl.h andglu.h*/
voiddisplay()
{
/* clear window*/
glClear(GL_COLOR_BUFFER_BIT);
/* draw unit square polygon*/
glBegin(GL_POLYGON);
glVertex2f(-0.5,-0.5);
glVertex2f(-0.5,0.5);
glVertex2f(0.5,0.5);
glVertex2f(0.5,-0.5);
glEnd();
/* flush GL buffers*/
glFlush();
}
void init()     // initializecolors
```

```
{
/* set clear color to black*/
glClearColor(0.0, 0.0, 0.0,0.0);
/* set fill color to white*/
glColor3f(1.0, 1.0,1.0);
}
void main(int argc, char**argv)
{
/* Initialize mode and open a window in upper left cornerof
/* screen*/
/* Window title is name of program (arg[0])*/
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE |GLUT_RGB);
glutInitWindowSize(500, 500); // Set windowSize
glutInitWindowPosition(0, 0);
                                    //Set WindowPosition
glutCreateWindow("simple");
                                //Create Window and Settitle
glutDisplayFunc(display);
                                //Call the Displayingfunction
init();                     //Initialize DrawingColors
glutMainLoop();
                    //Keep displaying until program isclosed.
}
```

Event-Driven Programming

- Another property of most windows-based programs is that they are event-driven. This means that the program responds to various events, such as a mouse click, the press of a keyboard key, or the resizing of a screen window. The system automatically manages an event queue, which receives messages that certain events have occurred, and deals with them on a first-come first-served basis.
- The programmer organizes a program as a collection of callback functions that are executed when events occur. A callback function is created for each type of event that might occur. When the system removes an event from the queue it simply executes the callback function associated with the type of that event. For programmers used to building programs with a "do this, then do this" structure some rethinking is required. The new structure is more like: "do nothing until an event occurs, then do the specified thing".
- The method of associating a callback function with an event type is often quite system dependent. But OpenGL comes with a Utility Toolkit (see Appendix 1), which provides tools to assist with event management. For instance glutMouseFunc(myMouse); // register the mouse action function registers the function myMouse() as the function to be executed when a mouse event occurs. The prefix "glut" indicates it is part of the OpenGL Utility Toolkit. The programmer puts code in myMouse() to handle all of the possible mouse actions of interest.

```
All of the callback functions are defined here
void main()
{
initialize things 5 create a screen window
glutDisplayFunc(myDisplay);
 //register the redraw function
glutReshapeFunc(myReshape);
 // register the reshape function
glutMouseFunc(myMouse);
 // register the mouse action function
glutKeyboardFunc(myKeyboard);
// register the keyboard action function perhaps
initialize other things glutMainLoop();
// enter the unending main loop
 }
```

4.2.3 Simple Interaction with the Mouse and Keyboard

- Interactive graphics applications let the user control the flow of a program by natural human motions: pointing and clicking the mouse, and pressing various keyboard keys. The mouse position at the time of the click, or the identity of the key pressed, is made available to the application program and is processed as appropriate.
- Recall that when the user presses or releases a mouse button, moves the mouse, or presses a keyboard key, an event occur. Using the OpenGL Utility Toolkit (GLUT) the programmer can register a callback function with each of these events by using the following commands:

- glutMouseFunc(myMouse) which registers myMouse() with the event that occurs when the mouse button is pressed or released;
- glutMotionFunc(myMovedMouse) which registers myMovedMouse() with the event that occurs when the mouse is moved while one of the buttons is pressed;
- glutKeyboardFunc(myKeyboard) which registers myKeyBoard() with the event that occurs when a keyboard key is pressed.

We next see how to use each of these.

Mouse Interaction

- How is data about the mouse sent to the application? You must design the callback function myMouse() to take four parameters, so that it has the prototype: void myMouse(int button, int state, int x, int y); When a mouse event occurs the system calls the registered function, supplying it with values for these parameters.
- The value of button will be one of: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON, with the obvious interpretation, and the value of state will be one of: GLUT_UP or GLUT_DOWN. The values x and y report the position of the mouse at the time of the event. Alert: The x value is the number of pixels from the left of the window as expected, but the y value is the number of pixels down from the top of the window!

Example: Placing Dots with the Mouse

- We start with an elementary but important example. Each time the user presses down the left mouse button a dot is drawn in the screen window at the mouse position. If the user presses the right button the program terminates. The version of myMouse() shown next does the job. Because the y-value of the mouse position is the number of pixels from the top of the screen window, we draw the dot, not at (x, y), but at (x, screenHeight – y), where screenHeight is assumed here to be the height of the window in pixels.

```
void myMouse(int button, int state, int x, int y)
{
if(button == GLUT_LEFT_BUTTON && state ==
GLUT_DOWN)
 drawDot(x, screenHeight -y);
 else if(button == GLUT_RIGHT_BUTTON && state
 == GLUT_DOWN) exit(-1);
}
```

The argument of –1 in the standard function exit() simply returns –1 back to the operating system: It is usually ignored.

Keyboard Interaction

- As mentioned earlier, pressing a key on the keyboard queues a keyboard event. The callback function myKeyboard() is registered with this type of event through glutKeyboardFunc(myKeyboard). It must have prototype: void myKeyboard(unsigned int key, int x, int y);
- The value of key is the ASCII value 12 of the key pressed. The values x and y report the position of the mouse at the time that the event occurred. (As before y measures the number of pixels down from the top of the window.) The programmer can capitalize on the many keys on the keyboard to offer the user a large number of choices to invoke at any point in a program.
- Most implementations of myKeyboard() consist of a large switch statement, with a case for each key of interest. Fig. 4.4 shows one possibility. Pressing 'p' draws a dot at the mouse position; pressing the left arrow key adds a point to some (global) list, but does no drawing 13; pressing 'E' exits from the program. Note that if the user holds down the 'p' key and moves the mouse around a rapid sequence of points is generated to make a "freehand" drawing.

An Example of the Keyboard Callback Function

```
void myKeyboard(unsigned char theKey, int mouseX, int
mouseY)
{
GLint x = mouseX;
 GLint y = screenHeight - mouseY;
// flip the y value as always switch(theKey)
{
case 'p': drawDot(x, y);
 // draw a dot at the mouse position break;
 case GLUT_KEY_LEFT: List[++last].x = x;
 // add a point List[ last].y = y; break;
 case 'E': exit(-1);
 //terminate the program default: break;
// do nothing
 }
 }
```

4.3 DIRECTX

- Microsoft DirectX is a collection of Application Programming Interfaces (APIs) for handling tasks related to multimedia, especially game programming and video, on Microsoft platforms. Originally, the names of these APIs all began with "Direct", such as Direct3D, DirectDraw, DirectMusic, DirectPlay, DirectSound, and so forth. The name DirectX was coined as a shorthand term for all of these APIs (the X standing in for the particular API names) and soon became the name of the collection. When Microsoft later set out to develop a gaming console, the X was used as the basis of the name Xbox to indicate that the console was based on DirectX technology. The X initial has been carried forward in the naming of APIs designed for the Xbox such as XInput and the Cross-platform Audio Creation Tool (XACT), while the DirectX pattern has been continued for Windows APIs such as Direct2D and Direct Write.
- Direct3D (the 3D graphics API within DirectX) is widely used in the development of video games for Microsoft Windows and the Xbox line of consoles. Direct3D is also used by other software applications for visualization and graphics tasks such as CAD/CAM engineering. As Direct3D is the most widely publicized component of DirectX, it is common to see the names "DirectX" and "Direct3D" used interchangeably.
- The DirectX Software Development Kit (SDK) consists of runtime libraries in redistributable binary form, along with accompanying documentation and headers for use in coding. Originally, the runtimes were only installed by games or explicitly by the user. Windows 95 did not launch with DirectX, but DirectX was included with Windows 95 OEM Service Release 2. Windows 98 and Windows NT 4.0 both shipped with DirectX, as has every version of Windows released since. The SDK is available as a free download. While the runtimes are proprietary, closed-source software, source code is provided for most of the SDK samples. Starting with the release of Windows 8 Developer Preview, DirectX SDK has been integrated into Windows SDK.

4.3.1 Versions

DirectX 9

- DirectX 9 was released in 2002 for Windows 98 and XP, and currently is supported by all subsequent versions. Microsoft continues to make changes in DirectX 9.0c, causing support to be dropped for some of the aforementioned operating systems. As of January 2007, Windows 2000 or XP is required. This also introduced Shader Model 2.0 containing Pixel Shader 2.0 and Vertex Shader 2.0. Windows XP SP2 and newer include DirectX 9.0c, but may require a newer DirectX runtime redistributable installation for DirectX 9.0c applications compiled with the February 2005 DirectX 9.0 SDK or newer.

DirectX 10

- A major update to DirectX API, DirectX 10 ships with and is only available with Windows Vista and later; previous versions of Windows such as Windows XP are not able to run DirectX 10-exclusive applications. Rather, programs that are run on a Windows XP system with DirectX 10 hardware simply resort to the DirectX 9.0c code path, the latest available for Windows XP computers.
- Changes for DirectX 10 were extensive. Many former parts of DirectX API were deprecated in the latest DirectX SDK and are preserved for compatibility only: DirectInput was deprecated in favor of XInput, DirectSound was deprecated in favor of the Cross-platform Audio Creation Tool system (XACT) and additionally lost support for hardware accelerated audio, since the Vista audio stack renders sound in software on the CPU. The DirectPlay DPLAY.DLL was also removed and was replaced with dplayx.dll; games that rely on this DLL must duplicate it and rename it to dplay.dll.
- In order to achieve backwards compatibility, DirectX in Windows Vista contains several versions of Direct3D:
  - ➢ **Direct3D 9:** emulates Direct3D 9 behavior as it was on Windows XP. Details and advantages of Vista's Windows Display Driver Model are hidden from the application if WDDM drivers are installed. This is the only API available if there are only XP graphic drivers (XDDM) installed, after an upgrade to Vista for example.

- Direct3D 9Ex (known internally during Windows Vista development as 9.0L or 9.L): allows full access to the new capabilities of WDDM (if WDDM drivers are installed) while maintaining compatibility for existing Direct3D applications. The Windows Aero user interface relies on D3D 9Ex.
- **Direct3D 10:** Designed around the new driver model in Windows Vista and featuring a number of improvements to rendering capabilities and flexibility, including Shader Model 4.
- Direct3D 10.1 is an incremental update of Direct3D 10.0 which shipped with, and required, Windows Vista Service Pack 1. This release mainly sets a few more image quality standards for graphics vendors, while giving developers more control over image quality. It also adds support for cube map arrays, separate blend modes per-MRT, coverage mask export from a pixel shader, ability to run pixel shader per sample, access to multi-sampled depth buffers and requires that the video card supports Shader Model 4.1 or higher and 32-bit floating-point operations. Direct3D 10.1 still fully supports Direct3D 10 hardware, but in order to utilize all of the new features, updated hardware is required.

4.4 WINDOWS AND MOTIF

4.4.1 Motif 2.1

- Motif is the industry standard graphical user interface, (as defined by the IEEE 1295 specification), used on more than 200 hardware and software platforms. It provides application developers, end users, and system vendors with the industry's most widely used environment for standardizing application presentation on a wide range of platforms. Motif is the leading user interface for the UNIX operating system.
- The Motif Graphical User Interface (GUI) toolkit facilitates the development of applications for heterogeneous, networked computing environments. By providing application portability across a variety of platforms, the Motif environment helps protect valuable investments in software and user training.
- Users of laptops, PCs, workstations, mainframes, and supercomputers benefit from the consistent screen appearance and behavior of applications provided by the Motif environment. Motif was the first graphical interface offering user-oriented PC-style behavior and screen appearance for applications running on systems that support the X Window System X11R5.
- Motif is also the base graphical user interface toolkit for the Common Desktop Environment (CDE). CDE, originally developed by Hewlett-Packard, IBM, Novell, and SunSoft, provides a single, standard graphical desktop and desktop tool set (such as mail and a group calendar) for all platforms that support the X Window System.

4.4.2 Motif and CDE Source Code Synchronization

- Motif Release 2.1 provides complete compatibility and convergence with CDE by integrating the critical features and functionality of Motif 2.0 and CDE 2.1

Motif 2.1 New Features

- Thread-safe libraries.
- Widget printing support.
- A merged Motif and CDE style guide.
- Internationalization enhancements for vertical text, "on-the-spot" input and user defined characters for Asian languages.
- As significant new features were added to Motif 2.1, some features of Motif 2.0 needed to be removed to bring the source code streams for Motif and CDE into synchronization. Items removed include .
  - Platform-independent uid files .
  - CSText widget .
  - Support for C++.
- The Motif 2.1 Window Manager (mwm) is a "lite" version of the CDE 2.1 Window Manager, desktop window manager (dtwm 2.1), and provides functionality compatible with Motif 1.2 mwm. Users of Motif 2.0 can continue to use 2.0 mwm without experiencing compatibility problems between Motif 2.0 and 2.1 releases.

Motif 2.1 Benefits

- Motif provides application developers, end users, independent soft ware vendors, and system vendors with a high degree of portability, interoperability, and scalability for their applications.

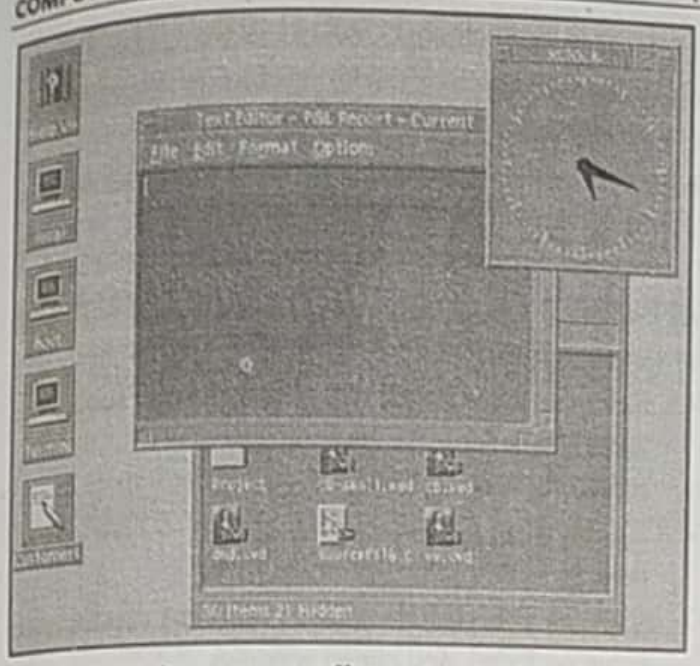


Fig. 4.5 : window manager offers a standard interface for moving, resizing or iconifying application windows

End Users

- Motif gives end users a way to leverage their investments in existing systems, applications, and training. Specifically, it offers end users the ability to select interfaces compatible with the ones they have, and to specify one standard interface for the future. Consistent from laptop to mainframe, the Motif style is similar to Microsoft Windows, providing added features familiar to open systems users. These characteristics help reduce training time and costs by easing skills transfer across heterogeneous systems.

Independent Software Vendors

- Using Motif, independent software vendors and other application developers can port applications across a variety of single and multi-user hardware platforms and build custom widgets with ease. Applications built with Motif's single, stable application programming interface have excellent performance characteristics.

System Vendors

- Motif functions on more than 200 different platforms with a single GUI toolkit. As a result, hardware vendors can unify the appearance and behavior of applications on all the platforms they supply.

The Elements of Motif

- The core components of the Motif technology include an extensible user interface toolkit; a stable application programming interface; a user interface language; and a window manager.

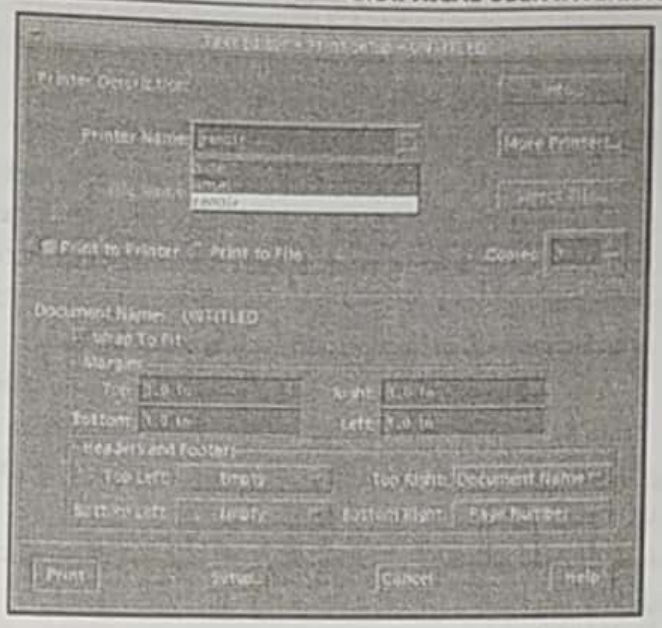


Fig. 4.6 : The convergence of the Motif and CDE style guides includes support for all new widgets, such as this combo box and spin box.

- Taken together, these technology components make Motif the graphical user interface of choice for system vendors, application developers and end users.

Application Programming Interface

- The Motif Application Programming Interface (API) specifies the interface to the User Interface Toolkit and the Motif Resource Manager. The API's behavior and appearance are compatible with that of Microsoft Windows, and with enhancements such as primary transfer and implicit focus, familiar to most workstation users.

Thread-Safe Libraries

- All necessary libraries in Motif 2.1 are thread-safe. Multi-threaded applications can use Motif without requiring that the programmer explicitly control "locking" of a library's routines or data or limiting the use of a library API to a single thread.

User Interface Toolkit

- The User Interface Toolkit provides a standard graphical user interface layer upon which applications are based. It includes a library of graphical objects used in the construction of application user interfaces such as menus and scroll bars.

Toolkit Intrinsics

- The Motif toolkit is based on the X11 Intrinsics, a toolkit framework pro vided with the X Window System. The Intrinsics have been specified as a U.S.

federal procurement standard and provide compliance with X-based applications and systems.

- The Intrinsics use an object-oriented model to create a class hierarchy of graphical objects known as widgets. The functionality of the Motif toolkit is further extended by windowless widgets called gadgets, which can be cached, performing specific functions with lower server overhead.
- Motif widgets have associated sets of resources that can be specific to a class of widgets or inherited from a super class of widgets. Resource values can be specified by the application program, read from a database through the User Interface Language, or defined by the user.

Motif Widgets

- The Motif widget set is a rich collection of user interface controls de signed to support a friendly user metaphor. Motif 2.0 and Motif 2.1 provide users with more flexibility in selecting visual styles as well as better looking user interfaces. New widgets introduced with Release 2.0 included.
  - ➢ Container/Icon Gadget that is used to provide both tabular and iconic presentations of a hierarchy of objects that can be used to build file/directory managers.
  - ➢ Notebook used to create property sheets or multi-page documents with tabs.
  - ➢ Spin box used for cycling through sets of choices such as setting dates and times.

Widget Printing

- Developers now have a standard way of printing from any application or utility. In Motif 2.1, an X Server receives the normal X network protocol from the source, and with the help of a new X extension for print management, can generate output for a printer rather than for a screen display. Use of this feature requires a print server based on X11 Release 6.2.

Development Features

A number of features simplify development of user interfaces with the Motif style. These include

- Dozens of basic objects, dialogue boxes, layout management elements, menus, and special-purpose widgets.
- Virtual key bindings, which enable applications to behave consistently with keyboards from different vendors.
- Keyboard traversal, making information input possible from either a key board or a mouse, providing compatibility with PC-style interaction.
- Facilities for scrolling, which are helpful for small screens.
- Functions that provide direct programmer support for focus management, allowing specification of input focus.
- Functions and call-backs to support context-specific help.
- Sophisticated layout-management tools such as rows, columns, forms, and paned windows.
- A comprehensive menu system, including tear-off menus .
- Immediate access to the file system through a file selection box.

Extensibility Framework

- Motif allows developers to build new widgets easily through the use of subclassing and traits.

Subclassing

- As with all Xt-based toolkits, subclassing requires detailed knowledge, experience, and access to the source code. Motif simplifies this process by allowing subclassing from the Primitive and Manager classes to be accomplished easily, without requiring access to source code. Documentation of Motif's class methods are included in *The Motif Widget Writer's Guide*. This book provides all necessary information to subclass from Primitive and Manager. Motif fully documents Xme (Xm extensibility) functions.

Traits

- Traits allow a given behavior to be associated to a widget irrespective of the widget hierarchical relationships. Before the availability of traits, many widgets checked whether their parent or child was a member of a particular class and changed their behavior accordingly.
- However, a developer building a new and related group of primitive widgets would rather define a new subclass of Primitive, and subsequently subclass all the new or custom widgets. Motif 2.1 allows any number

of traits to be associated with a class. Each trait has an associated group of class methods that a developer can use to implement the behavior promised by the trait. The standard traits and their class methods are fully documented in the widget writer's guide.

User Interface Language

- The User Interface Language is an application development tool that sups ports rapid user interface design and prototyping. It allows application developers to create a text file that contains a description of each widget and its resources. Interface designers with little programming experience will find the Motif User Interface Language easy to use.
- This high-level description is compiled into a resource file used by the application and loaded automatically at runtime. The User Interface Language allows an application designer to define and tune the presentation characteristics of an application interface independent of the application code, simplifying the description and maintenance of user interfaces.

The User Interface Language Offers

- Descriptions of the objects used in the user interface and their layout constraints.
- Support for every object in the Motif toolkit and for extensions to the toolkit through a widget meta language.
- Modularity to support applications that use multiple UIL files.
- Support for 64-bit platforms.

4.5 GRAPHICS STANDARDS

- Agreed specifications which define the common interfaces between computer systems or subsystems. Standards, if they are generally observed by manufacturers, promote the interchangeability of computer equipment. When standards promote the portability of programs, this is called device independence.
- In computer graphics, standards have evolved slowly and painfully. Most of them have at one time needed to have the qualifying adjective "de facto" or "proposed" attached to them, indicating that they are not absolute "standards" in the full sense of the word. However, the standards that now exist, or are in the course of being ratified at a national or international level, are helping to rationalize the interfacing of graphics systems.
- The ISO (International Standards Organization) Basic Reference Model of computer architecture defines a system in terms of layers. Thus, the best way to consider current graphics standards is to categorize them according to their individual levels in the layered architecture of a computer system.

Overview of Various Graphics Standards

A Reference Model

- Functional standards specify a model of a system, a set of operations on the model, and the *externally visible* effects of these operations. Such standards do not specify how these effects are to be implemented. In short, standards codify the exchange of information across an interface between two functional units and specify wftnl is to be exchanged, but not *how* the functional units carry out their operations.

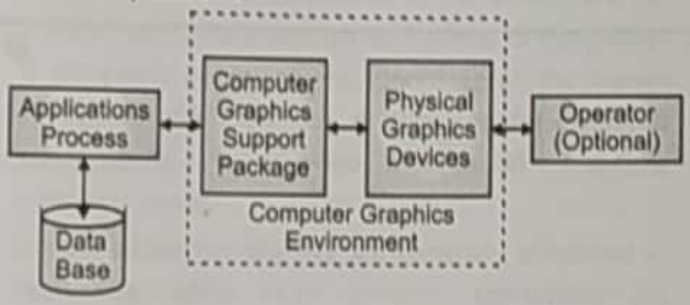


Fig. 4.7 : Major Components of Graphics Support Environment

- Fig. 4.7 is a very simple model of a computer graphics operating environment. The model emphasizes that a graphics application process interacts with physical graphics devices and human operators via a computer graphics environment. It also shows that the application may get information from an external product definition data base. The model is extremely schematic and, with the possible exception of using the term "database," is a schematic representation of the use of graphics systems over the past 20 years. At the time when the work on graphics standards got underway (c. 1975) this model might have been associated with the arrangement in Fig. 4.8. This shows the applications package communicating through a graphics package (usually at that time a library of FORTRAN routines) and via the operating system on the host computer over a serial line to a remote terminal. The graphics package was typically divided

into a device independent "front-end" and device dependent "back-end" or device driver. Depending on the affluence of the system's owner the terminal might have had some local intelligence, although the majority would have been unintelligent devices.

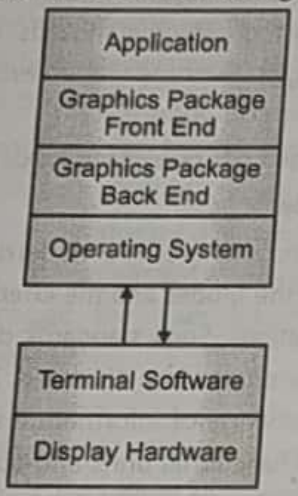


Fig. 4.8: Typical Arrangement c.1975

EXERCISE

1. Explain Client Server model in X with neat diagram.
2. Draw and explain X Window System protocols and architecture
3. Explain design principles of X.
4. Describe X window core protocol.
5. What is OpenGL? Explain OpenGL architecture with neat diagram.
6. Explain GLU, GLUT Package in OpenGL.
7. Write a program to draw a rectangle inOpenGL.
8. Explain mouse and keyboard interaction in OpenGL.
9. Enlist and explain versions of DirectX.
10. Explain Motif and CDE Source Code Synchronization.
11. Explain with neat diagram reference model of graphics standards.
12. Enlist and explain development features of Motif.

✠ ✠ ✠

UNIT V

5.1 ANIMATION

Introduction to Animation:

- Computer-graphics methods are now commonly used to produce animations for a variety of applications, including entertainment (motion pictures and cartoons), advertising, scientific and engineering studies, and training and education. Although we tend to think of animation as implying object motion, the term **computer animation** generally refers to any time sequence of visual changes in a picture.
- In addition to changing object positions using translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture. Advertising animations often transition one object shape into another: for example, transforming a can of motor oil into an automobile engine.
- We can also generate computer animations by varying camera parameters, such as position, orientation, or focal length, and variations in lighting effects or other parameters and procedures associated with illumination and rendering can be used to produce computer animations. Another consideration in computer-generated animation is realism. Many applications require realistic displays. An accurate representation of the shape of a thunderstorm or other natural phenomena described with a numerical model is important for evaluating the reliability of the model. Similarly, simulators for training aircraft pilots and heavy-equipment operators must produce reasonably accurate representations of the environment. Entertainment and advertising applications, on the other hand, are sometimes more interested in visual effects. Thus, scenes may be displayed with exaggerated shapes and unrealistic motions and transformations.
- However, there are many entertainment and advertising applications that do require accurate representations for computer-generated scenes. Also, in some scientific and engineering studies, realism is not a goal. For example, physical quantities are often displayed with pseudo-colors or abstract shapes that change over time to help the researcher understand the nature of the physical process.
- Two basic methods for constructing a motion sequence are **real-time animation** and **frame-by-frame animation.** In a real-time computer-animation, each stage of the sequence is viewed as it is created. Thus the animation must be generated at a rate that is compatible with the constraints of the refresh rate. For a frame-by-frame animation, each frame of the motion is separately generated and stored. Later, the frames can be recorded on film, or they can be displayed consecutively on a video monitor in "real-time playback" mode.
- Simple animation displays are generally produced in real time, while more complex animations are constructed more slowly, frame by frame. However, some applications require real-time animation, regardless of the complexity of the animation. A flight-simulator animation, for example, is produced in real time because the video displays must be generated in immediate response to changes in the control settings. In such cases, special hardware and software systems are often developed to allow the complex display sequences to be developed quickly.

Raster Methods for Computer Animation

- Most of the time, we can create simple animation sequences in our programs using real-time methods. In general, though, we can produce an animation sequence on a raster-scan system one frame at a time, so that each completed frame could be saved in a file for later viewing.
- The animation can then be viewed by cycling through the completed frame sequence, or the frames could be transferred to film. If we want to generate an

animation in real time, however, we need to produce the motion frames quickly enough so that a continuous motion sequence is displayed.

- For a complex scene, one frame of the animation could take most of the refresh cycle time to construct. In that case, objects generated first would be displayed for most of the frame refresh time, but objects generated toward the end of the refresh cycle would disappear almost as soon as they were displayed. For very complex animations, the frame construction time could be greater than the time to refresh the screen, which can lead to erratic motion and fractured frame displays.
- Because the screen display is generated from successively modified pixel values in the refresh buffer, we can take advantage of some of the characteristics of the raster screen-refresh process to produce motion sequences quickly.

1. Double Buffering

- One method for producing a real-time animation with a raster system is to employ two refresh buffers. Initially, we create a frame for the animation in one of the buffers. Then, while the screen is being refreshed from that buffer, we construct the next frame in the other buffer. When that frame is complete, we switch the roles of the two buffers so that the refresh routines use the second buffer during the process of creating the next frame in the first buffer.
- This alternating buffer process continues throughout the animation. Graphics libraries that permit such operations typically have one function for activating the double buffering routines and another function for interchanging the roles of the two buffers.
- When a call is made to switch two refresh buffers, the interchange could be performed at various times. The most straight forward implementation is to switch the two buffers at the end of the current refresh cycle, during the vertical retrace of the electron beam. If a program can complete the construction of a frame within the time of a refresh cycle, say 1/60 of a second, each motion sequence is displayed in synchronization with the screen refresh rate.
- However, if the time to construct a frame is longer than the refresh time, the current frame is displayed for two or more refresh cycles while the next animation frame is being generated. For example, if the screen refresh rate is 60 frames per second and it takes 1/50 of a second to construct an animation frame, each frame is displayed on the screen twice and the animation rate is only 30 frames each second. Similarly, if the frame construction time is 1/25 of a second, the animation frame rate is reduced to 20 frames per second because each frame is displayed three times.
- Irregular animation frame rates can occur with double buffering when the frame construction time is very nearly equal to an integer multiple of the screen refresh time. As an example of this, if the screen refresh rate is 60 frames per second, then an erratic animation frame rate is possible when the frame construction time is very close to 1/60 of a second, or 2/60 of a second, or 3/60 of a second, and so forth. Because of slight variations in the implementation time for the routines that generate the primitives and their attributes, some frames could take a little more time to construct and some a little less time.
- Thus, the animation frame rate can change abruptly and erratically. One way to compensate for this effect is to add a small time delay to the program. Another possibility is to alter the motion or scene description to shorten the frame construction time.

2. Generating Animations Using Raster Operations

- We can also generate real-time raster animations for limited applications using block transfers of a rectangular array of pixel values. This animation technique is often used in game-playing programs. A simple method for translating an object from one location to another in the *xy* plane is to transfer the group of pixel values that define the shape of the object to the new location.
- Two-dimensional rotations in multiples of 90° are also simple to perform, although we can rotate rectangular blocks of pixels through other angles using antialiasing procedures. For a rotation that is not a multiple of 90°, we need to estimate the percentage of area coverage for those pixels that overlap the rotated block.
- Sequences of raster operations can be executed to produce real time animation for either two-dimensional or three-dimensional objects, so long as we restrict the animation to motions in the projection plane. Then no viewing or visible-surface algorithms need be invoked.

- We can also animate objects along two-dimensional motion paths using **color table transformations.** Here we predefine the object at successive positions along the motion path and set the successive blocks of pixel values to color-table entries. The pixels at the first position of the object are set to a foreground color, and the pixels at the other object positions are set to the background color.
- The animation is then accomplished by changing the color-table values so that the object color at successive positions along the animation path becomes the foreground color as the preceding position is set to the background color (Fig. 5.1).

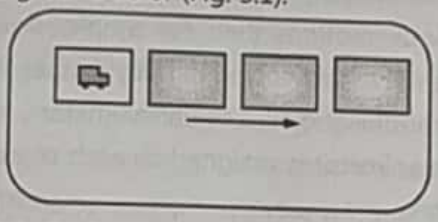


Fig. 5.1 : Real-time raster color-table animation

5.1.1 Conventional and Computer Based Animation

Traditional Animation Techniques

- Film animators use a variety of methods for depicting and emphasizing motion sequences. These include object deformations, spacing between animation frames, motion anticipation and follow-through, and action focusing.
- One of the most important techniques for simulating acceleration effects, particularly for non rigid objects, is **squash and stretch.** Fig. 5.2 shows how this technique is used to emphasize the acceleration and deceleration of a bouncing ball. As the ball accelerates, it begins to stretch. When the ball hits the floor and stops, it is first compressed (squashed) and then stretched again as it accelerates and bounces upwards.
- Another technique used by film animators is **timing,** which refers to the spacing between motion frames. A slower moving object is represented with more closely spaced frames, and a faster moving object is displayed with fewer frames over the path of the motion. This effect is illustrated in Fig. 5.3, where the position changes between frames increase as a bouncing ball moves faster.
- Object movements can also be emphasized by creating preliminary actions that indicate an **anticipation** of a coming motion. For example, a cartoon character might lean forward and rotate its body before starting to run; or a character might perform a "windup" before throwing a ball. Similarly, **follow-through actions** can be used to emphasize a previous motion.
- After throwing a ball, a character can continue the arm swing back to its body; or a hat can fly off a character that is stopped abruptly. An action also can be emphasized with **staging,** which refers to any method for focusing on an important part of a scene, such as a character hiding something.

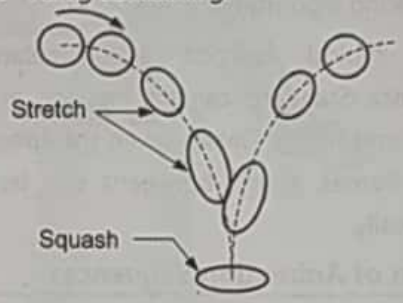


Fig. 5.2 : A bouncing-ball illustration of the "squash and stretch" technique for emphasizing object acceleration

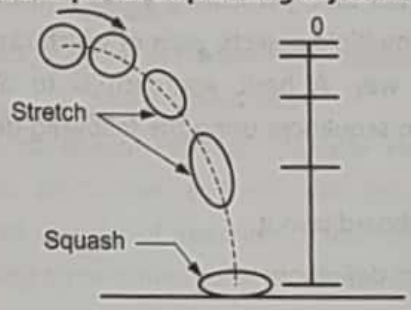


Fig. 5.3 : The position changes between motion frames for a bouncing ball increase as the speed of the ball increases

5.1.2 General Computer-Animation Functions

- Many software packages have been developed either for general animation design or for performing specialized animation tasks. Typical animation functions include managing object motions, generating views of objects, producing camera motions, and the generation of in-between frames. Some animation packages, such as Wave front for example, provide special functions for both the overall animation design and the processing of individual objects. Others are special-purpose packages for particular features of an animation, such as a system for generating in-between frames or a system for figure animation.

- A set of routines is often provided in a general animation package for storing and managing the object database. Object shapes and associated parameters are stored and updated in the database. Other object functions include those for generating the object motion and those for rendering the object surfaces. Movements can be generated according to specified constraints using two dimensional or three-dimensional transformations. Standard functions can then be applied to identify visible surfaces and apply the rendering algorithms.
- Another typical function set simulates camera movements. Standard camera motions are zooming, panning, and tilting. Finally, given the specification for the key frames, the in-betweens can be generated automatically.

5.1.3 Design of Animation Sequences

- Constructing an animation sequence can be a complicated task, particularly when it involves a story line and multiple objects, each of which can move in a different way. A basic approach is to design such animation sequences using the following development stages:
  1. Storyboard layout
  2. Object definitions
  3. Key-frame specifications
  4. Generation of in-between frames

1. Storyboard Layout

- The **storyboard** is an outline of the action. It defines the motion sequence as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches, along with a brief description of the movements, or it could just be a list of the basic ideas for the action. Originally, the set of motion sketches was attached to a large board that was used to present an overall view of the animation project. Hence, the name "storyboard."

2. Object Definitions

- An **object definition** is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygons or spline surfaces. In addition, a description is often given of the movements that are to be performed by each character or object in the story.

3. Key-Frame Specifications

- A **key frame** is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object (or character) is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great. More key frames are specified for intricate motions than for simple, slowly varying motions. Development of the key frames is generally the responsibility of the senior animators, and often a separate animator is assigned to each character in the animation.

4. Generation of In-Between Frames

- **In-betweens** are the intermediate frames between the key frames. The total number of frames, and hence the total number of in-betweens, needed for an animation is determined by the display media that is to be used. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 60 or more frames per second.
- Typically, time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames could be duplicated. As an example, a 1-minute film sequence with no duplication requires a total of 1,440 frames. If five in-betweens are required for each pair of key frames, then 288 key frames would need to be developed.
- There are several other tasks that may be required, depending on the application. These additional tasks include motion verification, editing, and the production and synchronization of a soundtrack. Many of the functions needed to produce general animations are now computer-generated. Fig. 5.4 (a) and Fig. 5.4 (b) show examples of computer-generated frames for animation sequences.

(a) (b)

One frame from the award-winning computer-animated short fil, Luxo Jr. The film was designed using a key-frame animation system and cartoon animation techniques to provide lifelike actions of the lamps. Final images were rendered with multiple light sources and procedural texturing techniques. (Courtesy of Pixer. © 1986 Pixer.)

One frame from the short film Tin Toy, the first computer-animated film to win an Oscar. Designed using a key-frame animation system, the film also required extensive facial-expression modelling. Final images were rendered using procedural shading, self-shadowing techniques, motion blur, and texture mapping. (Courtesy of Pixer. © 1988 Pixer.)

Fig. 5.4

5.1.4 Animation Languages (Computer)

- We can develop routines to design and control animation sequences within a general-purpose programming language, such as C, C++, Lisp, or Fortran, but several specialized animation languages have been developed.
- These languages typically include a graphics editor, a key-frame generator, an in-between generator, and standard graphics routines. The graphics editor allows an animator to design and modify object shapes, using spline surfaces, constructive solid geometry methods, or other representation schemes.
- An important task in an animation specification is *scene description*. This includes the positioning of objects and light sources, defining the photometric parameters (light-source intensities and surface illumination properties), and setting the camera parameters (position, orientation, and lens characteristics).
- Another standard function is *action specification,* which involves the layout of motion paths for the objects and camera. We need the usual graphics routines: viewing and perspective transformations, geometric transformations to generate object movements as a function of accelerations or kinematic path specifications, visible-surface identification, and the surface-rendering operations.
- **Key-Frame Systems** were originally designed as a separate set of animation routines for generating the in-betweens from the user-specified key frames. Now, these routines are often a component in a more general animation package. In the simplest case, each object in a scene is defined as a set of rigid bodies connected at the joints and with a limited number of degrees of freedom. As an example, the single-armed robot in Fig. 5.5 has 6 degrees of freedom, which are

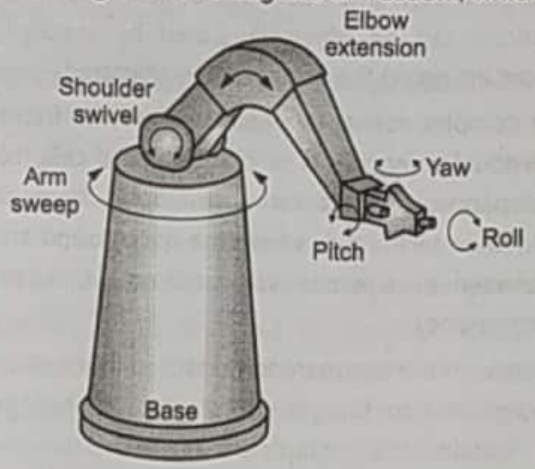


Fig. 5.5 : Degrees of freedom for a stationary, single-armed robot

- Referred to as arm sweep, shoulder swivel, elbow extension, pitch, yaw, and roll. We can extend the number of degrees of freedom for this robot arm to 9 by allowing three-dimensional translations for the base Fig. 5.6.

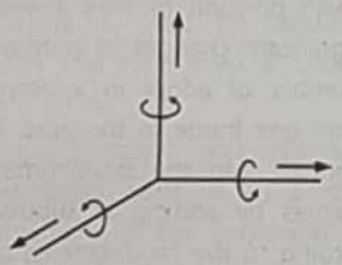


Fig. 5.6 : Translation and rotational degrees of freedom for the base of the robot arm

If we also allow base rotations, the robot arm can have a total of 12 degrees of freedom. The human body, in comparison, has more than 200 degrees of freedom.

- **Parameterized Systems** allow object motion characteristics to be specified as part of the object definitions. The adjustable parameters control such object characteristics as degrees of freedom, motion limitations, and allowable shape changes.

- **Scripting Systems** allow object specifications and animation sequences to be defined with a user-input *script*. From the script, a library of various objects and motions can be constructed.

5.1.5 Key-Frame Systems

- A set of in-betweens can be generated from the specification of two (or more) key frames using a key-frame system. Motion paths can be given with a *kinematic description* as a set of spline curves, or the motions can be *physically based* by specifying the forces acting on the objects to be animated.
- For complex scenes, we can separate the frames into individual components or objects called **cels** (celluloid transparencies). This term developed from cartoon animation techniques where the background and each character in a scene were placed on a separate transparency.
- Then, with the transparencies stacked in the order from background to foreground, they were photographed to obtain the completed frame. The specified animation paths are then used to obtain the next cel for each character, where the positions are interpolated from the key-frame times.
- With complex object transformations, the shapes of objects may change over time. Examples are clothes, facial features, magnified detail, evolving shapes, and exploding or disintegrating objects. For surfaces described with polygon meshes, these changes can result in significant changes in polygon shape such that the number of edges in a polygon could be different from one frame to the next. These changes are incorporated into the development of the in-between frames by adding or subtracting polygon edges according to the requirements of the defining key frames.

5.1.6 Morphing

- Transformation of object shapes from one form to another is termed **Morphing,** which is a shortened form of "metamorphosing." An animator can model morphing by transitioning polygon shapes through the in-betweens from one key frame to the next.
- Given two key frames, each with a different number of line segments specifying an object transformation, we can first adjust the object specification in one of the frames so that the number of polygon edges (or the number of polygon vertices) is the same for the two frames. This preprocessing step is illustrated in Fig. 5.7.
- A straight-line segment in key frame k is transformed into two line segments in key frame k +1. Because key frame k +1 has an extra vertex, we add a vertex between vertices 1 and 2 in key frame k to balance the number of vertices (and edges) in the two key frames. Using linear interpolation to generate the in-betweens, we transition the added vertex in key frame k into vertex 3_ along the straight-line path shown in Fig. 5.7. An example of a triangle linearly expanding into a quadrilateral is given in Fig. 5.8.
- We can state general preprocessing rules for equalizing key frames in terms of either the number of edges or the number of vertices to be added to a key frame. We first consider equalizing the edge count, where parameters Lk and $Lk+1$ denote the number of line segments in two consecutive frames. The maximum and minimum number of lines to be equalized can be determined as

$$L_{max} = \max(L_k, L_{k+1}),\ L_{min} = \min(L_k, L_{k+1}) \quad ...(5.1)$$

Next we compute the following two quantities:

$$N_e = L_{max} \bmod l_{min}$$

$$N_s = \text{int}\left(\frac{L_{max}}{L_{min}}\right) \quad ...(5.2)$$

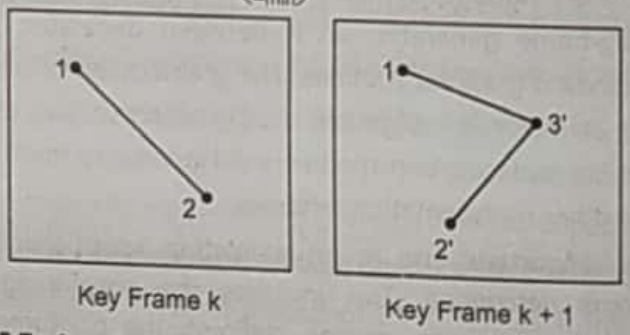


Fig. 5.7 : An edge with vertex positions 1 and 2 in key frame k evolves into two connected edges in key frame k + 1

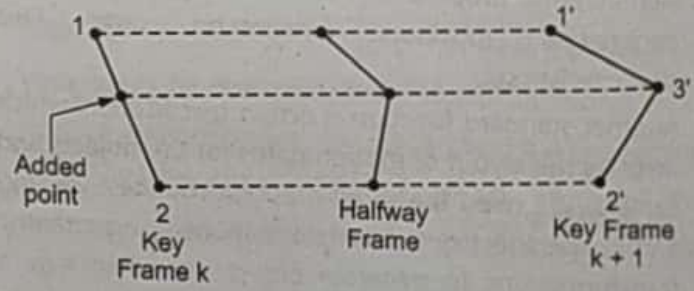


Fig. 5.8 : Linear interpolation for transforming a line segment in key frame k into two connected line segments in key frame k + 1

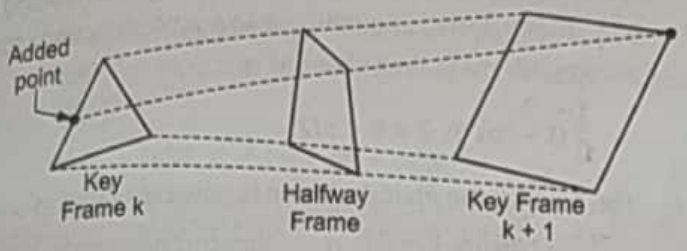


Fig. 5.9 : Linear interpolation for transforming a triangle into a quadrilateral

The preprocessing steps for edge equalization are then accomplished with the following two procedures:

1. Divide *Ne* edges of *keyframe*min into *Ns* + 1 sections.
2. Divide the remaining lines of *keyframe*min into *Ns* sections.

As an example, if Lk = 15 and Lk+1 = 11, we would divide four lines of *keyframe*k+1 into two sections each. The remaining lines of *keyframe*k+1 are left intact.

If we equalize the vertex count, we can use parameters *Vk* and *Vk*+1 to denote the number of vertices in the two consecutive key frames. In this case, we determine the maximum and minimum number of vertices as

$$V_{max} = \max(V_k, V_{k+1})$$
$$V_{min} = \min(V_k, V_{k+1}) \quad \ldots(5.3)$$

Then we compute the following two values:

$$N_{ls} = (V_{max} - 1) \bmod (V_{min} - 1)$$
$$N_p = \text{int}\left(\frac{V_{max} - 1}{V_{min} - 1}\right) \quad \ldots(5.4)$$

These two quantities are then used to perform vertex equalization with the following procedures:

1. Add Np points to Nls line sections of keyframemin.
2. Add *Np* – 1 points to the remaining edges of *keyframe*min.

For the triangle-to-quadrilateral example, *Vk* = 3 and *Vk*+1 = 4. Both *Nls* and *Np* are 1, so we would add one point to one edge of *keyframe*k . No points would be added to the remaining lines of *keyframe*k.

Simulating Accelerations

- Curve-fitting techniques are often used to specify the animation paths between key frames. Given the vertex positions at the key frames, we can fit the positions with linear or nonlinear paths. Fig. 5.10 illustrates a nonlinear fit of key frame positions. To simulate accelerations, we can adjust the time spacing for the in-betweens.

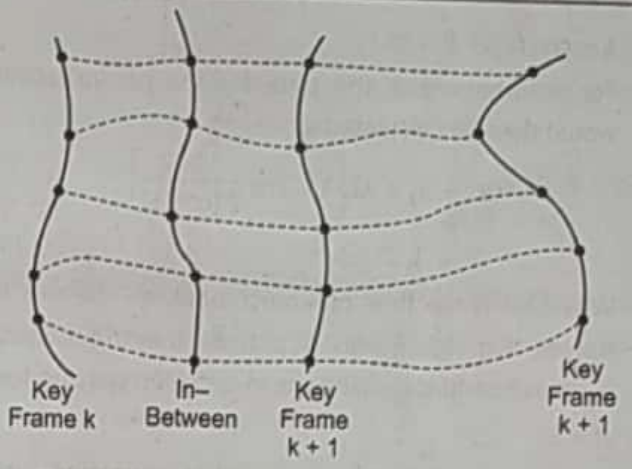


Fig. 5.10 : Fitting key-frame vertex positions with nonlinear splines

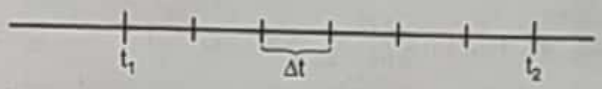


Fig. 5.11 : In-between positions for motion at constant speed

- If the motion is to occur at constant speed (zero acceleration), we use equal interval time spacing for the in-betweens. For instance, with *n* in-betweens and key-frame times of *t*1 and *t*2 Fig. 5.11, the time interval between the key frames is divided into *n*+1 equal subintervals, yielding an in-between spacing of

$$\Delta t = \frac{t_2 - t_1}{n + 1} \quad \ldots(5.5)$$

The time for the *j*th in-between is

$$tB_i = t_1 + i\Delta t$$
$$i = 1, 2, \ldots \pi \quad \ldots(5.6)$$

- This time value is used to calculate coordinate positions, color, and other physical parameters for that frame of the motion.
- Speed changes (nonzero accelerations) are usually necessary at some point in an animation film or cartoon, particularly at the beginning and end of a motion sequence. The startup and slowdown portions of an animation path are often modeled with spline or trigonometric functions, but parabolic and cubic time functions have been applied to acceleration modeling. Animation packages commonly furnish trigonometric functions for simulating accelerations.
- To model increasing speed (positive acceleration), we want the time spacing between frames to increase so that greater changes in position occur as the object moves faster. We can obtain an increasing size for the time interval with the function

$1 - \cos\theta, 0 < \theta < \pi/2$

- For n in-betweens, the time for the jth in-between would then be calculated as

$$tB_j = t_1 + \Delta t\left[1 - \cos\frac{j\pi}{2(n+1)}\right]$$

$$j = 1, 2, \ldots n \qquad \ldots(5.7)$$

- Where Δt is the time difference between the two key frames. Fig. 5.12 gives a plot of the trigonometric acceleration function and the in-between spacing for $n = 5$.
- We can model decreasing speed (deceleration) using the function $\sin\theta$, with $0 < \theta < \pi/2$. The time position of an in-between is then determined as

$$tB_j = t_1 + \Delta t \sin\frac{j\pi}{2(n+1)}, j = 1, 2, \ldots n \qquad \ldots(5.8)$$

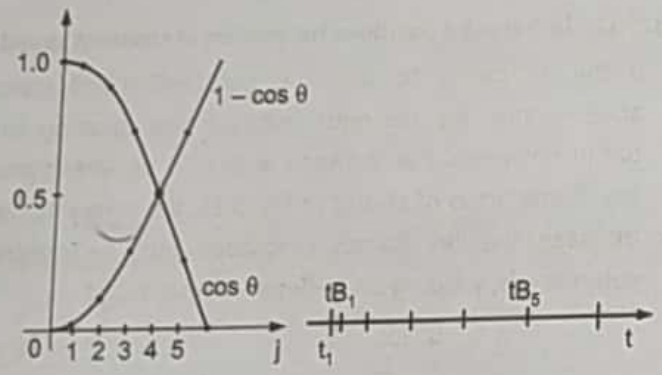


Fig. 5.12 : A trigonometric acceleration function and the corresponding in-between spacing for n = 5 and θ = /π/12in equation 7, producing increased coordinate changes as the object moves through each time interval.

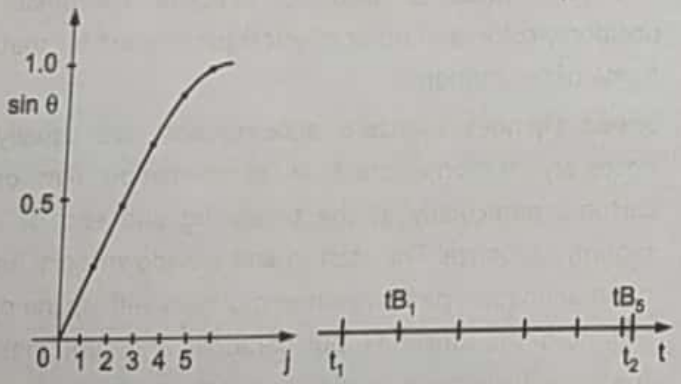


Fig. 5.13 : A trigonometric deceleration function and the corresponding in-between spacing for n = 5 and θ = jπ/12 in Equation 8, producing decreased coordinate changes as the object moves through each time interval.

- A plot of this function and the decreasing size of the time intervals is shown in Fig. 5.13 for five in-betweens.
- Often, motions contain both speedups and slowdowns. We can model a combination of increasing–decreasing speed by first increasing the in-between time spacing and then decreasing this spacing. A function to accomplish these time changes is

$$\frac{1}{2}(1 - \cos\theta), 0 < \theta < \pi/2$$

- The time for the jth in-between is now calculated as

$$tB_j = t_1 + \Delta t \quad ,$$

$$j = 1, 2, \ldots n \qquad \ldots(5.9)$$

- With Δt denoting the time difference between the two key frames. Time intervals for a moving object first increase and then decrease, as shown in Fig. 5.14.

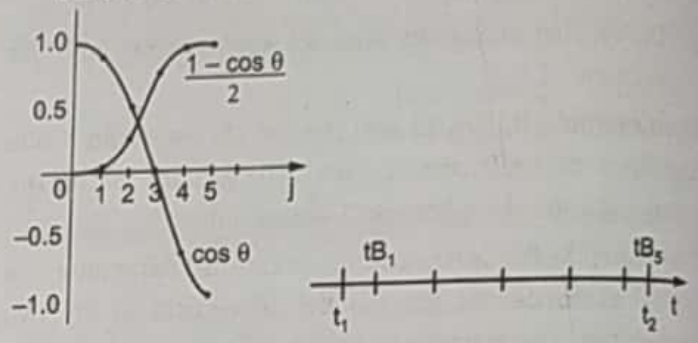


Fig. 5.14 : The trigonometric accelerate-decelerate function (1 – cos θ) / 2 and the corresponding in-between spacing for n = 5 in Equation 9.

- Processing the in-betweens is simplified by initially modeling "skeleton" (wire-frame) objects so that motion sequences can be interactively adjusted. After the animation sequence is completely defined, objects can be fully rendered.

5.1.7 Motion Specifications

- General methods for describing an animation sequence range from an explicit specification of the motion paths to a description of the interactions that produce the motions. Thus, we could define how an animation is to take place by giving the transformation parameters, the motion path parameters, the forces that are to act on objects, or the details of how objects interact to produce motion.

Direct Motion Specification

- The most straightforward method for defining an animation is *direct motion specification* of the geometric-transformation parameters. Here, we explicitly set the values for the rotation angles and translation vectors. Then the geometric transformation matrices are applied to transform coordinate positions. Alternatively, we could use an approximating equation

involving these parameters to specify certain kinds of motions. We can approximate the path of a bouncing ball, for instance, with a damped, rectified, sine curve Fig. 5.15:

$$y(x) = A\,|\sin(\omega x + \theta_0)|e^{-kx} \quad \ldots(5.10)$$

- Where A is the initial amplitude (height of the ball above the ground), ω is the angular frequency, $\theta 0$ is the phase angle, and k is the damping constant. This method for motion specification is particularly useful for simple user programmed animation sequences.

Goal-Directed Systems

- At the opposite extreme, we can specify the motions that are to take place in general terms that abstractly describe the actions in terms of the final results. In other words, an animation is specified in terms of the final state of the movements. These systems are referred to as *goal-directed*, since values for the motion parameters are determined from the goals of the animation. For example, we could specify that

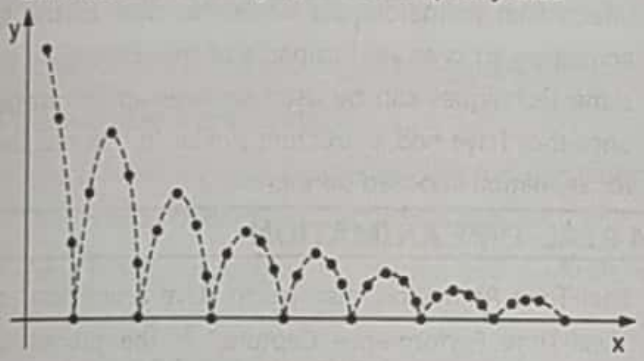


Fig. 5.15

- We want an object to "walk" or to "run" to a particular destination; or we could state that we want an object to "pick up" some other specified object. The input directives are then interpreted in terms of component motions that will accomplish the described task. Human motions, for instance, can be defined as a hierarchical structure of submotions for the torso, limbs, and so forth. Thus, when a goal, such as "walk to the door" is given, the movements required of the torso and limbs to accomplish this action are calculated.

Kinematics and Dynamics

- We can also construct animation sequences using *kinematic* or *dynamic* descriptions. With a kinematic description, we specify the animation by giving motion parameters (position, velocity, and acceleration) without reference to causes or goals of the motion. For constant velocity (zero acceleration), we designate the motions of rigid bodies in a scene by giving an initial position and velocity vector for each object.
- For example, if a velocity is specified as (3, 0, −4) km per sec, then this vector gives the direction for the straight-line motion path and the speed (magnitude of velocity) is calculated as 5 km per sec. If we also specify accelerations (rate of change of velocity), we can generate speedups, slowdowns, and curved motion paths. Kinematic specification of a motion can also be given by simply describing the motion path. This is often accomplished using spline curves.
- An alternate approach is to use *inverse kinematics*. Here, we specify the initial and final positions of objects at specified times and the motion parameters are computed by the system. For example, assuming zero acceleration, we can determine the constant velocity that will accomplish the movement of an object from the initial position to the final position. This method is often used with complex objects by giving the positions and orientations of an end node of an object, such as a hand or a foot. The system then determines the motion parameters of other nodes to accomplish the desired motion.
- Dynamic descriptions, on the other hand, require the specification of the forces that produce the velocities and accelerations. The description of object behavior in terms of the influence of forces is generally referred to as *physically based modeling*. Examples of forces affecting object motion include electromagnetic, gravitational, frictional, and other mechanical forces.
- Object motions are obtained from the force equations describing physical laws, such as Newton's laws of motion for gravitational and frictional processes, Euler or Navier-Stokes equations describing fluid flow, and Maxwell's equations for electromagnetic forces. For example, the general form of Newton's second law for a particle of mass m is

$$F = \frac{d}{dt}(mv) \quad \ldots(5.11)$$

where **F** is the force vector and **v** is the velocity vector. If mass is constant, we solve the equation **F** = m**a**, with **a** representing the acceleration vector. Otherwise, mass is a function of time, as in relativistic motions or the motions of space vehicles that consume measurable

amounts of fuel per unit time. We can also use *inverse dynamics* to obtain the forces, given the initial and final positions of objects and the type of motion required.

- Applications of physically based modeling include complex rigid-body systems and such non rigid systems as cloth and plastic materials. Typically, numerical methods are used to obtain the motion parameters incrementally from the dynamical equations using initial conditions or boundary values.

5.2 DEVICES FOR PRODUCING ANIMATION

A Stylus Pen

- Using a stylus pen on your tablet can help you get used to using a screen as your canvas so you can make the switch to digital art more easily.

A Small Graphics Tablet

- Once you're ready to break into the digital art scene, then you'll definitely need a graphics tablet. You can hook it up to your computer and start refining your art skills by having your sketches transfer right to your screen.

Drawing Gloves

- These gloves reduce the friction between your hands on the screen to keep your drawing precise.

A Large Graphic Tablet (Really)

- Graphics tablet will need to keep up with your growing imagination

A Flash Animation Program

- When you start to actually animate things, then you'll need the software to watch your creations come to life.

5.3 COMPUTER-ASSISTED ANIMATION

- Computer-assisted animation is usually classed as two-dimensional (2D) animation. Creators drawings either hand drawn (pencil to paper) or interactively drawn(drawn on the computer) using different assisting appliances and are positioned into specific software packages. Within the software package the creator will place drawings into different key frames which fundamentally create an outline of the most important movements. The computer will then fill in all the " in-between frames" commonly known as Tweening. Computer assisted animation is basically using new technologies to cut down the time scale that traditional animation could take, but still having the elements of traditional drawings of characters or objects.
- Two examples of films using computer-assisted animation are Beauty and the Beast and Antz.
- 2D animation can only be automated to the extent that the computer acts as an interactive assistant to the animator. The key problem is that the 3D information which is implicit in the animated drawings is unavailable and this has encouraged the view of 2D animation being a subset of 3D animation in its fullest generality. However, introducing more than the absolute minimum of 3D information, essentially that contained in a hierarchy of drawing overlays, has matching advantages and disadvantages because animators, for aesthetic reasons, deliberately break the rules of geometry and physics as they apply to real-world objects. A software environment which only supports 2D functionality and drawing overlays is sufficient to promote cost-effectively the full range of effects that animators use while the state of the 3D animation art is as yet incapable of this. Essentially the same techniques can be used on live-capture images once they have had a structure similar to that required for animation imposed on them.

5.4 REAL-TIME ANIMATION

- Real-Time Animation, also called "Live Animation," or "Real-Time Performance Capture," is the process of using a motion capture system to puppet a 3D character live and in real-time. It can be used for previsualization, so a director can preview an animation while the mocap actor is performing, or as a live feed so a real person or audience can communicate with the 3D character in front of them.
- Real-time mocap animation has the ability to bring 3D characters to life and allows them to be present at game cons, appear on talk shows, speak at keynotes, and cover breaking news.
- Real time animation is also called motion capture. It captures the motion of the human body through the use of sensors specifically placed on the joints. This data is relayed through the capture terminal to a computer graphics program which transposes the captured data onto a digital puppet. Generally if you

visit any SIGGRAPH event, you would be able to see the display of real time animation. Artist performing live to a digital puppet placed in the same position on the stage. It's amazing to watch.A good example is the movie Avatar.

5.5 ANIMATION METHOD OF CONTROLLING

1. Procedural control
2. Full explicit control
3. Kinematic as well as dynamic
4. Tracking live action
5. Constraint based system
6. Manual Recycling
7. Auto-Recycling
8. Create control and stop as well as animation controller allows to start (see method start) animations, pause (see method pause) see method resumes see method speed, and see method state.
9. Adjust the animation time (see property time).

1. Explicitly Declared Control:

- Object level by specifying simple transformations (translations, rotations , scaling) to objects.
- Frame level by specifying key frames and methods for interpolating between them.

2. Procedural Control

- Based on communication among different objects. each object obtains knowledge about the static/dynamic properties of other objects. Can be used to ensure consistency

3. Constraint-Based Control

- Many objects movements in the real world are determined by other objects which they come in contact. E.g. presence of strong wind Environment based constraints can be used to control object's motion.

4. Analyzing Live Action-Based Control

- Control is achieved by examining the motions of objects in the real world. Rotoscoping is a technique where animators trace live action movement , frame by frame, for use in animated films. Pre-recorded live-film images were projected onto a frosted glass panel and redrawn by an animator.

5. Kinematic and Dynamic Control

- Kinematics refer to the position and velocity of points" The cube is at the origin at time t = 0. Then, it moves with constant acceleration (1 meter along x, 1 meter along y, 5 meters along z) "
- Dynamics takes into account the physical laws that govern kinematics
- Newton laws for the movement of large objects
- Euler-Lagrange equations for fluids
- A particle moves with an acceleration proportional to the forces acting on it.

EXERCISE

1. What is a segment table? Explain the operations that can be performed on a segment table?
2. What is segment? How do we create it? Why do we need segments? Explain in detail the various operations of segment.
3. What is a segment ? Give its structure and also describe various operations carried out on the segment.
4. Describe the various operations carried out on the segment.
5. Write advantages and disadvantages of segments.
6. What is segment? Explain segment table.
7. Explain the data structures that can be used to implement the segment table.
8. Write the algorithm for the following
   (i) Change the visibility attribute of segment
   (ii) Delete a segment
9. Describe various operations carried out on the segment.
10. What is segment? Explain transformation operation on segment.
11. What is segment and segment table?
12. What are the various methods of controlling animation? Explain in detail.

13. Describe the steps required to produce real time animation.
14. Define animation and explain the methods of controlling the animation. Give different types of animation languages.
15. Explain animation method with controlling.
16. Enlist and explain devices for producing animation.
17. Explain the concept of morphing. State the necessary equations.
18. What is the difference between conventional and computer based animations? What are the various methods of controlling animation?
19. Define animation. Explain the methods for controlling animations.
20. Write basic guidelines for animation and gaming technology.
21. What are the applications of morphing?
22. Explain morphing? What is simulating acceleration?

✠ ✠ ✠

Model Question Papers for End-Semester Examination

PAPER I

Time : 3 Hours **Max. Marks : 60**

Instructions to the candidates :

1. Each Question carries 12 Marks.
2. Attempt any five questions to the following.
3. Illustrate your answers with neat sketches, diagram etc., wherever necessary.
4. If some part or parameter is noticed to be missing, you may appropriately assume it and should mention it clearly.

Attempt any Five Questions

1. **Attempt any Two of the Following :**
   - (a) Define Line, Line Segment , Pixel and Frame Buffer. [6]
   - (b) What is Anti-aliasing. Describe the methods for character Generation. [6]
   - (c) Write short note on scanner, joystick, touch panel, mouse. [6]
2. **Attempt any three of the following.**
   - (a) Write and explain Bresenham's line algorithm and find out which pixel would be turned on for the line with end points (3, 2) to (7,4) using the same. [4]
   - (b) Explain DDA line drawing algorithm with example. [4]
   - (c) Write short note on polygon clipping. [4]
   - (d) Define text clipping. [4]
3. **Attempt any two of the following.**
   - (a) Explain rotation about an arbitrary axis in 3D. [6]
   - (b) Explain Translation, Scaling & Rotation for 3D. [6]
   - (c) Explain Backface Detection and removal and Painter's Algorithm . [6]
4. **Attempt any two of the following.**
   - (a) Write Short note OpenGL. [6]
   - (b) Define Graphics Standard. [6]
   - (c) Explain Windows and Motif. [6]
5. **Attempt any two of the following.**
   - (a) Define animation. Explain the methods for controlling animation. [6]
   - (b) Give any 4 basic guideline for animation. [6]
   - (c) Explain computer assisted animation and devices for producing animation. [6]
6. **Attempt any two of the following.**
   - (a) Define animation. Explain the methods for controlling animations. [6]
   - (b) Write the algorithm for the following
     (i) Change the visibility attribute of segment (ii) Delete a segment [6]
   - (c) Describe the various operations carried out on the segment. [6]

✠ ✠ ✠

PAPER II

Time : 3 Hours **Max. Marks : 60**

Instructions to the candidates :

1. Each Question carries 12 Marks.
2. Attempt any five questions to the following.
3. Illustrate your answers with neat sketches, diagram etc., wherever necessary.
4. If some part or parameter is noticed to be missing, you may appropriately assume it and should mention it clearly.

Attempt any Five Questions

1. Attempt any Two of the Following :

(a) Explain Frame Buffer and its Types. [6]

(b) Describe the methods for character Generation. [6]

(c) Describe display devices. [6]

2. Attempt any two of the following.

(a) Enlist any three methods of polygon filling algorithms. Explain even-odd method of inside test. [6]

(b) Explain Cohen-Sutherland outcode algorithm with example. [6]

(c) Write and explain with an example Cohen-Sutherland line clipping algorithm. [6]

3. Attempt any three of the following.

(a) Which are the various approaches used to represent polygon? [4]

(b) What is inside test? Explain even odd method in detail. [4]

(c) Explain Projection and Types. [4]

(d) Describe hidden surface and Types of surface. [4]

4. Attempt any two of the following.

(a) Explain Backface Detection and removal and Painter's Algorithm. [6]

(b) Write basic guidelines for animation and gaming technology. [6]

(c) Write Short note OpenGL. [6]

5. Attempt any two of the following.

(a) Explain morphing? What is simulating acceleration? [6]

(b) Define animation. Explain the methods for controlling animations. [6]

(c) What are the various methods of controlling animation? Explain in detail. [6]

6. Attempt any two of the following.

(a) Give any 4 basic guideline for animation. [6]

(b) Explain computer assisted animation and devices for producing animation. [6]

(c) What is a segment table? Explain the operations that can be performed on a segment table? [6]

✠ ✠ ✠

PAPER III

Time : 3 Hours Max. Marks : 60

Instructions to the candidates :

1. Each Question carries 12 Marks.
2. Attempt any five questions to the following.
3. Illustrate your answers with neat sketches, diagram etc., wherever necessary.
4. If some part or parameter is noticed to be missing, you may appropriately assume it and should mention it clearly.

Attempt any Five Questions

1. **Attempt any Two of the Following :**
   (a) Explain Aliasing and Antialiasing [6]
   (b) Write down the different methods for testing a pixel inside of polygon? [6]
   (c) Enlist any three polygon filling algorithms. Explain even-odd method of inside test. [6]
2. **Attempt any three of the following.**
   (a) What are the different types of polygon? [4]
   (b) Explain the different methods for testing a pixel inside a polygon. [4]
   (c) Write flood fill algorithm [4]
   (d) Explain Cohen-Sutherland outcode algorithm with example. [4]
3. **Attempt any two of the following.**
   (a) Obtain the 3-D transformation matrices for :
   (i) Translation,(ii) Scaling, (iii) Rotation about an arbitrary axis [6]
   (b) Explain various steps to perform rotation about x-axis, y-axis and z-axis in 3D. [6]
   (c) Consider the square A(1, 0), B(0, 0), (0, 1), D(1, 1). Rotate the square ABCD by 45° anticlockwise about point A(1, 0). [6]
4. **Attempt any two of the following.**
   (a) Explain mouse and keyboard interaction in OpenGL. [6]
   (b) Explain Motif and CDE Source Code Synchronization. [6]
   (c) Draw and explain X Window System protocols and architecture [6]
5. **Attempt any two of the following.**
   (a) Explain with neat diagram reference model of graphics standards. [6]
   (b) What is segment? How do we create it? Why do we need segments? Explain in detail the various operations of segment. [6]
   (c) Explain morphing? What is simulating acceleration? [6]
6. **Attempt any two of the following.**
   (a) What are the various methods of controlling animation? Explain in detail. [6]
   (b) Explain the data structures that can be used to implement the segment table. [6]
   (c) Write the algorithm for the following [6]
   (i) Change the visibility attribute of segment (ii) Delete a segment

✠ ✠ ✠

PAPER IV

Max. Marks : 60

Time : 3 Hours

Instructions to the candidates :

1. Each Question carries 12 Marks.
2. Attempt any five questions to the following.
3. Illustrate your answers with neat sketches, diagram etc., wherever necessary.
4. If some part or parameter is noticed to be missing, you may appropriately assume it and should mention it clearly.

Attempt any Five Questions

1. Attempt any Two of the Following :

(a) Enlist any three methods of polygon filling algorithms. Explain even-odd method of inside test. [6]

(b) Write short note on scanner, joystick, touch panel, mouse. [6]

(c) Define Line, Line Segment , Pixel and Frame Buffer. [6]

2. Attempt any three of the following.

(a) Explain even-odd method of inside test. [4]

(b) Define Line, Line Segment , Pixel and Frame Buffer. [4]

(c) Explain boundary fill method for polygon? [4]

(d) Explain the 3-D viewing process with various 3-Dviewing parameters. [4]

3. Attempt any two of the following.

(a) Consider the square A(2, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square anticlockwise direction followed by reflection about x-axis. [6]

(b) What is necessary for 3D clipping and windowing algorithm? Explain any one of 3D clipping algorithm. [6]

(c) What are the different steps for rotation about an arbitrary axis? [6]

4. Attempt any two of the following.

(a) What is OpenGL? Explain OpenGL architecture with neat diagram. [6]

(b) Explain GLU, GLUT Package in OpenGL. [6]

(c) Explain mouse and keyboard interaction in OpenGL. [6]

5. Attempt any two of the following.

(a) Write basic guidelines for animation and gaming technology. [6]

(b) Define animation. Explain the methods for controlling animations. [6]

(c) Explain with neat diagram reference model of graphics standards. [6]

6. Attempt any two of the following.

(a) What is segment? How do we create it? Why do we need segments? Explain in detail the various operations of segment. [6]

(b) Explain the concept of morphing. State the necessary equations. [6]

(c) Describe the steps required to produce real time animation. [6]