P.G.DIPLOMA IN DATA SCIENCE

COMPUTER GRAPHICS

मध्यप्रदेश भोज (मुक्त) विश्वविद्यालय – भोपाल
MADHYA PRADESH BHOJ (OPEN) UNIVERSITY - BHOPAL

COURSE WRITERS

Rajendra Kumar, Professor and Head of Computer Science and Engineering Department, Vidya College of Engineering, Meerut
Units (1.2-1.2.1, 3.7, 3.7.4, 5.2.1, 5.2.2, 5.4, 5.5, 5.6, 5.7, 5.10)

Anirban Mukhopadhyay, Associate Professor, R.C.C. Institute of Information Technology, Kolkata
Arup Chattopadhyay, Head, Scientific and Technical Group, D.O.E.A.C.C. Centre, Jadavpur University, Kolkata
Units (1.0-1.1, 1.2.2, 1.3, 1.4, 1.6-1.8, 1.10-1.15, 2.0-2.2.9, 2.2.11, 2.2.14, 2.7-2.11, 3.0-3.1, 3.3, 3.4, 3.5, 3.7.2-3.7.3, 3.8, 3.10-3.15, 4.0-4.4, 4.6-4.10, 5.0-5.1, 5.8, 5.8.3, 5.11-5.15)

V.K. Khanna, Associate Professor, Dept. of Mathematics, Kirori Mal College, University of Delhi, Delhi
S.K. Bhambri, Associate Professor, Dept. of Mathematics, Kirori Mal College, University of Delhi, Delhi
Units (2.3, 2.4.2, 2.5, 2.6, 3.2)

Dr. Kavita Saini, Professor, School of Computing Science and Engineering (S.C.S.E.), Galgotias University, Greater Noida, Uttar Pradesh
Units (1.3.1-1.3.2, 1.4.1-1.4.5, 1.5, 1.5.1-1.5.2, 1.9-1.9.1, 2.2.10, 2.2.12-2.2.13, 2.4-2.4.1, 2.6.1, 3.6, 3.7.1, 3.7.5-3.7.7, 3.9, 4.5, 5.2, 5.3, 5.5.1, 5.8.1-5.8.2, 5.8.4, 5.9)

Published by Registrar, MP Bhoj (Open) University, Bhopal in 2020

Vikas® is the registered trademark of Vikas® Publishing House Pvt. Ltd.

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

SYLLABI-BOOK MAPPING TABLE

Computer Graphics

Syllabi	Mapping in Book
Unit – I **Computer Graphics Application** : Introduction to Computer Graphics, Application of Computer Graphics, **Devices for Graphics Output**, Monitor Basics, Picture Tube, Display Basics, Text Mode and Graphics Mode, Adapters and Displays, Monochrome Display Adapter (MDA), Color Graphics Adapter (CGA), Hercules Graphics Card, Enhanced Graphics Adapter, Professional Graphics Adapter, Digital vs Analog, Video Graphics Array, Super VGA, Refresh Cathode-Ray Tubes, Raster-Scan Displays, Random-Scan Displays, **Computer Display**, Flat-Panel Displays, Raster Scan Systems, Random Scan Systems, Hard Copy Output Devices	**Unit-1:** Computer Graphics Applications and Devices **(Pages 3-44)**
Unit – II **Graphics Input Devices**, Keyboards, Mouse, Trackball and Spaceball, Joysticks, Data Glove, Digitizers, Image Scanners, Touch Panels, Light Pens, Voice Systems, Input of Graphical Data, Logical Classification of Input Devices, Input Functions, Initial Values for Input Device Parameters, Interactive Picture Construction Techniques, **Matrices and Determinants**, Matrices, Types of Matrices, Determinants, Matrix (Definition), Type of Matrices, Submatrices of a Matrix: (Definition), Equality of Two Matrices: (Definition), Addition of Matrices, Scalar Multiplication, Multiplication of Two Matrices (Definition), Transpose of a Matrix (Definition), Symmetric and Skew-Symmetric Matrices, Adjoint and Inverse of a Matrix, Determinants, Properties of Determinants **Vectors**, Definition of a Vector, Vectors and Coordinate System, Algebra of Vectors Addition, Multiplication of a Vector by a Scalar, Components of a Vector, Direction and Magnitude of a Vector in Terms of its Components, Collinear and Coplanar Vectors, Some Applications to Geometry.	**Unit-2:** Graphics Input Devices, Matrices, Determinants and Vectors **(Pages 45-118)**
Unit – III **Raster Scan Graphics**, Derivative of a Function, Digital Differential Analyzer, Bresenham's Algorithm, Integer Bresenham's Algorithm, General Bresenham's Algorithm, Circle Generation - Bresenham's Algorithm, Scan Conversion-Generation of the Display, Real-Time Scan Conversion, Run-Length Encoding, Cell Encoding, Frame Buffers, Addressing the Raster, Line Display, Character Display, Solid Area Scan Conversion, Polygon Filling, Scan-Converting Polygons, A Simple Ordered Edge List Algorithm, More Efficient Ordered Edge List Algorithms, The Edge Fill Algorithm, The Edge Flag Algorithm, Seed Fill Algorithms, A Simple Seed Fill Algorithm, A Scan Line Seed Fill Algorithm, Fundamentals of Antialiasing, Simple Area Antialiasing, The Convolution Integral And Antialiasing, Half toning, **Windows and Clipping**, Two-Dimensional Clipping, Sutherland-Cohen Subdivision Line-Clipping Algorithm, Midpoint Subdivision Algorithm, Generalized Two-Dimensional Line Clipping for Convex Boundaries.	**Unit-3:** Raster Scan Graphics, Windows and Clipping **(Pages 119-207)**

Unit – IV

2D- Transformation, Representation of Points, Transformations and Matrix, Transformation of Straight Line, 2-D - Rotation, Reflection, Scaling, Combined Transformations, Translation and Homogeneous Coordinates, Translation, Rotation about an Arbitrary Point, Reflection through an Arbitrary Line**, 3-D-Transformation**, Representation of Points, 3D- Scaling, 3D- Shearing, 3D- Rotation, Three Dimensional Translation, 3D- Reflection, Multiple Transformations, Rotation about an Axis Parallel to a Coordinate Axis, Rotation about an Arbitrary Axis in Space.

Unit-4: 2D and 3D Transformation **(Pages 209-257)**

Unit – V

The Dimensional Perspective Geometry, Geometric Projection, Orthographic Projections, Oblique Projections, Perspective Transformations, Single-Point Perspective Transformation, Two-Point Perspective Transformation, Three-Point Perspective Transformation.

Hidden-Surface, Lines and Bezier Curve, Hidden Surfaces and Lines, Back-Face Detection, Back-Face Removal, Z-Buffers Algorithm, The Painter's Algorithm, Binary Space Partition, Franklin Algorithm, Cubic Belier Curve (No Derivations Needed), Properties of Bezier Curve, Joining Condition, Problems, **Multimedia and Animation**, Multimedia, Multimedia Terms, Multimedia Hardware, Hardware Peripherals, Basic tools in Multimedia, Multimedia Building Blocks (Media Forms/Elements), Sound, Image, Animation, Video, JPEG, MPEG, DVI Indeo, P*64, Graphic File Formats, Multimedia Applications

Unit-5: Geometry Perspective, Hidden-Surface, Lines and Bezier Curve, Multimedia and Animation **(Pages 259-327)**

CONTENTS

INTRODUCTION

NOTES

Computers have brought about major changes in all spheres of life and also brought about major changes in the way we communicate. Today, it is extremely difficult to imagine a world without computers. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

The term computer graphics covers almost everything that is not text or sound and signifies representation and manipulation of image data by a computer. Computer graphics has evolved as a branch of computer science which studies methods to digitally synthesize and manipulate visual data. Computer graphics can broadly be categorized into two-dimensional, three-dimensional and animated graphics. Today, computer generated imagery has permeated every aspect of our life. It can be found in newspapers, television, medical reports, weather forecast, etc. 2D computer graphics are the computer based generation of digital images-mostly from two-dimensional models, such as 2D geometric models, text and digital images, and by techniques specific to them. 2D computer graphics are primarily used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc. In such applications, the two-dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value. Hence, two-dimensional models are preferred, because they give more direct control of the image as compared to 3D computer graphics, whose approach is more akin to photography than to typography.

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data stored in the computer for performing calculations and rendering 2D images. Such images may be used for later display or for real-time viewing. Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques. 3D computer graphics are often referred to as

3D models. A 3D model is the mathematical representation of any three-dimensional object. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called 3D rendering or used in non-graphical computer simulations and calculations. There are 3D computer graphics software that helps the users to create 3D images.

NOTES

This book, *Computer Graphics* is divided into five units that follow the self-instruction mode with each unit beginning with an Introduction to the unit, followed by an outline of the Objectives. The detailed content is then presented in a simple but structured manner interspersed with Check Your Progress Questions to test the student's understanding of the topic. A Summary along with a list of Key Terms and a set of Self-Assessment Questions and Exercises is also provided at the end of each unit for recapitulation.

UNIT 1 COMPUTER GRAPHICS APPLICATIONS AND DEVICES

NOTES

Structure

1.0 INTRODUCTION

Computer graphics is the discipline of producing pictures or images using a computer. Computers have become important tools that produce pictures economically. Graphical displays can be used in all areas. Therefore, its extensive usage is not surprising. Although early applications in science and engineering had to rely on cumbersome and expensive equipment, current advances in computer technology have made interactive computer graphics a practical tool. Today, computer graphics are being used in diverse areas like engineering, science, business, medicine, government, industry, entertainment, training, education and art.

NOTES

The display medium for computer graphics-generated pictures has become widely diversified. Typical examples are CRT-based display, Liquid Crystal Display (LCD), Light Emitting Diode (LED), plasma-based display, stereoscopic display, etc. The CRT display is by far the most common display technology and most of the fundamental display concepts are embodied in the CRT technology.

Computer graphics deals exclusively with various input devices with different functional capabilities. Based on the logical interaction types, the input devices can be broadly classified as: locator devices that indicate a position (e.g., point coordinate) or orientation, pick devices that can select a graphical object, valuators that are used to input scalar values like rotation angle, scale factors, etc., keyboard or text input device and choice devices that are used to select menu options.

In this unit you will study about the introduction to computer graphics, application of computer graphics, monitor basics, display basics, text mode, graphics mode, adapters and displays, Monochrome Display Adapter (MDA), Color Graphics Adapter (CGA) and Hercules graphics card, enhanced graphics adapter, professional graphics adapter, digital vs analog, refresh cathode-ray tubes, raster scan displays, random scan displays, flat-panel display, raster scan systems, random scan systems and hard copy output devices.

1.1 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Explain the concept of computer graphics
- Discuss the various application of computer graphics
- Describe the display basics text mode and graphics mode
- Describe the graphic cards, adapters and displays
- Differentiate between the digital and analog methods of communication
- Describe the refresh cathode-ray tubes
- Explain the raster and random scan displays
- Elaborate on the flat-panel display

1.2 INTRODUCTION TO COMPUTER GRAPHICS

The term ‘computer graphics’ was first used by William Fetter in 1960. Fetter was a graphic designer for Boeing Aircraft Co. The term was actually given to him by Verne Hudson. The demonstration of computer graphics technology led to the development of computer graphics. The projects in computer graphics (like the SAGE and Whirlwind projects) gave an impetus to computer graphics as a discipline by introducing the CRT (Cathode Ray Tube) as a viable display and interaction interface, as also by introducing the light pen as an important graphics input device. The TX-2 computer developed in 1959 by MIT’s Lincoln Laboratory further

NOTES

continued the development of digital computers and interactive computer graphics.

A light pen, a display unit, and a bank of switches were the main components connected with the interface on which the first interactive computer graphics system was based. The TX-2 architecture was integrated with a number of man-machine interfaces. All that these interfaces needed was a user that would use them to make an on-line computer. A user was able to draw on the computer with a simple cathode ray tube and light pen on the TX-2's console and, the Sketchpad, and with it, the interactive computer graphics was born. On the TX-2 computer in the Lincoln Labs, the scientists' research work made them the 'grandfather' of Graphical User Interfaces (GUIs) and interactive computer graphics. The study and experimentations at the Massachusetts Institute of Technology (MIT) shaped the early computer and computer graphics industries.

Personal Computers (PCs) became more powerful during the late 1970s. These PCs could draw both complex and basic designed and shapes. During the 1980s graphic designers and users realized the significance of the PC, particularly the Macintosh and Commodore Amiga as a fundamental design tool; one that could not only draw more accurately, but also save time compared to other methods. SGI computers, powerful as they were, made three dimensional (3D) computer graphics effective in the late 1980s. Later these were used to create the first fully computer-generated short animations. Macintosh has been one of the most accepted tools for computer graphics in businesses and graphic design studios.

From the 1980s onwards, modern computer systems have frequently used a Graphical User Interface (GUI) to represent data and information with the help of symbols, icons and shortcuts, rather than the text user interface. Graphics are one of the five major key elements in the design of multimedia applications.

Three dimensional graphics became more popular during the 1990s in game designing, multimedia and animations. 'The Quake,' one of the first fully 3D games, was released in 1996. *Toy Story*, the first full-length computer-generated animation film, was commercially released in cinemas worldwide in 1995. Since then, computer graphics have become more truthful and comprehensive, due to more advanced computers and better 3D modelling software applications.

Computer Graphics

Computer graphics is the discipline of producing pictures or images using a computer. It includes creation of model, manipulation and storage of geometric objects, reproduction (an image converted from a scene), transformation (primitive graphics operations), illumination, rasterization, animation, and shading of the image.

Computer graphics are broadly used in such activities as graphics based presentations, paint systems, image processing, simulation, virtual reality and Computer-Aided Design (CAD) and entertainment. From the earliest text character images of a non-graphic mainframe computers to the latest photographic quality images of a high resolution personal computers, from vector displays to raster displays, from two dimensional (2-D) input, to three dimensional (3-D) input, computer graphics has gone through its short, rapid changing history.

NOTES

Types of Computer Graphics

In general, there are two types of computer graphics: vector (which is composed of paths) and raster (which is composed of pixels). Vector graphics use mathematical relationships between pixels and the paths connecting them to represent an image. Vector graphics are composed of paths. The image in Figure 1.1 (a) represents a bitmap and the image in Figure 1.1 (b) represents a vector graphic. Classically, raster images are more commonly called bitmap images. A bitmap image uses a matrix of individual pixels where each pixel in the image may contain different colours or brightness. Bitmaps consist of pixels. They may be shown at four times of the actual size to overstress the fact that the edges of a bitmap become uneven as it is scaled up.

Fig. 1.1 (a) *Bitmap Image*

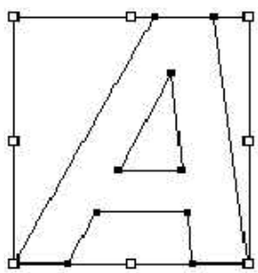


Fig. 1.1 (b) *Vector Graphic*

With the help of various image handling tools and by using point-to-point method rather than by pixels alone, computers can display various fonts and images. When a user scales an image up, the advantage of using a page-description language such as PostScript becomes clear. The more jagged it appears, the larger the user displays a bitmap. On the other hand, a vector image remains even at any size. That is because PostScript and TrueType fonts always appear smooth and they are based on vectors.

One can partially overcome the uneven appearance of bitmap images with the elp of 'anti-aliasing.' Anti-aliasing can be defined as the application of frail transitions in the pixels along the edges of images to minimize the uneven effect as shown in Figure 1.2 (a) and. A scalable vector image always appears smooth as shown in Figure 1.2 (b).

Fig. 1.2 (a) *Anti-Aliased Bitmap Image*

Fig. 1.2 (b) *Smooth Vector Image*

Bitmap images require higher resolutions (large number of rows and columns in the screen) and anti-aliasing for a smooth (even surface) appearance. On the other hand, vector-based graphics are mathematically much more useful and appear smooth at any resolution.

Graphics Primitives

A pixel (also called picture element) can be defined as the smallest piece of information in an image. Pixels are normally arranged in a regular two-dimensional grid (matrix), and are frequently represented using squares, rectangles or dots. Each pixel represents a sample of an original image, where more samples characteristically provide a more precise representation of the original image. The brightness of each pixel is incompatible. In typical colour systems (like RGB or CMYK), each pixel has three or four basic colour components such as red, green and blue (RGB system), or Cyan, Magenta, Yellow and Black (CMYK).

The display resolution of a digital television or computer monitor typically refers to the number of distinct pixels in each dimension that can be displayed.

A palette is a given finite set of colours for the management of digital images, or a small on-screen graphical element for choosing from a limited set of choices which are not necessarily colours.

Graphics Image File Format

There are several graphic image file formats that are used by most of the graphics systems. The following are the most regularly used formats:

Web Document Images

Web document images can be of two types as follows:

(a) ***Graphics Interchange Format (GIF):*** Images made using this format use a fixed colour palette which is limited to only 256 colours. This format downloads small, compressed files quickly from the Web. This format is

NOTES

most suitable for images with solid colours or uniform colour areas such as illustrations and logos.

(b) ***Joint Photographic Experts Group (JPEG):*** These files are used for photographic (continuous colour tone) images, i.e., those images that have a continuous colour tone. Unlike the Graphics Interchange Format files the Joint Photographic Experts Group format takes advantage of the full spectrum of colours available to the display unit. The JPEG format also uses compression for making smaller files and for obtaining faster downloads over the World Wide Web. However, unlike the compression method used in GIF files, the JPEG compression is also 'lossy compression' which means it discards some data in the decompression process. Once a file is saved in the JPEG format some data is lost permanently. But this does not affect the image.

Printed Documents

The following are the two types of printed documents.

(a) ***Encapsulated PostScript (EPS):*** It is an image file format used for both vector graphics and bitmaps. EPS files have a PostScript description of the graphic data within them. The EPS files are exclusive in that the graphics users use them for bitmap images, vector graphics, type, or even entire pages.

(b) ***Tagged-Image File Format (TIFF):*** Such files are used for bitmap format only. The TIFF formats are the files that are supported by virtually all graphics applications.

1.2.1 Applications of Computer Graphics

This section will discuss the applications of computer graphics for which an understanding of Computer-Aided Design (CAD) is essential.

Computer graphics is mainly used in the process of design, generally for engineering and architectural systems. But almost all products are now computer designed. Generally Computer-Aided Design (CAD) methods are now used to design automobiles, buildings, watercraft, aircraft, embedded systems, spacecraft, textiles and many other products.

Generally, objects are first displayed for design applications in a wireframe outline form which describes the internal features and the overall shape of the object. Wireframe displays also allow designers to rapidly see the effects of interactive adjustments to design the shape. The software packages for CAD applications generally provide a multi-window environment.

Animations are often used in CAD applications. Real time animations using wireframe displays on a video monitor are used to test the performance of a vehicle or system. When one does not display objects with rendered surfaces, the calculations for each segment of the animation can be performed quickly to produce a smooth real-time motion on the screen. Also, wireframe displays allow the designer to look into the interior of the vehicle and to watch the behaviour of inner components during the motion. Animations in *virtual reality* environments are used to determine how vehicle operations are affected by certain motions.

When object designs are complete, or near completion, realistic lighting models and surface rendering is applied to produce the displays which will show the appearance of the final product. Realistic displays can also be generated to advertise automobiles and other vehicles using special lighting effects and background scenes.

Building Architectures

Architects also use interactive graphics methods to layout floor plans, which show the positioning of the rooms, doors, windows, stairs, shelves, counters, and other building features. Working from the display of a building layout on a video monitor, an electrical designer can try out arrangements for wiring, electrical outlets, and fire warning systems. Also, facility-layout packages can be applied to the layout to determine space utilization in an office or on a manufacturing floor.

Realistic displays of architectural designs, allow both architects and their clients to study the appearance of a single building or a group of buildings, like a campus or an industrial complex. In addition to the realistic building displays, the graphics packages also provide facilities for experimenting with three-dimensional interior layouts and lighting.

Computer Art

Computer graphics methods are widely used in fine art as well as commercial art applications. Artists use a variety of computer methods including special purpose hardware, paintbrush programs, specific purpose software, desktop publishing software, CAD packages, animation packages, etc., to provide facilities for specifying object motions and designing object shapes.

The paintbrush program allows graphics users to paint pictures available on the video monitor. Actually the picture is usually painted electronically on a graphics tablet using a stylus, which can simulate different brush strokes, brush widths and colours.

Fine artists use a variety of other computer technologies to produce images. To create a picture, artists use a combination of three-dimensional modeling packages, texture mapping, drawing programs and CAD software. In the mathematical art, artists use a combination of mathematical functions, fractal procedures, mathematical software, ink-jet printers, and others systems to create a variety of three-dimensional and two-dimensional shapes and stereoscopic image pairs.

A common graphics method employed in many commercials is *morphing*, where one object is transformed into another. This method has been used in TV commercials to turn an oil container into an automobile engine, an automobile into a tiger, a puddle of water into a tire, and one person face into another face.

Presentation Graphics

A major application area of computer graphics is presentation graphics, which is used to produce illustrations for reports or to generate 35-mm slides or transparencies to use with projectors. Presentation graphics is commonly used to summarize statistical, financial, mathematical, economic and scientific data for managerial reports, research reports, and other types of reports. Workstation

NOTES

devices and service bureaus exist for converting screen displays into 35-mm slides or overhead transparencies for use in presentations. The main examples of presentation graphics are bar charts, pie charts, line graphs, surface graphs and other displays.

Entertainment

These days, computer graphics methods are widely used to make music videos, motion pictures, television shows, etc. Many TV shows and sports telecast regularly employ computer graphics methods. Music videos use graphics in several ways. Graphics objects can be combined with live action or graphics and image processing techniques can be used to produce a transformation of one person or object into another (i.e., morphing).

Image Processing

Although methods used in computer graphics and image processing overlap, the two areas are concerned with fundamentally different operations. Image processing applies techniques to modify or interpret existing pictures, such as satellite photographs and TV scans. Two principal applications of image processing are improving picture quality and machine perception of visual information, as used in robotics.

To apply image-processing methods, one first digitizes a photograph or any other picture into an image file. Then digital methods can be applied to rearrange picture parts, to increase colour separations, or to improve the quality of shading. These techniques are widely used in commercial art applications that involve retouching and rearranging of sections and of photographs and other art work. Similar methods are used to analyse satellite photos of the earth and photos of galaxies.

Medical science also makes wide use of image-processing techniques to enhance images through preprocessing steps, in tomography and in various simulations. Tomography is a technique of X-ray photography that allows cross-sectional views of physiological systems to be displayed. Both Computed X-ray Tomography (CT) and Position Emission Tomography (PET) use the projection method to reconstruct cross sections from digital data. These techniques are also used to monitor internal functions and show cross sections during surgery. Other medical imaging techniques include ultrasonic and nuclear medicine scanners. Ultrasonic uses high frequency sound waves, instead of X-rays.

Education and Training

Computer-generated methods of financial, physical, and economic systems are often used as educational aids for various purposes. Physiological systems, modeling of physical systems, population trends, equipment, etc., can be used to help trainees understand the operation of a system in an attractive manner. Special systems are designed for some training applications. Examples of such specialized systems are the simulators for practice session or training of aircraft pilots, ship captains, heavy-equipment operators, air traffic control personnel, etc. Some simulators have no video screen, for example, a flight simulator with only a control panel for instrument

flying. But most simulators provide graphics screens for visual operation. A keyboard is used to input parameters affecting the performance of an airplane or an environment, and the pen plotter is used to chart the path of an aircraft during a training session.

Visualization

Scientists, engineers, medical personnel, system analysts, and others often need to analyse large amounts of information to study the behaviour of certain processes. Numerical simulations can be carried out on supercomputers for frequently producing data files containing thousands and often millions of data values. Similarly, satellite cameras, radars and other sources are gathering large data files faster than they can be compiled. The process of scanning these large sets of data to determine trends and relationships is a deadly and unproductive process. But if graphics is applied to data, the converted form (visual form) is much more interesting. Producing graphical representations for engineering, scientific, and medical data sets and related processes is generally referred as scientific visualization. The term business visualization is used in connection with data sets related to industry, commerce and other nonscientific areas.

There are many different kinds of data sets, and effective visualization schemes that depend on the characteristics of the data. A collection data can contain scalar values, vectors, higher-order tensors, or any combination of these data types. Additional techniques include contour plots, graphs and charts, surface renderings, and visualizations of volume interiors. In addition, image processing techniques are combined with computer graphics to produce many data visualizations. Mathematicians, scientists, and others use visual techniques to analyse mathematical functions and processes or simply to produce interesting graphical representations.

1.2.2 Monitor Basics

To understand the fundamental operation of a simple LCD basically consists of a layer of liquid crystal which is sandwiched between two polarizing plates. The polarizers are aligned perpendicular to each other (one vertical and the other horizontal) so that the light incident on the first polarizer will be blocked by the second. Because a polarizer plate only passes photons (quanta of light) with their electric fields aligned parallel to the polarizing direction of that plate.

The LCD displays are addressed in a matrix fashion. Rows of matrix are defined by a thin layer of horizontal transparent conductors while columns are defined by another thin layer of vertical transparent conductors. The layers are placed between the LCD layer and the respective polarizer plate. The intersection of the two conductors defines a pixel position. This means that an individual LCD element is required for each display pixel unlike a CRT which may have several dot triads for each pixel.

The Liquid Crystal (LC) material is made up of long rod shaped crystalline molecules containing cyanobiphenyl units. The individual polar molecules in a nematic (spiral) LC layer are normally arranged in a spiral fashion such that the direction of polarization of polarized light passing through it is rotated by 90 degrees. Light from an internal source (backlight) enters the first polarizer (say horizontal)

NOTES

and is polarized accordingly (horizontally). As the light passes through the LC layer it is twisted 90 degrees (to align with the vertical) so that it is allowed to pass through the rear polarizer (vertical) and then reflect from the reflector behind the rear polarizer. The reflected light when reach the viewers eye travelling in the reverse direction, the LCD appears bright.

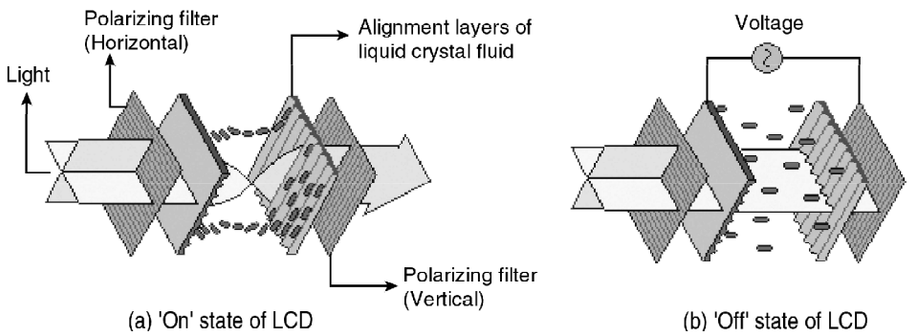


Fig. 1.3 *LCD: Fundamental Operations*

When an electric current is passed through the LCD layer the crystalline molecules align themselves parallel to the direction of light and thus do not produce polarizing effect. The light entering through the front polarizer is not allowed to pass through the rear polarizer due to mismatch of polarization direction. The result is zero reflection of light and the LCD appears black.

In a color LCD there are layers of three liquid crystal panels one on top of another. Each one is filled with a colored (red, green or blue) liquid crystal. Each one has its own set of horizontal and vertical conductors. Each layer absorbs an adjustable portion of just one color of the light passing through it. This is similar to how color images are printed. The principal advantage of this design is that it helps create as many screen pixels as intersections thus making higher resolution LCD panels possible. Each pixel comprises of three color cells or sub-pixel elements.

The image painting operation in LCD panels is a different from that of the CRT though both are of raster scan type. In a simple LCD panel, an entire line of screen pixels is illuminated at one time, then the next line and so on till the entire screen image is completed. Picture definitions are stored in a refresh buffer and the screen is refreshed typically at the rate of 60 frames per second. Once set, the screen pixels stay at fixed brightness until they are reset. The time required to set the brightness of a pixel is high compared to that of the CRT. Hence, the LCD panel pixels cannot be turned on or off anywhere near the rate at which pixels are painted on a CRT screen. Except the high quality *Active Matrix LCD panels*, others have trouble displaying movies which require quick refreshing.

Plasma Panel

Here a layer of gas (usually neon) is sandwiched between two glass plates. Thin vertical (column) strips of conductor run across one plate while horizontal (row) conductors run up and down the other plate. By applying high voltage to a pair of horizontal and vertical conductors, a small section of the gas (tiny neon bulb) at

the intersection of the conductors breaks down into glowing plasma of electrons and ions. Thus, in the array of gas bulbs each one can be set to an 'on' (glowing) state or 'off' state by adjusting voltages in the appropriate pair of conductors. Once set 'on' the bulbs remain in that state until explicitly made 'off' by momentarily reducing the voltage applied to the pair of conductors. Hence no refreshing is necessary.

Because of its excellent brightness, contrast and scalability to larger sizes, plasma panel is attractive. Researches are on to eliminate the color display limitation of such device at low production cost.

1.3 DISPLAY BASICS

The most prominent part in a personal computer (pc) is the display system that is responsible for graphic display. The display system may be attached with a PC to display character, picture and video output. Some of the common types of display systems available in the market are:

- Raster Scan Displays
- Random Scan Displays
- Direct View Storage Tube
- Flat Panel Displays
- Three-Dimensional Viewing Devices
- Stereoscopic and Virtual Reality System

The display systems are often referred to as *Video Monitor* or *Video Display* Unit (*VDU*). The most common video monitor that normally comes with a PC is the Raster Scan type. However, every display system has three basic parts – the *display adapter* that creates and holds the image information, the *monitor* which displays that information and the *cable* that carries the image data between the display adapter and the monitor.

Before we discuss the major display systems, let us first know about some basic terms.

Pixel

A pixel may be defined as the smallest size object or colour spot that can be displayed and addressed on a monitor. Any image that is displayed on the monitor is made up of thousands of such small pixels (also known as picture elements). The closely spaced pixels divide the image area into a compact and uniform two-dimensional grid of pixel lines and columns. Each pixel has a particular colour and brightness value. Though the size of a pixel depends mostly on the size of the electron beam within the CRT, pixels are too fine and close to each other to be perceptible by the human eye. The finer the pixels the more the number of pixels displayable on a monitor-screen. However, it should be remembered that the number of pixels in an image is fixed by the program that creates the image and not by the hardware that displays it.

NOTES

Resolution

There are two distinctly different terms, which are often confused. One is *image resolution* and the other is *screen resolution*. Strictly speaking, image resolution refers to the pixel spacing, i.e., the distance from one pixel to the next pixel. A typical PC monitor displays screen images with a resolution somewhere between 25 pixels per inch and 80 pixels per inch (ppi). In other words, resolution of an image refers to the total number of pixels along the entire height and width of the image. For example, a full-screen image with resolution 800 × 600 means that there are 800 columns of pixels, each column comprising 600 pixels, i.e., a total of 800 × 600 = 480000 pixels in the image area.

The internal surface of the monitor screen is coated with red, green and blue phosphor material that glows when struck by a stream of electrons. This coated material is arranged into an array of millions of tiny cells—red, green and blue, usually called *dots*. The *dot pitch* is the distance between adjacent sets (triads) of red, green and blue dots. It also refers to the shortest distance between any two dots of the same colour, i.e., from red-to-red, or, green-to-green like that. Usually, monitors are available with a dot pitch specification 0.25 mm to 0.40 mm. Each dot glows with a single pure colour (red, green or blue) and each glowing triad appears to our eye as a small spot of colour (a mixture of red, green and blue). Depending on the intensity of the red, green and blue colours, different colours results in different triads. The dot pitch of the monitor thus indicates how fine can be the coloured spots that make up the picture, though electron beam dia is an important factor in determining the spot size.

So, we understand that pixel is the smallest element of a displayed image, and dots (red, green and blue) are the smallest elements of a display surface (monitor screen). The dot pitch is the measure of screen resolution. The smaller the dot pitch, the higher the resolution, sharpness and detail of the image displayed.

In order to use different resolutions on a monitor, the monitor must support automatic changing of resolution modes. Originally, monitors were fixed at a particular resolution, but for most monitors today display resolution can be changed using software control. This lets you use higher or lower resolution depending on the need of your application. A higher resolution display allows you to see more information on your screen at a time and is particularly useful for operating systems such as Windows. However, the resolution of an image you see is a function of what the video card outputs and what the monitor is capable of displaying. Seeing a high resolution image, such as 1280 × 1024, requires both a video card capable of producing an image this dimension and a monitor capable of displaying it.

Image Resolution vs Dot Pitch

If the image resolution is more compared to the inherent resolution of the display device then the displayed image quality gets reduced. As the image has to fit in the limited resolution of the monitor, the screen pixels (comprising a red, a green and a blue dot) show the average colour and brightness of several adjacent image pixels. Only when the two resolutions match, the image is displayed perfectly and only then the monitor is used to its maximum capacity.

Aspect Ratio

The aspect ratio of the image is the ratio of the number of X pixels to the number of Y pixels. The standard aspect ratio for PCs is 4:3, and some resolutions even use a ratio of 5:4. Monitors are calibrated to this standard so that when you draw a circle, it appears to be a circle and not an ellipse. Displaying an image that uses an aspect ratio of 5:4 will cause the image to appear somewhat distorted. The only mainstream resolution that uses 5:4 is the high-resolution 1280 × 1024.

Table 1.1 **Common Resolutions, Respective Number of Pixels and Standard Aspect Ratios**

Resolution	Number of Pixels	Aspect Ratio
320 × 200	64,000	8:5
640 × 480	307,200	4:3
800 × 600	480,000	4:3
1024 × 768	786,432	4:3
1280 × 1024	1,310,720	5:4
1600 × 1200	1,920,000	4:3

1.3.1 Text Mode

Text mode is a computer display mode in which content is internally represented on a computer screen in terms of characters rather than individual pixels. The screen consists of a uniform rectangular grid of character cells, each of which contains one of the characters of a character set; at the same time, contrasted to All Points Addressable (APA) mode or other kinds of computer graphics modes. Text mode applications communicate with the user by using command-line interfaces and text user interfaces. Many character sets used in text mode applications also contain a limited set of predefined semi-graphical characters usable for drawing boxes and other rudimentary graphics, which can be used to highlight the content or to simulate widget or control interface objects found in GUI programs.

An important characteristic of text mode programs is that they assume monospace fonts, where every character has the same width on screen, which allows them to easily maintain the vertical alignment when displaying semi-graphical characters. This was an analogy of early mechanical printers which had fixed pitch. The output seen on the screen could be sent directly to the printer maintaining the same format.

Advantage of Text Mode

- The advantages of text modes as compared to graphics modes include lower memory consumption and faster screen manipulation. At the time text terminals were beginning to replace teleprinters in the 1970s, the extremely high cost of random access memory in that period made it exorbitantly expensive to install enough memory for a computer to simultaneously store the current value of every pixel on a screen, to form what would now be called a framebuffer. Early frame buffers were standalone devices which cost thousands of dollars, in addition to the expense of the advanced high-resolution displays to which they were connected.

NOTES

- Text mode avoids the problem of expensive memory by having dedicated display hardware re-render each line of text from characters into pixels with each scan of the screen by the cathode ray.
- Text mode is that it has relatively low bandwidth requirements in remote terminal use. Thus, a text mode remote terminal can necessarily update the screen much faster than a graphics mode remote terminal linked to the same amount of bandwidth (and in turn will seem more responsive), since the remote server may only need to transmit a few dozen bytes for each screen update in text mode, as opposed to complex raster graphics remote procedure calls that may require the transmission and rendering of entire bitmaps.

1.3.2 Graphics Mode

Graphics mode is a way of displaying images on a computer screen or other graphics device such that the basic unit is the pixel. Lines and characters on the screen are drawn pixel by pixel. The resolution and complexity of the image depends on how many pixels there are in total, and how many bits are assigned to each pixel. The more bits per pixel, the more different colors or shades of gray. A single graphics device can operate in a number of different graphics modes with different resolutions and color selections. A common mode for a desktop PC would be 1024 by 768 pixels with 256 different colors chosen from a much larger number available for each pixel is also known as text mode and computer graphics.

Graphics mode is a computer display mode that generates image using pixels. Today, most users operate their computer in a graphics mode opposed to a text mode or command line environment.

1.4 ADAPTERS AND DISPLAYS

The display adapter circuitry (on video card or motherboard) in a raster graphics system typically employs a special purpose processor called *Display Processor* or *Graphics Controller* or *Display Coprocessor* which is connected as an I/O peripheral to the CPU (refer Figure 1.19). Such processors assist the CPU in scan-converting the output primitives (line, circle, arc, etc.) into bitmaps in frame buffer and also perform raster operations of moving, copying and modifying pixels or block of pixels. The output circuitry also includes another specialized hardware called *Video Controller* which actually drives the CRT and produces the display on the screen.

The monitor is connected to the display adapter circuitry through a cable with 15-pin connectors. Inside the cable are three analog signals carrying brightness information in parallel for the three color components of each pixel. The cable also contains two digital signal lines for vertical and horizontal drive signals and three digital signal lines which carry specific information about the monitor to the display adapter.

The video controller is the output circuitry which generates the horizontal and vertical drive signals so that the monitor can sweep its beam across the screen during raster scan (refer Figure 1.5). Memory reference addresses are generated

in synchrony with the raster scan, and the contents of the memory are used to control the CRT beam intensity or color. Two registers (X register and Y register) are used to store the coordinates of the screen pixels. Assume that the y values of the adjacent scan lines increase by 1 in upward direction starting from 0 at the bottom of the screen to y_{max} at the top. And along each scan line the screen pixel positions or x values are incremented by 1 from 0 at the leftmost position to x_{max} at the rightmost position. The origin is at the lower left corner of the screen as in a standard Cartesian coordinate system. At the start of a refresh cycle, the X register is set to 0 and the Y register is set to y_{max}. This (x, y) address is translated into a memory address of frame buffer where the color value for this pixel position is stored. The controller retrieves this color value (a binary number) from the frame buffer, breaks it up into three parts and sends each part to a separate Digital-to-Analog Converter (DAC). After conversion, the DAC puts the proportional analog voltage signals on the three analog output wires going to the monitor. These voltages in turn control the intensity of three electron beams that are focused at the (x, y) screen position by the horizontal and vertical drive signals.

This process is repeated for each pixel along the top scan line, each time incrementing the X register by 1. As pixels on the first scan line are generated, the X register is incremented through x_{max}. Then the X register is reset to 0 and the Y register is decremented by 1 to access the next scan line. Pixels along this scan line are then processed and the procedure is repeated for each successive scan line until pixels on the last scan line ($y = 0$) are generated. For a display system employing a color look-up table, however, frame buffer value is not directly used to control the CRT beam intensity. It is used as an index to find the true pixel-color value from the look-up table. This look-up operation is done for each pixel on each display cycle.

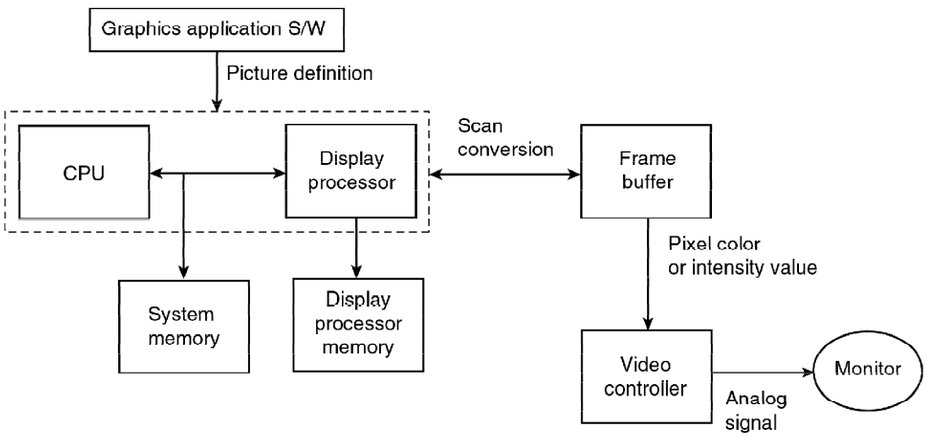


Fig. 1.4 *General Architecture of a Raster Display System*

As the time available to display or refresh a single pixel in the screen is too less (in the order of a few nanoseconds), accessing the frame buffer every time for reading each pixel intensity value would consume more time than what is allowed. Therefore multiple adjacent pixel values are fetched to the frame buffer in a single access and stored in a register. After every allowable time gap (as dictated by the refresh rate and resolution) one pixel value is shifted out from the register to control

NOTES

the beam intensity for that pixel. The procedure is repeated with the next block of pixels, and so on. Thus the whole group of pixels will be processed.

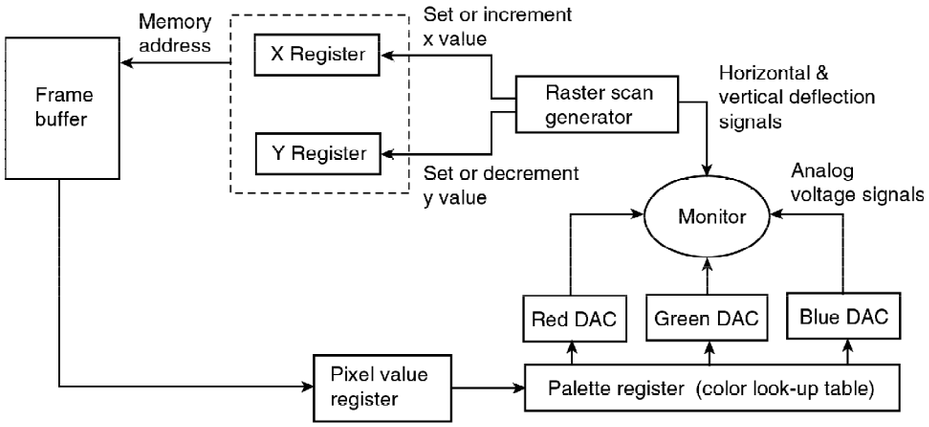


Fig. 1.5 *Logical Operations of the Video Controller*

1.4.1 Monochrome Display Adapter (MDA)

Monochrome Display Adapter (MDA, also MDA card, Monochrome Display and Printer Adapter, MDPA) is standard video display card and computer display standard for the PC introduced in 1981. The MDA does not have any pixel-addressable graphics modes, only a single monochrome text mode which can display 80 columns by 25 lines of high resolution text characters or symbols useful for drawing forms.

The MDA was based on the display system, and was intended to support business and word processing use with its sharp, high-resolution characters. Each character is rendered in a box of 9×14 pixels, of which 7×11 depicts the character itself and the other pixels provide space between character columns and lines. Some characters, such as the lowercase 'm', are rendered eight pixels across.

The theoretical total screen display resolution of the MDA is 720×350 pixels, if the dimensions of all character cells are added up, but the MDA cannot address individual pixels to take full advantage of this resolution. Each character cell can be set to one of 256 bitmap characters stored in ROM on the card, and this character set cannot be altered from the built-in hardware code.

1.4.2 Color Graphics Adapter (CGA)

The Color Graphics Adapter (CGA), originally also called the Color/Graphics Adapter or /Graphics Monitor Adapter. The CGA graphics card was built around the Motorola 6845 display controller, came with 16 kilobytes of video memory built in, and featured several graphics and text modes. The highest display resolution of any mode was 640×200, and the highest color depth supported was 4-bit (16 colors).

The CGA card could be connected either to a direct-drive CRT monitor using a 4-bit digital (TTL) RGBI interface, such as the color display, or to an NTSC-compatible television or composite video monitor via an RCA connector. The RCA connector provided only baseband video, so to connect the CGA card to a television set without a composite video input required a separate RF modulator.

Color display was not available, customers could either use the composite output, or the direct-drive output with available third-party monitors that supported the RGBI format and scan rate. Some third-party displays lacked the intensity input, reducing the number of available colors to eight, and many also lacked unique circuitry which rendered the dark-yellow color as brown, so any software which used brown would be displayed incorrectly.

1.4.3 Hercules Graphics Card

The Hercules Graphics Card (HGC) is a computer graphics controller made by Hercules Computer Technology, Inc. MDA display standard with a bitmapped graphics mode. This allows the HGC to offer both high-quality text and graphics from a single card. The HGC was very popular, and became a widely supported de facto display standard onIBM PC compatibles. The HGC standard was used long after more technically capable systems had entered the market, especially on dual-monitor setups.

The Hercules Graphics Card (HGC) is a piece of hardware which gives IBM personal computers display capabilities superior than their default hardware. The card was developed in 1982 by Van Suwannukul who wanted to be able to display the Thai alphabet on a computer which wasn't possible at lower resolutions or using the stock IBM Monochrome Display Adapter.

The Hercules Graphics Card can operate in two different modes, text mode and graphics mode. In text mode, the card is compatible with the IBM Monochrome Display Adapter. It supports 80x25 characters on the screen. Each character is 9x14 pixels, but only the center 7x11 are used for a character. The total screen resolution is 720x350, but you cannot modify the screen at the pixel level, only the character level. On a standard 4:3 ratio monitor, the pixel ratio is 1:1.55.

In graphics mode, every pixel on the screen is independently addressable allowing for more complex graphics. The resolution is 720x348 (two pixels are lost since the dimensions must be divisible by four technical reasons). The pixel ratio is again 1:1.55 on a standard 4:3 display.

1.4.4 Enhanced Graphics Adapter

The Enhanced Graphics Adapter (EGA) is video adapter for IBM PC graphics adapter and facto computer display standard from 1984 that superseded the CGA standard introduced with the original IBM PC, and was itself superseded by the VGA standard in 1987. In addition to the original EGA card manufactured by IBM, many compatible third-party cards were manufactured, and EGA graphics modes continued to be supported by VGA and later standards. This means it has the electronic circuits your computer needs to display images on the screen.

Most new graphics and windows programs will function with an EGA. But do not buy one new-get a VGA instead, or better yet, a Super VGA. In graphics mode, EGAS can display a maximum of 640 by 350 pixels (dots of colored light) on the screen. That resolution is barely acceptable, and it produces squashed looking images (squares look like rectangles). EGAS can display text too, but EGA text is a little too fuzzy to read for long periods of time.

NOTES

1.4.5 Professional Graphics Adapter

Professional Graphics Controller (PGC, often called Professional Graphics Adapter and sometimes Professional Graphics Array) is a graphics card manufactured by IBM for PCs Personal Computers. It consists of three interconnected PCBs, and contains its own processor and memory. The PGC was, at the time of its release, the most advanced graphics card for the IBM XT and aimed for tasks, such as CAD.

Introduced in 1984,the Professional Graphics Controller offered a maximum resolution of 640×480 with 256 colors at a refresh rate of 60 hertz, a higher resolution and color depth than CGA and EGA supported. This mode is not BIOS-supported. It was intended for the computer-aided design market and included 320 kB of display RAM and an on-board Intel 8088 microprocessor. The 8088 ran software routines, such as 'draw polygon' and 'fill area' from an on-board 64 kB ROM so that the host CPU did not need to load and run these routines itself. While never widespread in consumer-class personal computers, its US$4,290 list price compared favorably to US$50,000 dedicated CAD workstations of the time model. It was discontinued in 1987 with the arrival of VGA and 8514.

1.5 DIGITAL VS ANALOG

Data communication and networks deal with data or information transmission. Data can be represented in many ways such as a human voice, a bunch of numbers, images, text and sounds, etc. There are two ways to communicate, display, store or manipulate information, as follows:

- Analog
- Digital

In the analog form of electronic communication, information is represented as a continuous electromagnetic wave form as shown in Figure 1.6. Digital communication represents information in binary form through a series of discrete pulses as shown in Figure 1.7.

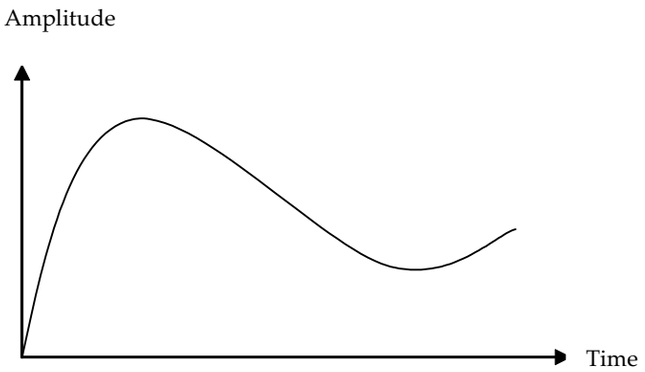


Fig. 1.6 *Representation of Analog Signals*

NOTES

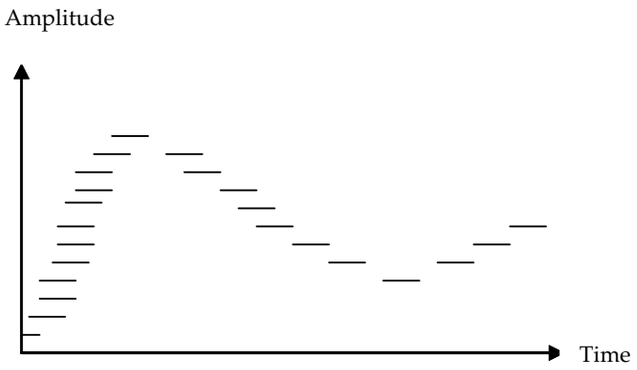


Fig. 1.7 *Representation of Digital Signals*

Analog signal

Analog is best explained by the transmission of such signals as human speech or sound, over an electrified copper wire. In its native form, human speech is an oscillatory disturbance in the air as shown in Figure 1.6, which varies in terms of its volume, or power (amplitude) and its pitch or tone (frequency). Analog signals are therefore defined as continuous electrical signals varying in time as shown in Figure 1.8. Analogous variations in radio or electrical waves are created in order to transmit the analog information signal for video or audio or both over a network from a transmitter (TV station or CATV source) to a receiver (TV set, computer connected with antenna). At the receiving end, an approximation (analog) of the original information is presented. Information that is analog in its native form (image and audio) can vary continuously in terms of intensity (brightness or volume) and frequency (color or tone) as shown in Figures 1.6 and 1.7. These variations in the native information stream are translated, in an analog electrical network, into variations in the frequency and amplitude of the carrier signal. In other words, the carrier signal is modulated (varied) in order to create an analog of the original information stream.

The electromagnetic sinusoidal waveform or sine wave as shown in Figure 1.8 can be varied in amplitude at a fixed frequency, using Amplitude Modulation (AM). Alternatively, the frequency of the sine wave can be varied at constant amplitude, using Frequency Modulation (FM). Additionally, both amplitude and frequency can be modulated simultaneously. Figures 1.9 and 1.10 represent a sinusoidal waveform in amplitude and frequency form. The example of analog signal in the field of data communication is telephone voice signal in which the intensity of the voice causes electric current variations. At the receiving end, the signal is reproduced in the same proportion.

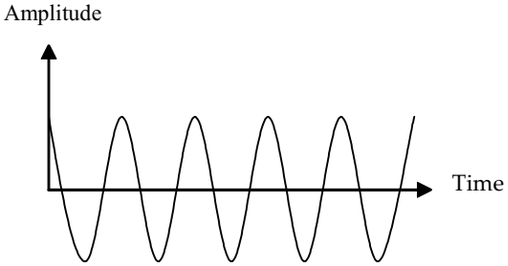


Fig. 1.8 *Waveform in the Form of Sine Wave*

NOTES

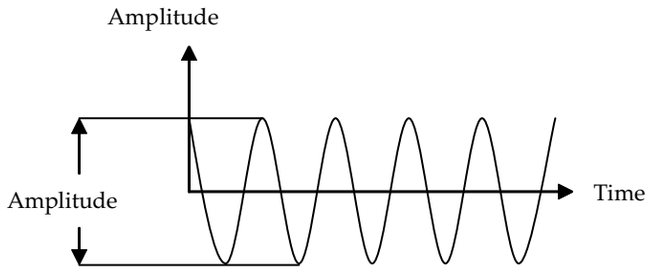


Fig. 1.9 *Amplitude*

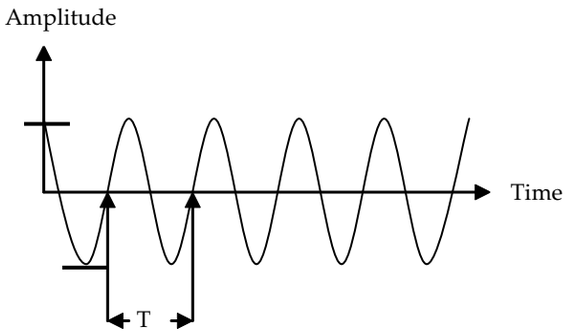


Fig. 1.10 *Frequency Representation*

Voice: A voice grade channel is approximately 4,000 Hz, or 4 kHz. Approximately 3.3 kHz (200 Hz to 3,500 Hz) is used for the voice signal itself. The remaining bandwidth is used for the purpose of network signalling and control in order to maintain separation between information channels. While human speech transmission and reception encompasses a much wider range of frequencies, 3.3 kHz is considered to be quite satisfactory and cost-effective. Band-limiting filters are used in carrier networks to constrain the amount of bandwidth provided for a voice application.

Video: A CATV video channel is approximately 6 MHz. Approximately, 4.5 MHz is used for information transmission, while the balance is used for guard bands to separate the various adjacent channels using the common, analog coaxial cable system.

Digital signal

Computers are digital in nature. Computers communicate, store and process information in binary form, i.e., in the combination of 1s and 0s, which has specific meaning in computer language. A binary digit (bit) is an individual 1 or 0. Multiple bit streams are used in a computer network. The computer systems communicate in binary mode through variations in electrical voltage. The digital signals that are non-continuous change in individual steps consisting of digits or pulses with discrete values or levels. The value of each pulse is uniform but there is an abrupt change from one digit to the next. They have two amplitude levels, which are specified as one of two possibilities like 1 or 0, high or low, true or false and so on. In other words, the digital signalling, in an electrical network, involves a signal which varies in voltage to represent one of two discrete and well-defined states as depicted in Figure 1.11 such as either a positive (+) voltage and a null or zero (0) voltage

(unipolar) or a positive (+) or a negative (–) voltage (bipolar).

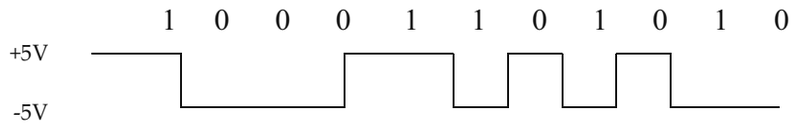


Fig. 1.11 *Binary Representation Forming Digital Signal*

NOTES

Data and Signals

Signals

Information exchange is an essential part of communication. It may be exchange of information among users or equipment in the communication system. In the communication context, signalling refers to the exchange of information between components required to provide and maintain data communication service. In case of PSTN, signalling between a telephone user and the telephone network may include dialling digits, providing dial tone, accessing a voice mailbox and sending a call-waiting tone etc. Looking at networking, perspectives, it is transmission of service information such as addresses, type of service etc., between nodes and/or terminals of a network. In other words, it is a process of exchanging and generating information between components of a telecommunications system to establish, release, or monitor connections (call handling functions) and to control related network and system operations (other functions).

Signalling System7 (SS7)

Signalling System 7 is the protocol designed for public switched telephone system for providing services and setting up calls.The various value-added features such as providing intelligence to PSTN services come under the service of SS7. Earlier the same physical path was used for both the call-control signalling and the actual connected call. This is called in-band signalling technique. This method of signalling was inefficient and replaced by out-of-band or common-channel signalling techniques. Out-of-band signalling performs its job by utilizing two networks in one. As we know that in PSTN, our voice and data is carried over circuit-switched network. It provides a physical path between the destination and source. The other one is the signalling network, which carries the call control traffic. It is a packet-switched network using a common channel switching protocol.

Functions of SS7

- It controls the network.
- The SS7 network sets up and tears down the call.
- It handles all the routing decisions and supports all telephony services including local number portability (LNP), remote network management, called ID and forwarding.

In order to accomplish the above functions, SS7 uses voice switches, which are known as Service Switching Points (SSPs). They handle the SS7 control network as well as the user circuit-switched network. Basically, the SS7 control network tells the switching office which paths to establish over the circuit-switched network. SSPs also query Service Control Point (SCP) databases using packet switches called

NOTES

Signal Transfer Points (STPs). The STPs route SS7 control packets across the signaling network. The concept of SSP, STP and SCP has been illustrated in Figure 1.12.

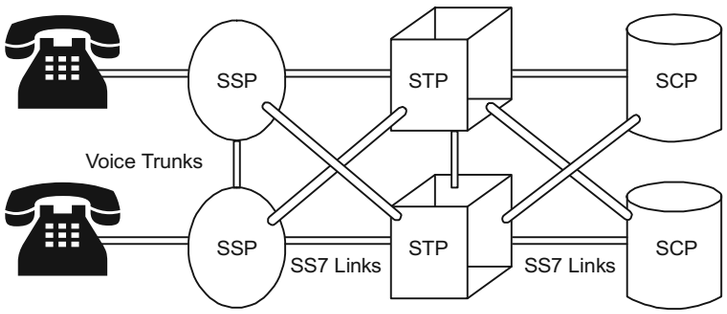


Fig. 1.12 *SS7 Signaling Points*

1.5.1 Video Graphics Array

VGA stands for video graphics array, is currently the most popular standard for PC screen display equipment. Technically, a VGA is a type of video adapter (circuitry in the computer that controls the screen). IBM developed the VGA for its PS/2 line of computers (the name 'Video Graphics Array' is an IBM trademark), but loads of other manufacturers make VGA add-in boards (that plug into a slot in the pc) and VGA chips (in some pcs, these VGA chips are built right into the main part of the computer, the motherboard). A VGA monitor is a monitor that works with a VGA adapter.

A standard VGA system displays up to 640x 480 pixels (little dots) on the screen, with up to 16 different colors at a time. In lower resolution, 320x 200 pixels, the screen can show up to 256 colors at once. These specifications are much better than the older video adapter standards, the CGA and EGA, but they are not good enough for many people. If you are buying a new system or replacing an older video adapter, make sure you get a 'Super VGA' adapter, which can handle higher resolutions (800x 600 or higher) and many more colors. The higher the resolution and the more colors you have to work with, the slower the display will function, and the more memory you are need on the card.

The EGA and CGA monitors, VGA monitors are analog devices, meaning they can display an infinite range of colors. When you are shopping for a VGA monitor, keep several points in mind. First, if you want to use higher resolutions than the VGA standard of 640x 480, you need a multiscan monitor-a plain VGA monitor will not work at higher resolutions. Second, some VGA monitors give a sharper image than others. Partly, this depends on the dot Pitcha monitor with a smaller dot-pitch will have better image clarity than one with a larger dot-pitch.

A VGA monitor requires an interface card and a cable. You need to know how much memory is on the card. You may want to add more memory, especially if you plan to create and use complex graphic or photographic images. The VGA is the current standard right now in monitors, and such as usually the most readily available.

NOTES

1.5.2 Super VGA

Super VGA (SVGA) is a broad term that covers a wide range of computer display standards that extended IBM's VGA specification.In the late 1980s, after the release of IBM's VGA, third-party manufacturers began making graphics cards based on its specifications with extended capabilities. As these cards grew in popularity they began to be referred to as 'Super VGA.'

Super VGA cards broke compatibility with the IBM VGA standard, requiring software developers to provide specific display drivers and implementations for each card their software could operate on.

Initially the heavy restrictions this placed on software developers slowed the uptake of Super VGA cards, which motivated VESA to produce a unifying standard, the VESA BIOS Extensions, to provide a common software interface to all cards implementing the VBE specification. SVGA uses a VGA connector, the same DE-15 as the original standard, and otherwise operates over the same cabling and interfaces as VGA.

Check Your Progress

1. What is computer graphics?
2. List out some common types of display systems.
3. What happens when the image resolution is more than the inherent resolution of the display device?
4. Define the term text mode.
5. State about the Monochrome Display Adapter (MDA).
6. What is Video Graphics Array (VGA)?

1.6 REFRESH CATHODE-RAY TUBES

A CRT is similar to a big vacuum glass bottle. It contains three electron guns that squirt out focused beam of electrons, deflection apparatus (magnetic or electrostatic), that deflects these beams both up and down and sidewise and a phosphor-coated screen upon which these beams impinge. The vacuum is necessary to let those electron beams travel across the tube without running into air molecules that could absorb or scatter them.

The primary component in an electron gun is a *cathode* (negatively charged) encapsulated by a metal cylinder known as the *control grid*. A heating element inside the cathode causes the cathode to be heated up as current is passed, and as a result electrons 'boil-off' from the hot cathode surface. These electrons are accelerated towards the CRT screen by a high positive voltage applied near the screen or by an *accelerating anode*. If allowed to continue uninterrupted, the naturally diverging electrons would simply flood the entire screen. The cloud of electrons is forced to converge to a small spot as it touches the CRT screen by a *focusing system* using an electrostatic or magnetic field. Just as an optical lens focuses a beam of light at a particular focal distance, a positively charged metal cylinder focuses the electron beam passing through it on the centre of the CRT

NOTES

screen. A pair of magnetic *deflection coils* mounted outside the CRT envelope deflects the concentrated electron beam to converge at different points on the screen in the process of scanning. Horizontal deflection is obtained by one pair of coils and vertical deflection by the other pair, and the deflection amount is controlled by adjusting the current passing through the coils. When the electron beam is deflected away from the centre of the screen, the point of convergence tends to fall behind the screen resulting in blurred (defocused) display near the screen edges. In the high-end display devices, this problem is eliminated by a mechanism which dynamically adjusts the beam focus at different points on the screen.

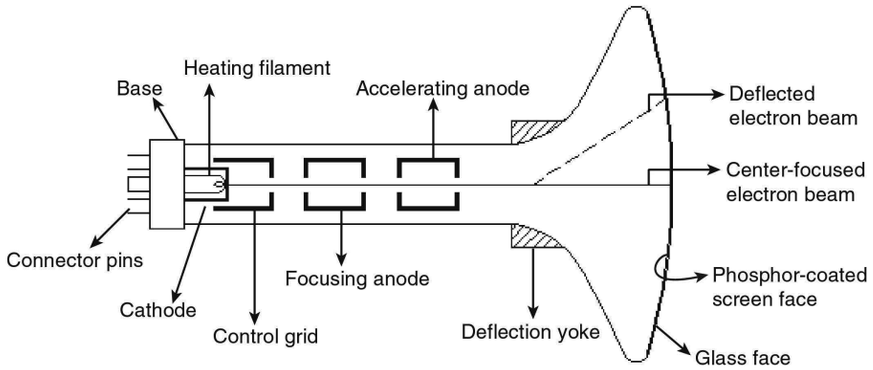


Fig. 1.13 *Schematic Diagram of a Raster Scan CRT*

When the electron beam converges onto a point on the phosphor-coated face of the CRT screen, the phosphor dots absorb some of the kinetic energy from the electrons. This causes the electrons in the phosphor atoms to jump to higher energy orbits. After a short time these excited electrons drop back to their earlier stable state, releasing their extra energy as small quantum of light energy. As long as these excited electrons return back to their stable state, phosphor continues to glow (*phosphorescence*) but gradually loses brightness. The time between the removal of excitation and the moment when phosphorescence has decayed to ten per cent of the initial brightness is termed as *persistence* of phosphor. The brightness of the light emitted by phosphor depends on the intensity with which the electron beam (number of electrons) strikes the phosphor. The intensity of the beam can be regulated by applying measured negative voltage at the control grid. Corresponding to a zero value in the frame buffer, a high negative voltage is applied in the control grid, which, in turn, will shut off the electron beam by repelling the electrons and stopping them from coming out of the gun and hitting the screen. The corresponding points on the screen will remain black. Similarly a bright white spot can be created at a particular point by minimizing the negative voltage at the control grid of the three electron guns when they are directed to that point by the deflection mechanism.

Apart from brightness, the size of the illuminated spot created on the screen varies directly with the intensity of the electron beam. As the intensity or number of electrons in the beam increases, the beam diameter and spot size also increase. Also, the highly excited bright phosphor dots tend to spread the excitation to the neighbouring dot, thereby further increasing the spot size. Therefore, the total number of distinguishable spots (pixels) that can be created on the screen depends

NOTES

on the individual spot size. The lower the spot size, the higher the image resolution.

In a monochrome CRT, there is only one electron gun, whereas in a colour CRT there are three electron guns, each controlling the display of red, green and blue light respectively. Unlike the screen of a monochrome CRT, which has a uniform coating of phosphor, the colour CRT has three colour-phosphor dots (dot triad) – red, green and blue – at each point on the screen surface. When struck by electron beam the red dot emits red light, the green dot emits green light and the blue dot emits blue light. Each triad is arranged in a triangular pattern, as are the three electron guns. The beam deflection arrangement allows all the three beams to be deflected at the same time to form a raster scan pattern. There are separate video streams for each RGB (red, green and blue) colour component which drive the electron guns to create different intensities of RGB colours at each point on the screen. To ensure that the electron beam emitted from individual electron guns strikes only the correct phosphor dots (e.g., the electron gun for red colour excites only the red phosphor dot), a *shadow mask* is used just before the phosphor screen. The mask is a fine metal sheet with a regular array of holes punched in it. It is so aligned that as the set of three beams sweeps across the shadow mask they converge and intersect at the holes and then hit the correct phosphor dot; the beams are prevented or masked from intersecting other two dots of the triad. Thus, different intensities can be set for each dot in a triad and a small colour spot is produced on the screen as a result.

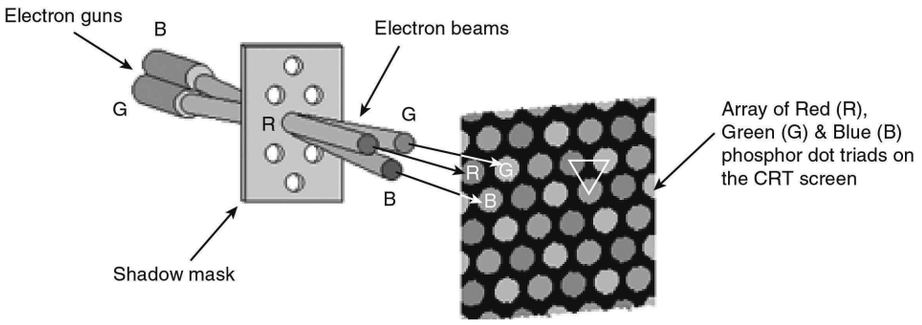


Fig. 1.14 *Electron Guns passing through CRT*

An alternative way to accomplish the masking function is adopted by some CRTs. Instead of a shadow mask, they use an *aperture grill*. In this system, the metal mesh is replaced by hundreds of fine metal strips that run vertically from the top of the screen to the bottom. In these CRTs, the electron guns are placed side-by-side (not in a triangular fashion).The gaps between the metal wires allow the three electron beams to illuminate the adjacent columns of coloured phosphor which are arranged in alternating stripes of red, green and blue. This configuration allows the phosphor stripes to be placed closer together than conventional dot triads. The fine vertical wires block less of the electron beam than ordinary shadow masks, resulting in a brighter and sharper image. This design is the most common in Sony's popular *Trinitron*. Trinitron monitors are curved only in the horizontal plane but are flat vertically.

For TV sets and monitors, the diagonal dimension is stated as the size. As a portion at the edge of the picture tube is covered by the case, the actual viewable portion of the tube measures only 19 inches diagonally. For standard monitors, the height is about three-fourths of the width. For a 19-inch monitor, the image width will be 15 inches and the height will be 11 inches.

NOTES

1.7 RASTER SCAN DISPLAYS

The raster scan display basically employs a CRT or LCD Panel for display. The CRT works just like the picture tube of a television set. Its viewing surface is coated with a layer of arrayed phosphor dots. At the back of the CRT is a set of electron guns (cathodes) which produce a controlled stream of electrons (electron beam). The phosphor material emits light when struck by these high-energy electrons. The frequency and intensity of the light emitted depend on the type of phosphor material used and energy of the electrons. To produce a picture on the screen, these directed electron beams start at the top of the screen and scan rapidly from left to right along the row of phosphor dots. They return to the leftmost position one line down and scan again, and repeat this to cover the entire screen. The return of the beam direction to the leftmost position one line down is called *horizontal retrace* during which the electron flow is shut off. In performing this scanning or sweeping type motion, the electron guns are controlled by the video data stream coming into the monitor from the video card, which varies the intensity of the electron beam at each position on the screen. The instantaneous control of the intensity of the electron beam at each dot is what controls the colour and brightness of each pixel on the screen. All this happens quickly, and the entire screen is drawn in a fraction (say, 1/60th) of a second.

An image in raster scan display is basically composed of a set of dots and lines; lines are displayed by making those dots bright (with desired colour) which lie as close as possible to the shortest path between the endpoints of a line.

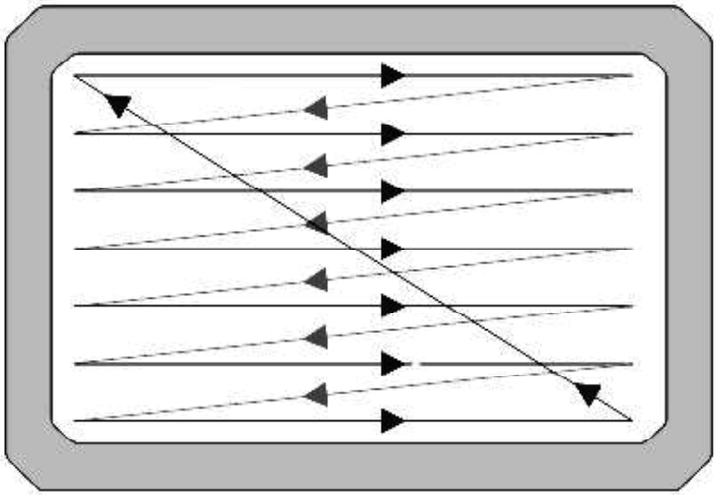


Fig. 1.15 *Raster Scan Display*

1.7.1 Random Scan Displays

Basically there are two types of CRT's – raster scan type and random scan type. The main difference between the two is the technique with which the image is generated on the phosphor coated CRT screen. In the raster scan method, the electron beam sweeps the entire screen in the same way you would write a full page text in a notebook, word by word, character by character, from left to right,

and from top to bottom. In the random scan technique, on the other hand, the electron beam is directed straightway to the particular point(s) of the screen where the image is to be produced. It generates the image by drawing a set of random straight lines much in the same way one might move a pencil over a piece of paper to draw an image – drawing strokes from one point to another, one line at a time. This is why this technique is also referred as *vector drawing* or *stroke writing* or *calligraphic display*.

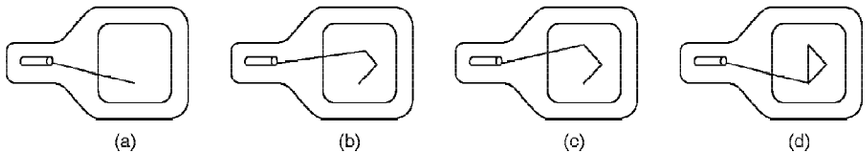


Fig. 1.16 *Drawing a Triangle on a Random Scan Display*

There are of course no bit planes containing mapped pixel values in vector system. Instead, the display buffer memory stores a set of line drawing commands along with end point coordinates in a *display list* or *display program* created by a graphics package. The Display Processing Unit (DPU) executes each command during every refresh cycle and feeds the *vector generator* with digital *x*, *y* and *Dx*, *Dy* values. The vector generator converts the digital signals into equivalent analog deflection voltages. This causes the electron beam to move to the start point or from the start point to the end point of a line or vector. Thus, the beam sweep does not follow any fixed pattern; the direction is arbitrary as dictated by the display commands. When the beam focus must be moved from the end of one stroke to the beginning of the other, the beam intensity is set to 0.

Though the vector-drawn images lack in depth and real-like colour precision, the random displays can work at higher resolutions than the raster displays. The images are sharp and have smooth edges unlike the jagged edges and lines on raster displays.

A Computer Graphics Hardware System (General)

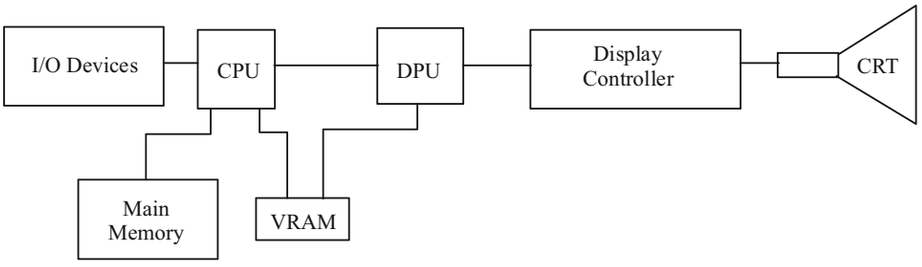


Fig. 1.17 *Block Diagram of Random Scan Display*

1.8 FLAT-PANEL DISPLAY

To satisfy the need of a compact portable monitor , modern technology has gifted us with *Liquid crystal Display* (*LCD*) *panel, plasma display panel, Light Emitting Diode* (*LED*) *panel* and *thin CRT*. These display devices are smaller, lighter and specifically thinner than the conventional CRT and thus are termed as

NOTES

Flat Panel Display (FPD). FPD in general and LCD panels in particular are most suitable for laptops (notebook) but are expensive to produce. Though hardware prices are coming down sharply, the cost of the LCD or plasma monitors is still too high to compete with CRT monitors in desktop applications. However, the thin CRT, is comparatively economical. To produce a thin CRT, the tube length of a normal CRT is reduced by bending it in the middle. The deflection apparatus is modified so that electron beams can be bent through 90 degrees to focus on the screen and at the same time can be steered up and down and across the screen.

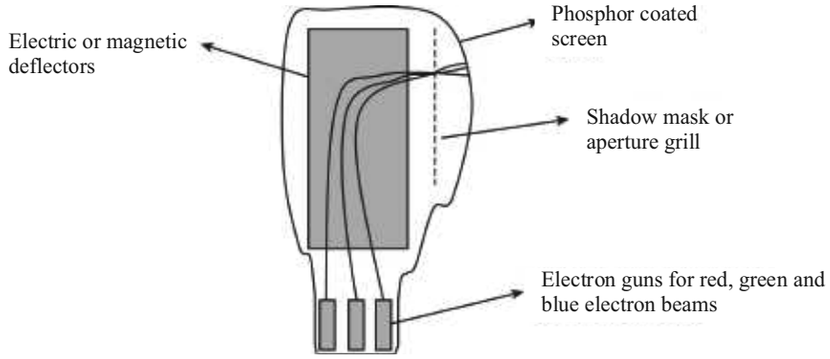


Fig. 1.18 *Thin CRT*

LCD

To understand the fundamental operations of a simple LCD, a model is shown in Figure 1.19. LCD basically consists of a layer of liquid crystal, sandwiched between two polarizing plates. The polarizers are aligned perpendicular to each other (one vertical and the other horizontal), so that the light incident on the first polarizer will be blocked by the second. Because a polarizer plate only passes photons (quanta of light) with their electric fields aligned parallel to the polarizing direction of that plate.

The LCD displays are addressed in a matrix fashion. Rows of matrix are defined by a thin layer of horizontal transparent conductors, while columns are defined by another thin layer of vertical transparent conductors; the layers are placed between the LCD layer and the respective polarizer plate. The intersection of the two conductors defines a pixel position. This means that an individual LCD element is required for each display pixel, unlike a CRT which may have several dot triads for each pixel.

The liquid crystal material is made up of long rod-shaped crystalline molecules containing cyanobiphenyl units. The individual polar molecules in a nematic (spiral) LC layer are normally arranged in a spiral fashion such that the direction of polarization of polarized light passing through it is rotated by 90 degrees. Light from an internal source (backlight)* enters the first polarizer (say horizontal) and is polarized accordingly (horizontally). As the light passes through the LC layer, it is twisted 90 degrees (to align with the vertical) so that it is allowed to pass through the rear polarizer (vertical) and then reflect from the reflector behind the rear polarizer. When the reflected light reaches the viewer's eye travelling in the reverse direction, the LCD appears bright.

NOTES

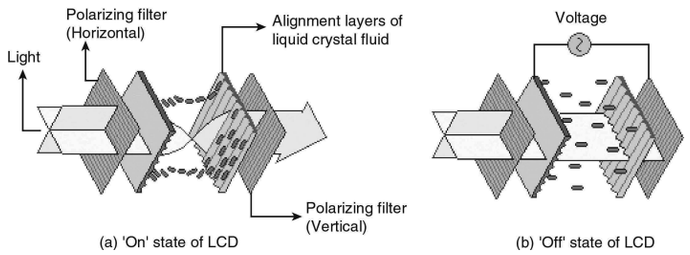


Fig. 1.19 *An LCD Model*

When an electric current is passed through the LCD layer, the crystalline molecules align themselves parallel to the direction of light and thus have no polarizing effect. The light entering through the front polarizer is not allowed to pass through the rear polarizer due to mismatch of polarization direction. The result is zero reflection of light and the LCD appears black.

In a colour LCD, there are layers of three liquid crystal panels, one on top of another. Each one is filled with a coloured (red, green or blue) liquid crystal. Each one has its own set of horizontal and vertical conductors. Each layer absorbs an adjustable portion of just one color of the light passing through it. This is similar to how colour images are printed. The principal advantage of this design is that it helps create as many screen pixels as intersections, thus making the higher-resolution LCD panels possible. In the true sense, each pixel comprises three colour cells or sub-pixel elements.

The image painting operation in LCD panels is a different from that in the CRT, though both are of the raster scan type. In a simple LCD panel, an entire line of screen pixels is illuminated at one time, then the next line, and so on till the entire screen image is completed. Picture definitions are stored in a refresh buffer and the screen is refreshed typically at the rate of 60 frames per second. Once set, the screen pixels stay at fixed brightness until they are reset. The time required to set the brightness of a pixel is high compared to that of the CRT. This is why LCD panel pixels cannot be turned on or off anywhere near the rate at which pixels are painted on a CRT screen. Except the high quality *Active Matrix LCD panels**, others have trouble displaying movies, which require quick refreshing.

1.9 RASTER SCAN SYSTEMS

A raster scan or raster scanning is a design or pattern of the image detection and its reconstruction on computer screen and is stored in computer memory. The word 'raster' has its origin from the Latin word *rastrum* (a rake), which is further derived from the word *radere,* which means to scrape. When drawn straight, the pattern formed by the lines of a rake forms the parallel lines of a raster. Thus, raster is created through line-by-line scanning. In it, one line is created at a time, making the scanning process systematic. The various raster scan systems are computer monitors, printers, televisions, plotters, fire-control radars, etc. The advantages of raster scan are as follows:

- Raster displays are of less resolution.
- The lines produced are zigzag because the plotted values are discrete.

- High-degree pragmatism can be achieved in an image using the advanced shading and hidden surface technique.
- Memory costs are decreasing.

NOTES

Computer Screen

In a raster scan, the image is broken into a sequence of strips, generally horizontally, which are termed as 'scan lines'. The analog signal form is used to transmit each scan line, because it is easily read by the detector, which further divides it into distinct pixels to be processed by a computer system. When any image is displayed on computer screen , each scan line is curved back to a line across the computer screen. After displaying a scan line, the current location of the scan line is advanced downward across the image , which is termed as vertical scanning. Then the next scan line in sequence is detected, transmitted, stored and displayed. Such an ordering of pixels in rows is termed as the raster scan ordering. Figure 1.20 shows the various scanning patterns:

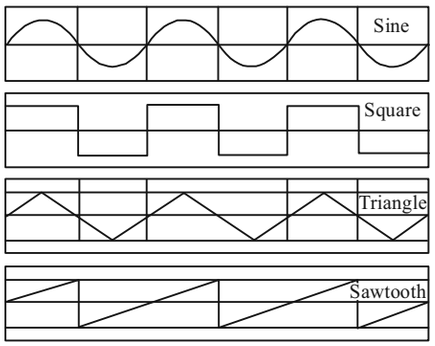


Fig. 1.20 *Scanning Patterns*

In a raster scanning process, the beam horizontally sweeps from left to right at a stable rate and the blank quickly moves back to the left, where it again turns back to sweep out the next line. Here, sweep is the movement of the beam(s) and the circuit thus created is termed as sweep circuit.

The display in a CRT is horizontal due to deflecting components of the magnetic field formed by the deflection. As a result of this, the unblanked beams scan 'forward' from left to right at a constant rate. The data for successive pixels for primary colours moves to the digital-to-analog converters at the pixel clock rate. The pixel data is digital in the case of flat-panel displays.

Printers

Computer printers generally produce the output of images using the raster scanning method. Laser printers too use the raster scanning method to produce the output using a spinning polygonal mirror which scans across the photosensitive drum. The movement of the paper is provided by the other scan axis. In inkjet printers, there are multiple nozzles in the print heads, hence *n* number of 'scan lines' are printed together. However, converting a vector-based data into the required form needs a Raster Image Processor (RIP).

The resolution is the maximum number of points displayed without overlap. These are the pixels and (picture elements). For example, a monitor might have a resolution of 1024 × 768 pixels. Computer graphics systems can be random scan or raster scan.

A raster scan device scans the screen from top to bottom in a specific pattern, as shown in Figure 1.21.

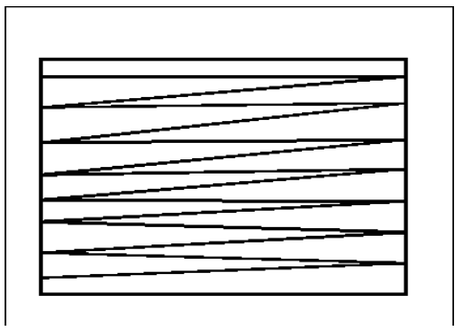


Fig. 1.21 *Scanning the Screen from Top to Bottom in a Specific Pattern*

The electron beam can be turned on/off so that the image is displayed on the screen in dot format one scan or raster line at a time. All the PCs use raster scan for colouring/shading.

The colour CRT uses the raster scan shadow masks of different phosphors, generally red, green, blue and then combined. Figure 1.22 shows the raster scan methodology.

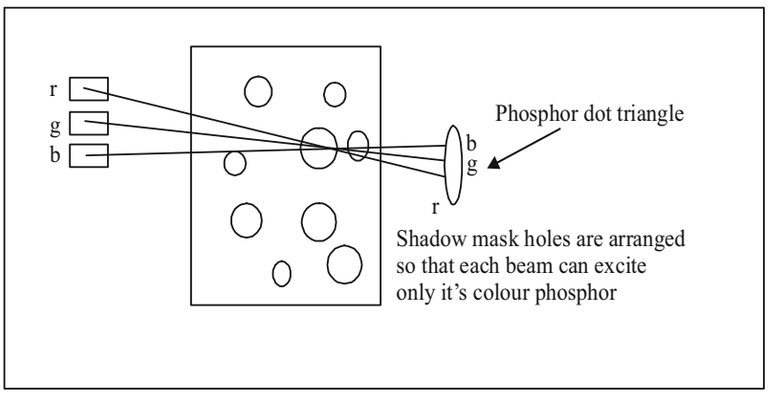


Fig. 1.22 *Raster Scan Methodology*

Hence, different colours can be obtained by combining different intensities, as shown below:

Red	Green	Blue	Final Colour
0	0	0	black
0	0	1	blue
0	1	0	green
0	1	1	cyan
1	0	0	red
1	0	1	magenta
1	1	0	yellow
1	1	1	white

NOTES

NOTES

1.9.1 Random Scan Systems

A random scan display in a CRT has the electron beam focused only on the parts of the screen where an image is to be drawn. Random scan monitors depict an image, one line at a time, and hence are termed as vector displays. The constituent lines of an image can be drawn using a random scan system in any specified order. A pen plotter is a typical example.

Random scan systems are specially designed for drawing lines. It cannot display sensible shaded scenes. Because the defined picture is saved as an instruction set of line drawings and not as a set of power values for all the screen points, the vector display is of higher resolution in comparison to raster systems. The vector displays create smooth line drawings because the CRT beam follows the line path.

The random scan systems have the following features:

- Random displays have high resolutions because the image definition is saved as a set of line drawing commands and not as a set of power values.
- Smooth lines are created because the electron beam directly follows the line path.
- Pragmatism is difficult.
- Random scan systems are very efficient and sensitive.

As we know a computer graphics systems may be random scan or raster scan. Random Scan system, which draws the image directly, is also termed as vector, calligraphic or stroke characters of electron beam.

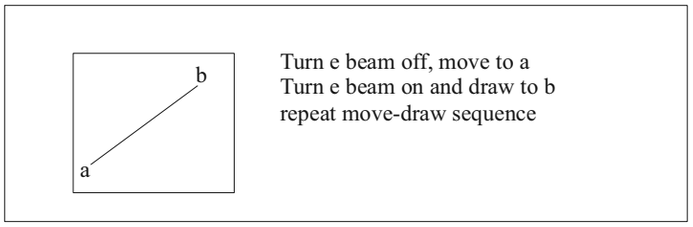


Fig. 1.23 *Electron Beam Drawing Image*

The advantages of the random scan system are as follows:

- Has high resolution
- Simple animation so that image can be drawn at different positions
- Needs less memory

The disadvantages of the random scan system are as follows:

- Needs intelligent electron beam or processor controller
- Limited screen density, hence cannot draw a compound image
- Limited colour capacity
- Very expensive

NOTES

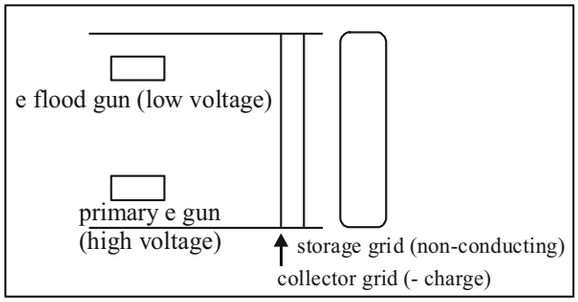


Fig. 1.24 *Random Scan Methodology*

The electron gun draws the image by sending electrons from the storage grid producing a pattern. Once an image is drawn on screen, it remains there until it is erased using a charge. A pen plotter and laser light show are also the examples of random scan display devices.

The following is the hardware system interface for interactive graphics.

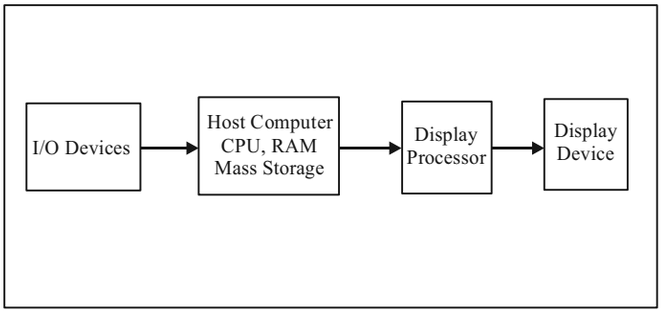


Fig. 1.25 *Hardware System Interface*

The host system executes a display file with graphics commands for random scan which can be operated by the display processor. For simple PCs, the raster scan system executes the processor performance conversion, from a mathematical model to a frame buffer image. Now the frame buffer is read by a display processor which turns the electron beams on/off. The advanced and costly raster scan systems have a graphic processor to perform the scan conversion.

1.10 HARD COPY OUTPUT DEVICES

In this section, we will discss the hard copy output devices.

Printer

The printer is an important accessory of any computer system, specially for a graphics system. Because most of the graphics creations using computer graphics have their ultimate utilization in printed form – used for documentation, exhibition or publication in print media or books, it is the quality of printed output that finally matters in many businesses.

Based on the available printing technology, the major factors which control the quality of printer are the individual dot size on the paper and the number of dots per inch (dpi). Quite obviously, the minimum the size of the dots the better the detail of the figure reproduced. Higher dpi values increase the sharpness and detail

NOTES

of a figure and enhance the number of intensity levels that a printer supports. Other important factors for selection of a printer are printing speed and print area or printer memory.

Impact vs Non-impact Technologies

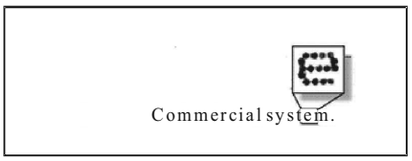
There are several printer technologies available. These technologies can be grouped under the following two main categories:

- Impact technology: Impact printers have a mechanism whereby formed character faces are pressed against an inked ribbon onto the paper in order to create an image, e.g., dot matrix printer and line printer.
- Non-impact technology: Non-impact printers do not touch the paper, rather they use laser techniques, ink sprays, xerographic processes and electrostatic methods to produce image on paper, e.g., laser printer, inkjet printer, electrostatic printer, drum plotter, flatbed plotter.

Dot Matrix Printer

A dot matrix printer refers to a type of computer printer with a print head (usually containing 9 to 24 pins) that runs back and forth on the page and prints by impact, striking an ink-soaked cloth ribbon against the paper, much like a typewriter. Unlike a typewriter or daisy wheel printer, letters are drawn out of a dot matrix, and thus, varied fonts and arbitrary graphics can be produced. Because the printing involves mechanical pressure, these printers can create carbon copies. The print head normally prints along every raster row of the printer paper and the colour of print is the colour of the ink of the ribbon.

Dot Matrix Printer



Typical Output from Dot Matrix Printer

Fig. 1.26 *Dot Matrix Printer and its Output*

Each dot is produced by a tiny yet stiff metal rod, also called a 'wire' or 'pin', which is driven forward by the power of a tiny electromagnet or solenoid, either directly or through small levers (pawls). The pins are usually arranged vertically where marginal offsets are provided between columns to reduce the inter dot spacing. The position of pins in the print head actually limits the quality of such printer.

Though hardware improvements to dot matrix printers boosted the carriage speed, added more (typeface) font options, increased the dot density (from 60 dpi up to 240 dpi), and added pseudo-colour printing through multi-colour ribbon, still such printers lack the ability to print quality computer generated images at acceptable quality. However, they are good for text printing in continuous sheets.

Strictly speaking, 'dot matrix' in this context is a misnomer, as nearly all inkjet, thermal and laser printers produce dot matrices. However, in common parlance, these are seldom called 'dot matrix' printers so as to avoid confusion with dot matrix impact printers.

Line Printer

A line printer is a form of high speed impact printer in which a line of type is printed at a time.

In a typical design, a fixed font character set is engraved onto the periphery of a number of print wheels, the number matching the number of columns (letters in a line). The wheels spin at high speed and paper and an inked ribbon are moved past the print position. As the desired character for each column passes the print position, a hammer strikes the paper and ribbon causing the desired character to be recorded on the continuous paper. Printed type is set at fixed positions and a line could consist of any number of character positions with 132 columns the most common, but 80 column, 128 column and 160 column variants are also in use. Other variations of line printer have the type on moving bars or a horizontal spinning chain.

The line printer technology is usually both faster and less expensive (in total ownership) than laser printers. It has its use in medium volume accounting and other large business applications, where print volume and speed are priorities and not quality. Because of the limited character set engraved on the wheels and the fixed spacing of type, this technology was never useful for material of high readability such, as books or newspapers.

Fig. 1.27 *Line Matrix Printer*

Inkjet Printer

An inkjet printer is a non-impact printer that places extremely small droplets of ink onto paper to create an image. It is popular because it costs less but generates the attractive graphic output.

The dots sprayed on paper are extremely small (usually between 50 and 60 microns in diameter) and are positioned very precisely, with resolutions of up to 1440 × 720 dpi. Theys can have different colours combined together to create photo-quality images.

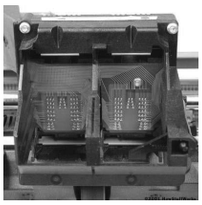
NOTES

The core of an inkjet printer is the print head that contains a series of nozzles that are used to spray drops of ink. The ink is contained in ink cartridges that come in various combinations, such as separate black and colour cartridges, or a cartridge for each ink colour. A stepper motor moves the print head assembly (print head and ink cartridges) back and forth across the paper. The mechanical operation of the printer is controlled by a small circuit board containing a microprocessor and memory.

There are two main inkjet technologies currently used by printer manufacturers.

- **Thermal bubble (or bubble jet):** This technology used by manufacturers such as Canon and Hewlett Packard. In a thermal inkjet printer, tiny resistors create heat, and this heat vaporizes ink to create a bubble. As the bubble expands, some of the ink is pushed out of a nozzle onto the paper. When the bubble 'pops' (collapses), a vacuum is created. This pulls more ink into the print head from the cartridge. A typical bubble jet print head has 300 or 600 tiny nozzles, and all of them can fire a droplet simultaneously.
- **Piezoelectric:** Patented by Epson, this technology uses piezo crystals. A crystal is located at the back of the ink reservoir of each nozzle. The crystal receives a tiny electric charge that causes it to vibrate. When the crystal vibrates inward, it forces a tiny amount of ink out of the nozzle. When it vibrates outward, it pulls some more ink into the reservoir to replace the ink sprayed out.

Canon Inkjet Printer



Print Head Assembly

***Fig. 1.28** An Inkjet Printer*

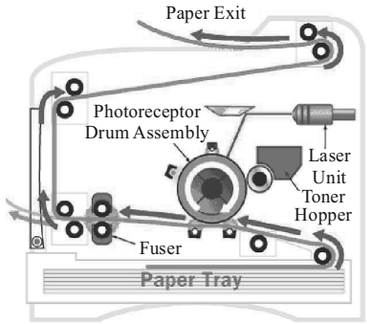
Laser Printer

A laser printer employs the technology similar to a photocopy machine. A laser beam focuses a positively charged selenium-coated rotating drum. The laser gun removes the positive charge from the drum except for the area to be printed (black portion of the paper). In this way, the laser draws the letters and images to be printed as a pattern of electrical charges — an electrostatic image. The negatively charged black toner powder first adheres to this positively charged area (image) on the drum from where it is transferred to the rolling white paper. Before the paper rolls under the drum, it is given a positive charge stronger than the positive charge of the electrostatic image, so the paper can pull the toner powder away. The paper is then subjected to mild heating to melt and fix the loose toner on the paper. The laser printer is mainly a bi-level printer. In case of colour lasers, this process is repeated three times.

For the printer controller and the host computer to communicate, they need to speak the same page description language. The primary printer languages these

days are Hewlett Packard's Printer Command Language (PCL) and Adobe's Postscript. Both the languages describe the page in vector form — that is, as mathematical values of geometric shapes, rather than as a series of dots (a bitmap image). Apart from image data, the printer controller receives all of the commands that tell the printer what to do — what paper to use, how to format the page, how to handle the font, etc. Accordingly, the controller sets the text margins, arranges the words and places the graphics. When the page is arranged, the raster image processor (RIP) takes the page data, either as a whole or piece by piece, and breaks it down into an array of tiny dots so that laser can write it out on the photoreceptor drum. In most laser printers, the controller saves all print-job data in its own memory. This lets the controller put different printing jobs into a queue so that it can work through them one at a time.

HP Laser Printer



Path of Paper inside a Laser Printer

Fig. 1.29 *A Laser creation by the Laser on Drum*

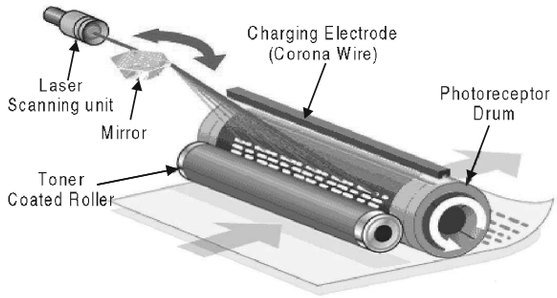


Fig. 1.30 *Image creation by the Laser on Drum*

Electrostatic Printer

In inkjet printers, the single printing head moves left-to-right and prints as it is travelling. In contrast, the electrostatic printer has many print heads, actually covering the entire 36" media width. So, instead of a single print head moving across the width of the media, the electrostatic printer prints the entire width of the page at one time. The media (paper, vellum, film) is electrostatically charged (energized). The toner solution is circulated past the media and 'sticks' to the energized portion of the media, thus producing a fast high quality image.

The printer creates colour prints by breaking colour data down into three basic colours (cyan, magenta and yellow) plus black, and printing one colour at a

NOTES

time. In 5-pass print mode, combinations of cyan, magenta, yellow and black provide a wide range of colours. Using the registration marks printed during the preliminary registration pass ensures that the colour plot is beautiful with no misalignment.

Fig. 1.31 *HP Electrostatic Printer*

Plotter

In contrast to the printer which is primarily a raster scan device, the plotter is a vector device. In colour plotters, the carriage accommodates a number of pens with varying colours and widths. The microprocessor in the plotter receives instructions from the host computer and executes commands like 'move' (moving the carriage to a given position with pens up) and 'draw' (drawing geometric entities like point, line, arc, circle, etc., with pens down). Since the plotter is a vector device, it can directly reach specific positions on printer paper without following the raster row sequence. In *flatbed plotter,* the paper lies flat and stationary, while the pen moves from one location to another on the paper. But in *drum plotter,* the paper itself slides on a cylindrical drum and the pen moves over the drum.

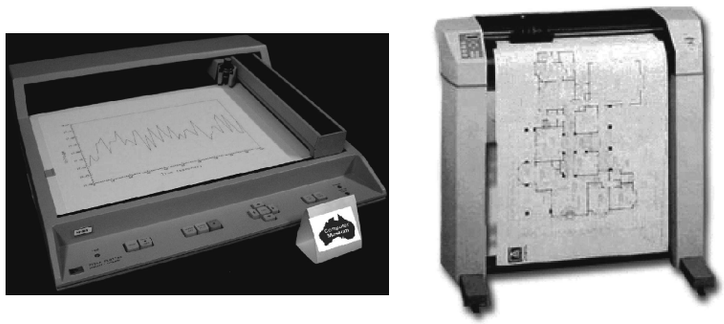


Fig. 1.32 *Fl2atbed Plotter and Drum Plotter*

Check Your Progress

7. What are the constituents of an image in raster scan display?
8. Differentiate between raster scan technique and the random scan technique.
9. Define the term random scan system.
10. Differentiate between the impact and non-impact printers.
11. What is the difference between a flatbed plotter and a drum plotter?

1.11 ANSWERS TO 'CHECK YOUR PROGRESS'

NOTES

1. Computer Graphics is the discipline of producing pictures or images using a computer. It includes creation of model, manipulation and storage of geometric objects, reproduction (an image converted from a scene), transformation (primitive graphics operations), illumination, rasterization, animation, and shading of the image.
2. Some of the common types of display systems available in the market are:
   - Raster Scan Displays
   - Random Scan Displays
   - Direct View Storage Tube
   - Three-Dimensional Viewing Devices
   - Stereoscopic and Virtual Reality System
3. If the image resolution is more compared to the inherent resolution of the display device, then the quality the displayed image gets reduced.
4. Text mode is a computer display mode in which content is internally represented on a computer screen in terms of characters rather than individual pixels. The screen consists of a uniform rectangular grid of character cells, each of which contains one of the characters of a character set; at the same time, contrasted to All Points Addressable (APA) mode or other kinds of computer graphics modes.
5. Monochrome Display Adapter (MDA, also MDA card, Monochrome Display and Printer Adapter, MDPA) is standard video display card and computer display standard for the PC introduced in 1981.
6. VGA stands for video graphics array, is currently the most popular standard for PC screen display equipment. Technically, a VGA is a type of video adapter (circuitry in the computer that controls the screen).
7. An image in raster scan display is basically composed of a set of dots and lines; lines are displayed by making those dots bright (with desired colour) which lie as close as possible to the shortest path between the endpoints of a line.
8. In the raster scan technique, the electron beam sweeps the entire screen in the same way you would write a full page text in a notebook, word by word, character by character, from left to right and from top to bottom. In the random scan technique, on the other hand, the electron beam is directed straightway to the particular point(s) of the screen where the image is to be produced.
9. A random scan display in a CRT has the electron beam focused only on the parts of the screen where an image is to be drawn. Random scan monitors depict an image, one line at a time, and hence are termed as vector displays.
10. The impact printers have a mechanism whereby formed character faces are pressed against an inked ribbon onto the paper in order to create an image. The non-impact printers, on the other hand, do not touch the paper. They

NOTES

use laser techniques, ink sprays, xerographic processes and electrostatic methods to produce image on paper.

11. In a flatbed plotter, the paper lies flat and stationary, while the pen moves from one location to another on the paper. In a drum plotter, on the other hand, the paper itself slides on a cylindrical drum and the pen moves over the drum.

1.12 SUMMARY

- The term 'computer graphics 'was first used by William Fetter in 1960. Fetter was a graphics designer for Boeing Aircraft Co. The term was actually given to him by Verne Hudson.
- Computer Graphics is the discipline of producing pictures or images using a computer. It includes creation of model, manipulation and storage of geometric objects, reproduction (an image converted from a scene), transformation (primitive graphics operations), illumination, rasterization, animation, and shading of the image.
- The LCD displays are addressed in a matrix fashion. Rows of matrix are defined by a thin layer of horizontal transparent conductors while columns are defined by another thin layer of vertical transparent conductors.
- A pixel may be defined as the smallest size object or colour sport that can be display and addressed on a monitor. Any image that is displayed on the monitor is made up of the thousands of such small pixels (also known as picture elements.)
- If the image resolution is more compared to the inherent resolution of the display device, then the quality the displayed image gets reduced.
- Text mode is a computer display mode in which content is internally represented on a computer screen in terms of characters rather than individual pixels. The screen consists of a uniform rectangular grid of character cells, each of which contains one of the characters of a character set; at the same time, contrasted to All Points Addressable (APA) mode or other kinds of computer graphics modes.
- Monochrome Display Adapter (MDA, also MDA card, Monochrome Display and Printer Adapter, MDPA) is standard video display card and computer display standard for the PC introduced in 1981.
- The Hercules Graphics Card (HGC) is a computer graphics controller made by Hercules Computer Technology, Inc.
- The Enhanced Graphics Adapter (EGA) is video adapter for IBM PC graphics adapter and facto computer display standard from 1984 that superseded the CGA standard introduced with the original IBM PC, and was itself superseded by the VGA standard in 1987.
- There are two way to communicate display, store or manipulate information are follows digital and analog.

NOTES

- VGA stands for video graphics array, is currently the most popular standard for PC screen display equipment. Technically, a VGA is a type of video adapter (circuitry in the computer that controls the screen).
- A CRT is similar to a big vacuum glass bottle. It contains three electron guns that squirt out focused beam of electrons, deflection apparatus (magnetic or electrostatic), that deflects these beams both up and down and sidewise and a phosphor-coated screen upon which these beams impinge
- In the raster scan technique, the electron beam sweeps the entire screen in the same way you would write a full page text in a notebook, word by word, character by character, from left to right and from top to bottom. In the random scan technique, on the other hand, the electron beam is directed straightway to the particular point(s) of the screen where the image is to be produced.
- The LCD displays are addressed in a matrix fashion. Rows of matrix are defined by a thin layer of horizontal transparent conductors, while columns are defined by another thin layer of vertical transparent conductors; the layers are placed between the LCD layer and the respective polarizer plate.
- A random scan display in a CRT has the electron beam focused only on the parts of the screen where an image is to be drawn. Random scan monitors depict an image, one line at a time, and hence are termed as vector displays.
- The printer is an important accessory of any computer system, specially for a graphics system.
- A laser printer employs the technology similar to a photocopy machine. A laser beam focuses a positively charged selenium-coated rotating drum.

1.13 KEY TERMS

- **Computer Graphics:** It is the discipline of producing pictures or images using a computer.
- **Pixel:** It refers the smallest size objects or colour sport that can be displayed and addressed on a monitor.
- **Resolution of an image:** It refers to the total number of pixels along the entire height and width of the image.
- **Hercules Graphics Card (HGC)**: It is a computer graphics controller made by Hercules Computer Technology, Inc.
- **Raster scan system:** A raster scan or raster scanning is a design or pattern of the image detection and its reconstruction on computer screen and is stored in computer memory.
- **Plotter:** In contrast to the printer which is primarily a raster scan device, the plotter is a vector device.

1.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. What is presentation graphics?
2. State about the plasma panel.
3. Differentiate between image resolution and screen resolution.
4. Define the term graphics mode.
5. What is color graphics adapter?
6. What do you mean by the term super VGA?
7. What is raster scan displays?
8. How will you define the flat-panel display?
9. State about the raster scan system.

Long-Answer Questions

1. Discuss various types of computer graphics with the help of appropriate example.
2. Describe the various applications of computer graphics.
3. Explain the fundamentals operations of liquid crystal display with the help of diagram.
4. Explain the display adapter with the help of appropriate example.
5. Describe the enhanced and professional graphics adapter.
6. Differentiate between digital and analog with the help of diagram.
7. Discuss the Cathode-Ray Tube (CRT) with the help of diagram.
8. Explain the block diagram of random scan display.
9. Explain with the help of a diagram the concept of the random scan system.
10. Explain the laser printer with respect to the following points: printing mechanism, speed, printing quality and application.

1.15 FURTHER READING

Hearn, Donald and M. Pauline Baker. *Computer Graphics*. New Jersey: Prentice Hall.

Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Higher Education.

Foley, James D., Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice*. London: Addison-Wesley Professional.

Mukhopadhyay, Anirban and Arup Chattopadhyay. *Introduction to Computer Graphics and Multimedia*. New Delhi: Vikas Publishing House Pvt Ltd.

UNIT 2 GRAPHICS INPUT DEVICES, MATRICES, DETERMINANTS AND VECTORS

NOTES

Structure

2.0 INTRODUCTION

Various input devices, such as digitizer, graphic tablets, keyboard, light pen, joysticks, trackballs and spaceball, data glove, image scanner, flatbed scanner, etc., are available for data input to general purpose computer systems with graphic capabilities or sophisticated workstations designed for graphics applications.

The matrix inversion process is explained for finding matrix B that satisfies the prior equation for a given invertible matrix A. In linear algebra, an n-by-n square matrix A is called invertible or non-singular, if there exists an n-by-n matrix B such that, AB = BA = In, where In denotes the n-by-n identity matrix and the

multiplication used is ordinary matrix multiplication. In this case, matrix B is uniquely determined by A and is called the inverse of A, denoted by A^1.

Vector, in mathematics, refers to a quantity, such as velocity, that is specified by a magnitude and a direction. A vector is a definite mathematical structure. There are numerous physical and geometric applications of vectors. The ability to simultaneously represent magnitude and direction facilitates its numerous applications.

In this unit you will study about the graphics input devices, input of graphical data, logical classification of input devices, input function, input device parameter, interactive picture construction techniques, metrics and its types, submatrices and determinants, properties of determinants, vector, components of vector, collinear and coplanar vectors, some applications to geometry.

2.1 UNIT OBJECTIVES

After going through this unit, you will be able to:

- Explain various graphics input devices
- Discuss the input of graphical data
- Describe the logical classification of input devices
- Explain the input function and input device parameter
- Explain the interactive picture construction techniques
- Describe the matrices and its types
- Discuss the concepts of submatrices and determinants
- Describe the various properties of determinants
- Explain the components of vector
- Discuss the vector applications to geometry

2.2 GRAPHICS INPUT DEVICES

Various devices are available for data input to general-purpose computer systems with graphic capabilities or sophisticated workstations designed for graphics applications. Examples include keyboard, mouse, trackball, joystick, digitizer, graphics tablet, touch panel, light pen, data glove and voice system. Presented below are the basic functional characteristics and applications of these devices.

2.2.1 Keyboard

Using a keyboard, a person can type a document, use keystroke shortcuts, access menus, play games and perform a variety of other tasks. Though keyboards can have different keys depending on the manufacturer, the operating system they are designed for, and whether they are attached to a desktop computer or part of a laptop, still most keyboards have between 80 and 110 keys, including:

- Typing keys (letters A to Z, a to z, characters like < , ? + = etc.)
- A numeric keypad (numbers 0 to 9, characters like ! @ # ( ) etc.)
- Function keys (F1 to F12)

- Control keys (Ctrl, Alt, Del, Pg Up, Pg Dn, Home, End, Esc, , Fn, arrow keys etc.)

Function keys allow users to enter frequently used operations in a single keystroke and control keys allow cursor and screen control. Displayed objects and menus can be selected using the control keys.

A keyboard is much like a miniature computer. It has its own processor, circuitry (key matrix) and a ROM storing the character map. It uses a variety of switch technologies.

Though the basic working technology is same, there are design variations to make the keyboards easier, safer, versatile and elegant. Some of the non-traditional keyboards are Das keyboard, Virtual Laser keyboard, True-touch Roll-up keyboard, Ion Illuminated keyboard, Wireless keyboard, etc.

Fig. 2.1 *Microsoft Wireless Keyboard*

2.2.2 Mouse

A mouse is basically a handheld pointing device, designed to sit under one hand of the user and to detect movement relative to its two-dimensional supporting surface. It has become an inseparable part of a computer system, just like keyboard. There is a cursor in the shape of an arrow or cross-hair always associated with a mouse. We reach out for mouse whenever we want to move the cursor or activate something or drag and drop or resize some object on display. Drawing or designing figures and shapes using graphic application packages, such as AutoCAD, Photoshop, CorelDraw, Paint, etc., is almost impossible without a mouse.

The mouse's 2D motion typically translates into the motion of a pointer on a display. In a *mechanical mouse,* a ball – roller assembly is used; one roller used for detecting X direction motion and the other for detecting Y direction motion. An *optical mouse* uses LED and photodiodes (or opto-lectronic sensors) to detect the movement of the underlying surface, rather than moving some of its parts as in a mechanical mouse. Modern *laser mouse* uses a small laser instead of a LED.

A mouse may have one, two or three buttons on the top. Usually, clicking the primary or leftmost button will select items or pick screen-points, while clicking the secondary or rightmost button will bring up a menu of alternative actions applicable to the selected item or specific to the context. Extra buttons or features are included in the mouse to add more control or dimensional input.

Fig. 2.2 *Microsoft's Two-button Wireless Mouse*

2.2.3 Trackball

A trackball is a pointing device consisting of a ball housed in a socket containing sensors to detect the rotation of the ball about two axes—like an upside-down mouse with an exposed protruding ball. The user rolls the ball with his thumb, fingers, or the palm of their hand to move a cursor. A potentiometer captures the track ball orientation which is calibrated with the translation of the cursor on screen. Tracker balls are common on CAD workstations due to ease of use and, before the advent of the touchpad, they were commonly used in portable computers, where there may be no desk space on which to use a mouse.

Fig. 2.3 *A Logitech Trackball*

2.2.4 Joystick

A joystick is used as a personal computer peripheral or general control device consisting of a handheld stick that pivots about the base and steers the screen cursor around. Most joysticks are two-dimensional, having two axes of movement (similar to a mouse), but three-dimensional joysticks do exist. A joystick is generally configured so that moving the stick left or right signals the movement along the X-axis, and moving it forward (up) or backward (down) signals the movement along the Y-axis. In joysticks that are configured for three-dimensional movement, twisting the stick left (counter-clockwise) or right (clockwise) signals the movement along the Z-axis. In conventional joysticks, potentiometers or variable resistors are used to dynamically detect the location of the stick and springs are there to return the stick to centre position as it is released.

In many joysticks, optical sensors are used instead of analog potentiometer to read the stick movement digitally. One of the biggest additions to the world of

joysticks is force feedback technology. On using a force feedback (also called haptic feedback) joystic, if you are shooting a machine gun in an action game, the stick would vibrate in your hands. Or if you crashed your plane in a flight simulator, the stick would push back suddenly which means the stick moves in conjunction with onscreen actions.

Joysticks are often used to control games, and usually have one or more push-buttons whose state can also be read by the computer. Most I/O interface cards for PCs have a joystick (game control) port. Joysticks were popular throughout the mid-1990s for playing games and flight-simulators, although their use has declined with promotion of the mouse and keyboard.

Fig. 2.4 *The Flighterstick, a Modern Programmable USB Joystick*

2.2.5 Digitizer and Graphics Tablet

A digitizer is locator device used for drawing, painting, or interactively selecting coordinate positions on an object. A graphic tablet is one such digitizer that consists of a flat surface upon which the user may draw an image using an attached stylus, a pen-like drawing apparatus. The image generally does not appear on the tablet itself but, rather, is displayed on the computer monitor.

The first graphics tablet resembling contemporary tablets was the RAND Tablet, also known as the Grafacon (for Graphic Converter), employed an orthogonal grid of wires under the surface of the pad. When pressure is applied to a point on the tablet using a stylus, the horizontal wire and vertical wire associated with the corresponding grid point meet each other, causing an electric current to flow into each of these wires. Since an electric current is only present in the two wires that meet, a unique coordinate for the stylus can be retrieved. The coordinate returned are tablet coordinates which are converted to user or screen coordinates by an imaging software. Even if it does not touch the tablet, proximity of the stylus to the tablet surface can also be sensed by virtue of a weak magnetic field projected approximately one inch from the tablet surface. It is important to note that, unlike the RAND Tablet, modern tablets do not require electronics in the stylus and any tool that provides an accurate 'point' may be used with the pad. In some tablets, the multiple button hand-cursor is used instead of stylus. Graphics tablets are available in various sizes and price ranges. A 6-sized tablets being relatively inexpensive and a 3-sized tablets being far more expensive.

Modern tablets usually connect to the computer via a USB interface. Because of their stylus-based interface and (in some cases) ability to detect

pressure, tilt, and other attributes of the stylus and its interaction with the tablet, they are widely used to create two-dimensional computer graphics. Free-hand sketches by an artist or drawing following an existing image on the tablet are useful while digitizing old engineering drawings, electrical circuits and maps and toposheets for GIS. Indeed, many graphics packages (e.g., Corel Painter, Inkscape, Photoshop, Pixel image editor, Studio Artist, The GIMP) are able to make use of the pressure (and, in some cases, stylus tilt) information generated by a tablet, by modifying attributes such as brush size, opacity, and colour. Three-dimensional graphics can also be created by a 3D digitizer that uses sonic or electromagnetic transmissions to record positions on a real object as the stylus moves over its surface.

Fig. 2.5 *A Tablet with Hand Cursor*

2.2.6 Touch Panel

A touch panel is a display device that accepts user input by means of a touch sensitive screen. The input is given by touching displayed buttons or menus or icons with finger. In a typical *optical touch panel,* LEDs are mounted in adjacent edges (one vertical and one horizontal). The opposite pair of adjacent edges contain light detectors. These detectors instantly identify which two orthogonal light beams emitted by the LEDs are blocked by a finger or other pointing device and thereby record the x, y coordinates of the screen position touched for selection. However, because of its poor resolution, the touch panel cannot be used for selecting very small graphic objects or accurate screen positions.

The other two type of touch panels are *electrical* (or *capacitive*) and *acoustical* touch panels. In an electrical touch panel, two glass plates coated with appropriate conductive and resistive materials are placed face to face, similar to capacitor plates. Touching a point on the display panel generates force which changes the gap between the plates. This in turn causes a change in capacitance across the plates that is converted to coordinate values of the selected screen position. In a acoustic type touch panel, similar to the light rays, sonic beams are generated from the horizontal and vertical edges of the screen. The sonic beam is obstructed or reflected back by putting a finger in the designed location on the screen. From the time of travel of the beams, the location of the finger tip is determined.

Touch panels have gained wide acceptance in bank ATMs, video games and railway or tourist information system.

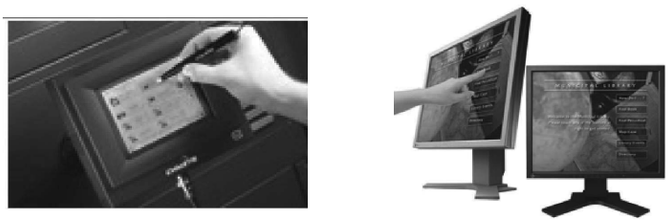


Fig. 2.6 *Touch Panels*

2.2.7 Light Pen

A light pen is a pointing device shaped like a pen and is connected to the computer. The tip of the light pen contains a light-sensitive element (photoelectric cell) which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen. It allows the user to point to displayed objects, or draw on the screen, in a similar way as a touch screen 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 actually 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 positions of the pen can be resolved. The pen position is updated on every refresh of the screen.

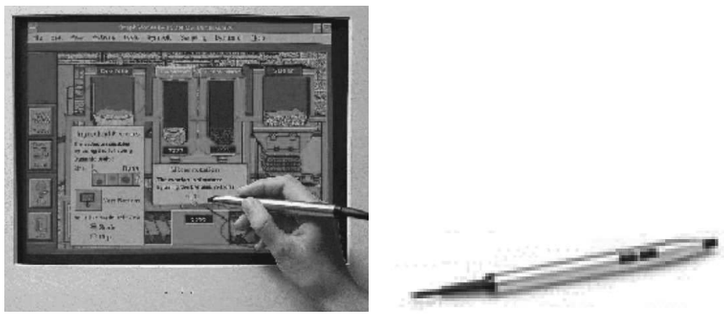


Fig. 2.7 *Light Pen*

Light pens are popularly used to digitize map or engineering drawing or signature or handwriting.

2.2.8 Data Glove

A data glove is an interface device that uses position tracking sensors and fiber optic strands running down each finger and connected to a compatible computer; the movement of the hand and fingers are displayed live on the computer monitor which in turn allows the user to virtually touch an object displayed in the same monitor. With the object animated, it would appear that the user (wearing the data glove) can pick up an object and do things with it just as he would do with a real object. In modern data glove devices, tactile sensors are used to provide the user with an

additional feeling of touch or the amount of pressure or force the fingers or hands are exerting even though the user is not actually touching anything. Thus, data glove is an agent to transport the user to virtual reality.

NOTES

Fig. 2.8 *Data Glove*

2.2.9 Voice Recognition System

The voice recognition system or speech recognition system is a sophisticated input device that accepts voice or speech input from the user and transforms it to digital data that can be used to trigger graphic operations or enter data in specific fields. A dictionary is established for a particular operator (voice) by recording the frequency patterns of the voice commands (words spoken) and corresponding functions to be performed. Later, when a voice command is given by the same operator, the system searches for a frequency pattern match in the dictionary and if it is found, the corresponding action is triggered. If a different operator is to use the system, then the dictionary has to be re-established with the new operator's voice patterns.

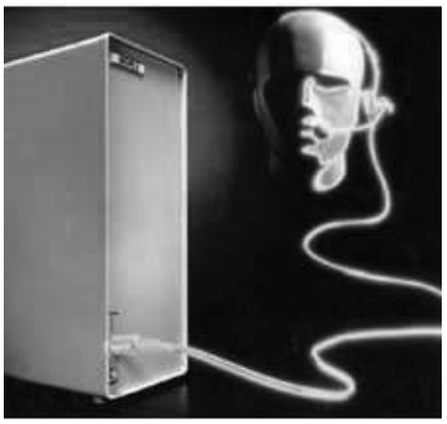


Fig. 2.9 *Operator's Speech Recording*

2.2.10 Spaceball

Spaceball is a graphical input device that is based on a fixed spherical ball. It inputs six different values defined by the orientation of the ball and the pressure

together with the direction that is applied to it. It allows complex objects to be positioned and rotated in three-dimensional space using the single input device. Internally a spaceball is normally made from a set of strain-gauges.

2.2.11 Image Scanners

The image scanner is a graphic device which directly copies images from a paper or photograph and converts it into the digital format for display, storage and graphic manipulations [refer Figure 1.10(a)]. Traditionally, design and publishing houses have been the prime users of scanners but the phenomenal growth of Internet has made the scanner more popular among the web designers. Today scanners are becoming affordable tools to the graphic artists and photographers.

There are basically three types of scanners – *Drum*, *Flatbed* and *Sheetfed*. Drum are the high end ones whereas sheetfed scanners are the ordinary type. Flatbed scanners strike a balance between the two in quality as well as price. There are also *handheld scanners* or *barcode readers* which are typically used to scan documents in strips of about 4 inches wide by holding the scanner in one hand and sliding it over the document. The purpose of scanner is to create a point cloud of geometric samples on the surface of the scanner [refer Figure 2.10(b)].

Fig. 2.10 *(a) Scanner*

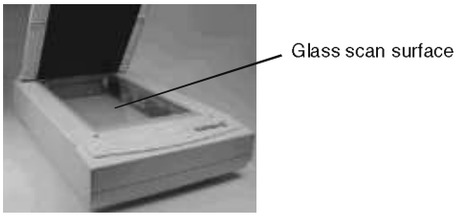


Fig. 2.10 *(b) Scan Surface in a Scanner*

Flatbed Scanner

A flatbed scanner uses a light source, a lens, a Charge Coupled Device (CCD) array and one or more Analog-to-Digital Converters (ADCs) to collect optical information about the object to be scanned and transform it to an image file. A CCD is a miniature photometer that measures incident light and converts that into an analog voltage.

When you place an object on the copyboard or glass surface (like a copier machine) and start scanning, the light source illuminates a thin horizontal strip of the object called a *raster line*. Thus, when you scan an image you scan one line at a time. During the exposure of each raster line, the scanner carriage optical imaging elements which is a network of lenses and mirrors is mechanically moved over a short distance using a motor. The light reflected is captured by the CCD array. Each CCD converts the light to an analog voltage and indicates the gray level for one pixel. The analog voltage is then converted into a digital value by an ADC using 8, 10 or 12 bits per color. Figure 2.11 shows mirror and lens assembly in the scanner carriage.

NOTES

Fig. 2.11 *Mirror and Lens Assembly*

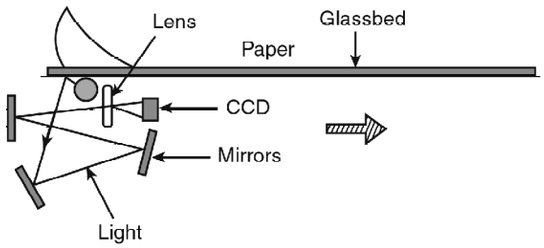


Fig. 2.12 *Scanning Operation*

Figure 2.12 shows scanning operation in which the assembly of light, mirrors, lens and CCD moves over the length of the glassbed in the direction shown while scanning a paper.

The CCD elements are all in a row with one element for each pixel in a line. If you have 300 CCD elements per inch across the scanner you can have a maximum potential optical resolution of 300 pixels per inch also referred to as dots per inch (dpi).

There are two methods by which the incident white light is sensed by the CCD. The first involves a rapidly rotating light filter that individually filters the red, green and blue components of the reflected light which are sensed by a single CCD device. Here, the color filter is fabricated into the chip directly. In the second method, a prismatic beam splitter first splits the reflected white light and three CCDs are used to sense the red, green and blue light beams.

Another imaging array technology that has become popular in inexpensive flatbed scanners is Contact Image Sensor (CIS). CIS replaces the CCD array, mirrors, filters, lamp and lens with rows of red, green and blue Light Emitting Diodes (LEDs). The image sensor mechanism, consisting of 300 to 600 sensors spanning the width of the scan area, is placed very close to the glass plate that the document rests upon. When the image is scanned, the LEDs combine to provide white light. The illuminated image is then captured by the row of sensors. CIS scanners are cheaper, lighter and thinner but do not provide the same level of quality and resolution found in most CCD scanners.

The output of a scanner is a bitmap image file, usually in a Personal Computer Exchange (PCX) or Joint Photographic (Experts) Group (JPG) format. If you scan a page of text, it may be saved as an image file which can not be edited in a word processing software. Optical Character Recognition (OCR) softwares are intelligent programs which can convert a scanned page of text into editable text either into a plain text file, a Word document or even an Excel spreadsheet which can be easily edited. OCR can also be used to scan and recognize printed, typewritten or even handwritten text. The OCR software requires a raster image as an input which may be an existing image file or an image transferred from a scanner. OCR analyses the image to find blocks of image information that resemble possible text fields and creates an index of such areas. The software examines these areas which compares shape of each object with a database of words categorized by different fonts or typefaces and recognizes individual text characters from the information.

NOTES

2.2.12 Input of Graphical Data

Graphical programs are required to input graphical data. To input data in an illustration, we require specific inputs which include the values for coordinate positions and character-string parameters. Values are also defined for menu options, recognition of picture and special scalar parameters for transformation. Input functions can be modified and structured on the basis of the use of a particular input hardware device which makes them user-defined functions.

The logical classification of input devices decides the way to receive the graphical data to be processed by a graphic package.

- `LOCATOR` (Locates point x, y)
- `STROKE` (Series of x, y)
- `STRING` (Text or collection of symbols)
- `VALUATOR` (Scalar knob, slider)
- `CHOICE` (Menu selector)
- `PICK` (Select components, select window)

In some graphical packages, a single logical device can be used for performing LOCATOR and STROKE.

2.2.13 Input Functions

The graphical input functions can be set up in such a way that they can allow users to specify the following options:

- Which physical devices are to provide input within a particular logical classification (tablet as a stroke device)
- How the graphics operations and devices are to interact in input mode. Either the program or the devices can initiate the data entry operation.
- When the data are to be input and which input device is to be used at that time to provide a particular input type to the specified data variables at specified memory location.

  Input functions are performed using keyboards, mouse, data tablets, etc.

Input can also be done using graphics mode and text mode. Basically, they are two different applications. To invoke these two modes, you should not use a GUI interface, otherwise your text console I/O functions, such as printf(), scanf(), cin, cout, etc., will not work.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int num;
```

```
int gd=DETECT, gm;
initgraph(&gd,&gm," ");
settextstyle(1,0,2);
outtextxy(100,100,"Enter number here : ");
scanf("%d",&num);
outtextxy(100,200,"The number entered is : %d",num);
getch();
closegraph();
}
```

The input function in computer graphics works in the windows mode, if you draw buttons. Association of a callback() function with a specific type of event mostly used in windowing system. The declaration in C++ is as follows:

```
int main (...)
{
....glutMouseFunc(mouse);
//Input function associated with input device mouse
glutDisplayFunc(...);
...}
void mouse_callback_func(int button, int state, int x,
int y)
{
 if (button ==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
drawSquare(x,y);
//users function
if (button ==GLUT_MIDDLE_BUTTON && state==GLUT_DOWN)
exit ();
} /* window reshape principle */
```

2.2.14 Interactive Picture Construction Techniques

Before studying the techniques for picture construction, a brief survey of various interaction devices is essential. There are three types of instruction devices, which are discussed as follows:

1. Locator Devices

The locator devices are further classified according to their independent characteristics.

(*i*) ***Absolute devices,*** such as touch panel, have a reference frame or origin and the positions are reported with respect to this origin.

(*ii*) **Relative devices,** such as mouse, have no absolute origin and report only changes from their previous position.

(*iii*) **Direct devices,** such as light pen, can be used as the user points at the screen directly with a finger or surrogate finger.

(*iv*) **With an Indirect devices,** such as a mouse, help the user move a cursor on the screen using a device, not directly interacting on the screen.

NOTES

(*v*) **Continuous devices** have the capability to create a smooth cursor motion, e.g., mouse.

(*vi*) **Discrete devices,** such as cursor control keys (up arrow, down arrow, etc.) do not offer smooth position change.

2. Keyboard Devices

The typewriter type keyboard is in use for a long time. This type of keyboard generally slows down the process of typing. Sometimes alphabetically organized keyboards are used by non-typists.

3. Choice Devices

Function keys are the most common choice devices.

Techniques of Picture Construction

There are many techniques for constructing pictures. These techniques are discussed as follows:

1. Rubber-Band Technique

The rubber-band technique is employed extensively in graphics packages for drawing graphic entities using the mouse only. In this technique, when the user clicks the mouse button, (say left button), the starting position is established. As the user moves the mouse, the cursor moves from the first point to the second, and the program displays a line from the first end point to the current point or a circle centered at the first end point and passing through the current point or a rectangle with one corner at the first point and the other corner at the current point, according to the entity choosen by the user for drawing.

In this method, the user generally has to *drag* the mouse pointer, i.e., to move the mouse cursor while the button is down. The entity shape and/or size is dynamically changed with every mouse movement while the button is down. When the button is released, the end point is frozen and the figure is there in its final shape, size and position.

Rubber-Band Algorithm

1. do
   GetMousePosition (&button , &x , &y)†
   while (NOT (button = 1 and MousePointer in WorkArea)
   : Thus we run a do-while loop until a left button is clicked and a corresponding
   x and y is obtained.
2. x_1 = previousx_2 = x
   y_1 = previousy_2 = y x_1 and y_1 is the starting point for the Rubber band.
3. do
   GetMousePosition(&button , &x, &y)
   if MousePosition not equal to x_1 & y_1 then
   previousx_2 = x_2
   previousy_2 = y_2

NOTES

$x_2 = x$

$y_2 = y$: This ensures previous x_2 & y_2 are stored in previousx_2 and previousy_2. The current value of MousePosition is stored in x_2 and y_2

If mouse moved from previous position, i.e., MousePosition not equal to x_2 & y_2 then

If previously a shape had been drawn (which is ensured by a restore flag (i.e.,) if RestoreFlag = 1) then

It is restored in Workarea from stack by calling function RestoreFlag().

The corresponding shape is drawn with corresponding color with appropriate parameters

Parameter for:

Line – x_1, y_1 as starting point
x_2, y_2 as end point

Rectangle – x_1, y_1 as Left-top corner
x_2, y_2 as Right-bottom corner.

Square – x_1, y_1 as Left-top corner
Side = Minimum($x_2 - x_1, y_2 - y_1$).

Circle diameter – x_1, y_1 and x_2, y_2 are two ends of the of the circle.
Centrex = $(x_1 + x_2)/2$.
Centrey = $(y_1 + y_2/2)$.
Radius = SquareRoot($(x_2 - \text{centrex})^2 + (y_2 - \text{centrey})^2$).

Ellipse diameter – x_1, y_1 and x_2, y_2 are two ends of the of the ellipse.
Centrex = $(x_1 + x_2)/2$.
Centrey = $(y_1 + y_2/2)$.
Radiusx = Absolute($(x_2 - x_1)/2$).
Radiusy = Absolute($(y_2 - y_1)/2$).
Restoreflag = 1

The algorithms to initialize stack, push to stack, pop from stack and clear stack are implemented by pointers to structures using usual data structures.

2. Constraint Technique

A *constraint* is a rule that we wish certain input information, in this case the input coordinates, to obey. We enforce the constraints by applying a transformation to the input coordinates, generating a new pair of coordinates that satisfy the constraint.

The most common form of constraint is the *modular constraint*, which forces the input point to the nearest intersection on a *grid*. Another useful and relatively simple constraint is the *directional constraint* applied to straight lines.

Some applications use only horizontal and vertical lines and are easier to operate if all input lines are constrained to be either horizontal or vertical.

3. Gravity Field Technique

NOTES

While the figures are constructed, it is sometimes necessary to connect lines at positions between end points. The exact positioning of the screen cursor at the connecting point is sometimes very difficult, so any input position near a line is to be converted to a position on the line. Creating a *gravity field* area around the line does this. Any selected position within the gravity field of a line is converted to the nearest position on the line. The size of the gravity field is chosen large enough to aid positioning, but small enough to minimize the chance of overlap with other lines.

4. Painting Technique

A raster display incorporating a random access frame buffer can be treated as a painting surface for interactive purpose. As the user moves the cursor around (by dragging), a trace of its path can be left on the screen.

Freehand-drawing algorithm

Left - click the mouse in workarea for the first point

GetMousePosition(button, x, y)

previous-x = x

previous-y = y

while left button down

GetMousePosition(button, x, y)

If x not equal to previous-x or y not equal to previous-y then

Draw Line(previous-x, previous-y, x, y) using Bresenham's algorithm

previous-x = x

previous-y = y

end if

end while

It is possible to provide a range of tools for painting on a raster display. These tools take the form of *brushes* that lay down the trails of different thickness and colour.

Similar to brushes, *eraser* tools are necessary to complement painting on a raster display as erasers are required along with the pencil for drawing on paper. These tools use the same logic as brush-painting; the only difference is that painting is done with background colour.

Brush-drawing algorithm (n pixel thick brush)

Left-click the mouse to select the first point, say $p_1 = (x_1, y_1)$.

While left button down

GetMousePosition(button, x, y)

NOTES

$x_2 = x$

$y_2 = y$

Trace all points joining (x_1, y_1) and (x_2, y_2), i.e., without corresponding Setpixels using Bresenham's algorithm.

For each point (x_k, y_k) in the traced line

Draw circle with radius = n pixels and centre = (x_k, y_k) with current Forecolor and fill circle with current Forecolor.

$x_1 = x_2$.

$y_1 = y_2$.

Eraser-algorithm (n pixel thick eraser)

Left-click the mouse to select the first point, say $p_1 = (x_1, y_1)$.

while left button down

GetMousePosition(button, x, y)

$x_2 = x$

$y_2 = y$

Trace all points joining (x_1, y_1) and (x_2, y_2), i.e., without corresponding Setpixels using Bresenham's algorithm.

for each point (x_k, y_k) in the traced line

Draw square with side = n pixels and intersection of diagonals = (x_k, y_k) with default Backcolor.

$x_1 = x_2$

$y_1 = y_2$

5. Selection Technique

To move (transform), delete or copy a part or whole of an element on the screen, the user should have the ability of selecting the desired element.

The following techniques can be used to help the user for achieving unambiguous selection:

- **The use of selection point:** In order to select an element, the user points a specific spot, such as centre of a circle or an end point of a line.
- **Defining a bounding rectangle:** The user can define two opposite corners of a rectangle and in this way select an element that lies within the rectangle. This technique is most useful for multiple element selection.

Algorithm for 'select' (rectangle)

1. The Selected area is marked with Dotted rectangle with color = Backcolor.

2. Open file 'TEMP.PIC' in WRITE mode. This ensures that the most recent selected portion is in file 'TEMP.PIC'

for each row in Selected area

for each column in Selected area

Convert the color value integer to ASCII character by adding 48† with it.

Write the ASCII character in file 'TEMP.PIC'.

3. Close file 'TEMP.PIC'

Algorithm for 'Select All'

The algorithm is same as in case of 'select' except that the selected area is the entire workarea and everything drawn on the screen is selected.

- Multiple keys for selection: When the user has positioned the cursor over the item he wishes to select, he can press one of several keys according to the type of item, i.e., one key to select a line, another to select a circle, and so on.
- Prefix commands: The command is given before the selection and may specify which type of item is to be selected. For example, there may be three different DELETE commands, DELETE LINE, DELETE CIRCLE and DELETE RECTANGLE.

Algorithm for Cut

Select a rectangular portion in the workarea using Select algorithm

Setpixel(Column, Row, Backcolor): To clear Selected area with Backcolor

Algorithm for Copy

Algorithm for copy is similar to Select algorithm as this algorithm copies the information of the selected area in the file 'TEMP.PIC'

Algorithm for Paste

1. Left-click on workarea to select top-left corner of the new position in which to be pasted.
2. Open 'TEMP.PIC' in READ mode.
3. for each row

   for each column

   Read an ASCII character from 'TEMP.PIC'

   Convert it to integer (color value) by subtracting 48 from the ASCII code

   Setpixel (column, row, color)
4. Close 'TEMP.PIC'

Algorithm for Delete

Select a rectangular portion without saving information in 'TEMP.PIC'

for each row in Selected area

for each column in Selected area

Setpixel(Column, Row, Backcolor). : To clear Selected area with Backcolor

Check Your Progress

1. Define a trackball.
2. What is a data glove?
3. Define the term raster line.
4. What are the various types of locator devices?
5. How does the rubber-band technique of picture construction work?

2.3 MATRICES

What is a Matrix?

Let F be a an array and n, m be two integers ≥ 1. An array of elements F of the type

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{pmatrix}$$

is called a matrix F. We denote this matrix by (a_{ij}), $i = 1, ..., m$ and $j = 1, ..., n$. We say that it is a $m \times n$ matrix (or matrix of order $m \times n$). It has m rows and n columns. For example, the first row is $(a_{11}\ a_{12} ..., a_{1n})$ and first column is,

$$\begin{pmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{pmatrix}$$

Also, a_{ij} denotes the element of the matrix (a_{ij}) lying in ith row and jth column and we call this element as the (i, j)th element of the matrix.

For example, in the matrix,

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$a_{11} = 1$, $a_{12} = 2$, $a_{32} = 8$, i.e., (1, 1)th element is 1
(1, 2)th element is 2
(3, 2)th element is 8

Notes:

1. Unless otherwise stated, we shall consider matrices over the field C of complex numbers only.
2. A matrix is simply an arrangement of elements and has no numerical value.

Example 2.1: If $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 0 & 1 & 2 \end{pmatrix}$, find $a_{11}, a_{22}, a_{33}, a_{31}, a_{41}$.

Solution: a_{11} = Element of A in first row and first column = 1

a_{22} = Element of A in second row and second column = 5

a_{33} = Element of A in third row and third column = 9

a_{31} = Element of A in third row and first column = 7

a_{41} = Element of A in fourth row and first column = 0

Types of Matrices

1. ***Row Matrix:*** A matrix which has exactly one row is called a *row matrix.* For example, (1 2 3 4) is a row matrix.
2. ***Column Matrix:*** A matrix which has exactly one column is called a *column matrix*.

   For example, $\begin{pmatrix} 5 \\ 6 \\ 7 \end{pmatrix}$ is a column matrix.

3. ***Square Matrix:*** A matrix in which the number of rows is equal to the number of columns is called a *square matrix*.

   For example, $\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ is a 2×2 square matrix.

4. ***Null or Zero Matrix:*** A matrix each of whose elements is zero is called a *null matrix* or *zero matrix.*

   For example, $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ is a 2×3 Null matrix.

5. ***Diagonal Matrix:*** The elements a_{ij} are called diagonal elements of a *square matrix* (a_{ij}). For example, $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ in matrix,

   the diagonal elements are $a_{11} = 1, a_{22} = 5, a_{33} = 9$

   A square matrix whose every element other than diagonal elements is zero, is called a *diagonal matrix*. For example,

   $$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$ is a diagonal matrix.

   Note that, the diagonal elements in a diagonal matrix may also be zero. For example,

   $\begin{pmatrix} 0 & 0 \\ 0 & 2 \end{pmatrix}$ and $\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$ are also diagonal matrices.

NOTES

6. ***Scalar Matrix:*** A diagonal matrix whose diagonal elements are equal, is called a *scalar matrix*. For example,

$$\begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}, \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \text{ are scalar matrices.}$$

7. ***Identity Matrix:*** A diagonal matrix whose diagonal elements are all equal to 1 (unity) is called *identity matrix or (unit matrix)*. For example,

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ is an identity matrix.}$$

8. ***Triangular Matrix:*** A square matrix (a_{ij}), whose elements $a_{ij} = 0$ when $i < j$ is called a *lower triangular matrix.*

   Similarly, a square matrix (a_{ij}) whose elements $a_{ij} = 0$ whenever $i > j$ is called an *upper triangular matrix.*

   For example,

$$\begin{pmatrix} 1 & 0 & 0 \\ 4 & 5 & 0 \\ 7 & 8 & 9 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 2 & 0 \end{pmatrix} \text{ are lower triangular matrices}$$

$$\text{and } \begin{pmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{pmatrix}, \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix} \text{ are upper triangular matrices.}$$

Submatrices of a Matrix: (Definition)

A submatrix of A is a matrix formed by selecting from A: a subset of the rows and: a subset of the columns and forming a new matrix by using those entries, in the same relative positions, that appear in both the rows and columns of those selected.

2.3.1 Equality of Two Matrices

Equality

Two matrices *A* and *B* are said to be equal if,

(*i*) *A* and *B* are of same order.

(*ii*) Corresponding elements in *A* and *B* are same. For example, the following two matrices are equal.

$$\begin{pmatrix} 3 & 4 & 9 \\ 16 & 25 & 64 \end{pmatrix} = \begin{pmatrix} 3 & 4 & 9 \\ 16 & 25 & 64 \end{pmatrix}$$

But the following two matrices are not equal.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

As matrix on left is of order 2 × 3, while on right it is of order 3 × 3

The following two matrices are also not equal.

$$\begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 8 & 9 \end{pmatrix}$$

As (2, 1)th element in LHS matrix is 7 while in RHS matrix it is 4

2.3.2 Addition of Matrices

If A and B are two matrices of the same order then addition of A and B is defined to be the matrix obtained by adding the corresponding elements of A and B.

For example, if

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, B = \begin{pmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{pmatrix}$$

$$\text{Then, } A + B = \begin{pmatrix} 1+2 & 2+3 & 3+4 \\ 4+5 & 5+6 & 6+7 \end{pmatrix} = \begin{pmatrix} 3 & 5 & 7 \\ 9 & 11 & 13 \end{pmatrix}$$

$$\text{Also, } A - B = \begin{pmatrix} 1-2 & 2-3 & 3-4 \\ 4-5 & 5-6 & 6-7 \end{pmatrix} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Note that addition (or subtraction) of two matrices is defined only when A and B are of the same order.

Properties of Matrix Addition

(*i*) Matrix addition is commutative.

i.e., $A + B = B + A$

For, (i, j)th element of $A + B$ is $(a_{ij} + b_{ij})$ and of $B + A$ is $(b_{ij} + a_{ij})$, and they are same as a_{ij} and b_{ij} are real numbers.

(*ii*) Matrix addition is associative,

i.e., $A + (B + C) = (A + B) + C$

For, (i, j)th element of $A + (B + C)$ is $a_{ij} + (b_{ij} + c_{ij})$ and of $(A + B) + C$ is $(a_{ij} + b_{ij}) + c_{ij}$ which are same.

(*iii*) If O denotes null matrix of the same order as that of A then,

$A + O = A = O + A$

For (i, j)th element of $A + O$ is $a_{ij} + O$ which is same as (i, j)th element of A.

(*iv*) To each matrix A there corresponds a matrix B such that,

$A + B = O = B + A$.

For, let (i, j)th element of B be $-a_{ij}$. Then (i, j)th element of $A + B$ is, $a_{ij} - a_{ij} = 0$.

Thus, the set of $m \times n$ matrices forms an abelian group under the composition of matrix addition.

NOTES

Example 2.2: If $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ and $B = \begin{pmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \end{pmatrix}$

Verify $A + B = B + A$.

Solution: $A + B = \begin{pmatrix} 1+0 & 2+1 & 3+2 \\ 4+3 & 5+4 & 6+5 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$

$$B + A = \begin{pmatrix} 0+1 & 1+2 & 2+3 \\ 3+4 & 4+5 & 5+6 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$$

So, $A + B = B + A$

Example 2.3: If A and B are matrices as in Example 3.2

and $C = \begin{pmatrix} -1 & 0 & 1 \\ 1 & 2 & 3 \end{pmatrix}$, verify $(A + B) + C = A + (B + C)$.

Solution: Now $A + B = \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 11 \end{pmatrix}$

So, $(A + B) + C = \begin{pmatrix} 1-1 & 3+0 & 5+1 \\ 7+1 & 9+2 & 11+3 \end{pmatrix} = \begin{pmatrix} 0 & 3 & 6 \\ 8 & 11 & 14 \end{pmatrix}$

Again, $B + C = \begin{pmatrix} 0-1 & 1+0 & 2+1 \\ 3+1 & 4+2 & 5+3 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 3 \\ 4 & 6 & 8 \end{pmatrix}$

So, $A + (B + C) = \begin{pmatrix} 1-1 & 2+1 & 3+3 \\ 4+4 & 5+6 & 6+8 \end{pmatrix} = \begin{pmatrix} 0 & 3 & 6 \\ 8 & 11 & 14 \end{pmatrix}$

Therefore, $(A + B) + C = A + (B + C)$

Example 2.4: If $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$, find a matrix B such that $A + B = 0$.

Solution: Let, $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$

Then, $A + B = \begin{pmatrix} 1+b_{11} & 2+b_{12} \\ 3+b_{21} & 4+b_{22} \\ 5+b_{31} & 6+b_{32} \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$

It implies, $b_{11} = -1$, $b_{12} = -2$, $b_{21} = -3$, $b_{22} = -4$,

$b_{31} = -5$, $b_{32} = -6$

Therefore required $B = \begin{pmatrix} -1 & -2 \\ -3 & -4 \\ -5 & -6 \end{pmatrix}$

2.3.3 Scalar Multiplication

If k is any complex number and A, a given matrix, then kA is the matrix obtained from A by multiplying each element of A by k. The number k is called *Scalar*.

For example, if

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \text{ and } k = 2$$

Then, $kA = \begin{pmatrix} 2 & 4 & 6 \\ 8 & 10 & 12 \end{pmatrix}$

It can be easily shown that,

(*i*) $k(A + B) = kA + kB$ (*ii*) $(k_1 + k_2)A = k_1A + k_2A$

(*iii*) $1A = A$ (*iv*) $(k_1k_2)A = k_1(k_2A)$

Example 2.5: (*i*) If $A = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 3 & 4 \\ 4 & 5 & 6 \end{pmatrix}$ and $k_1 = i$, $k_2 = 2$, verify,

$$(k_1 + k_2)\, A = k_1A + k_2A$$

(*ii*) If $A = \begin{pmatrix} 0 & 2 & 3 \\ 2 & 1 & 4 \end{pmatrix}$, $B = \begin{pmatrix} 7 & 6 & 3 \\ 1 & 4 & 5 \end{pmatrix}$, find the value of $2A + 3B$.

Solution: (*i*) Now $k_1A = \begin{pmatrix} 0 & i & 2i \\ 2i & 3i & 4i \\ 4i & 5i & 6i \end{pmatrix}$ and $k_2A = \begin{pmatrix} 0 & 2 & 4 \\ 4 & 6 & 8 \\ 8 & 10 & 12 \end{pmatrix}$

So, $k_1A + k_2A = \begin{pmatrix} 0 & 2+i & 4+2i \\ 4+2i & 6+3i & 8+4i \\ 8+4i & 10+5i & 12+6i \end{pmatrix}$

Also, $(k_1 + k_2)\, A = \begin{pmatrix} 0 & 2+i & 4+2i \\ 4+2i & 6+3i & 8+4i \\ 8+4i & 10+5i & 12+6i \end{pmatrix}$

Therefore, $(k_1 + k_2)A = k_1A + k_2A$

(*ii*) $2A = \begin{pmatrix} 0 & 4 & 6 \\ 4 & 2 & 8 \end{pmatrix}$

$$3B = \begin{pmatrix} 21 & 18 & 9 \\ 3 & 12 & 15 \end{pmatrix}$$

So, $2A + 3B = \begin{pmatrix} 21 & 22 & 15 \\ 7 & 14 & 23 \end{pmatrix}$

Example 2.6: If $A = \begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix}$ find $A^2 + 3A + 5I$ where I is unit matrix of order 2.

Solution: $A^2 = \begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ -3 & 0 \end{pmatrix} = \begin{pmatrix} -5 & 2 \\ -3 & -6 \end{pmatrix}$

$$3A = \begin{pmatrix} 3 & 6 \\ -9 & 0 \end{pmatrix}$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\ 5I = \begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}$$

So, $A^2 + 3A + 5I = \begin{pmatrix} -5 & 2 \\ -3 & -6 \end{pmatrix} + \begin{pmatrix} 3 & 6 \\ -9 & 0 \end{pmatrix} + \begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}$

$$= \begin{pmatrix} 3 & 8 \\ -12 & -1 \end{pmatrix}$$

NOTES

Example 2.7: If $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$ show that, $AB = -BA$ and $A^2 = B^2 = I$.

Solution: Now, $AB = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \begin{pmatrix} i & 0 \\ 0 & -i \end{pmatrix}$

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

So, $AB = -BA$

Also, $A^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$

$$B^2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

This proves the result.

Example 2.8: In an examination of Mathematics, 20 students from college *A*, 30 students from college *B* and 40 students from college *C* appeared. Only 15 students from each college could get through the examination. Out of them 10 students from college *A* and 5 students from college *B* and 10 students from college *C* secured full marks. Write down the above data in matrix form.

Solution: Consider the matrix,

$$\begin{pmatrix} 20 & 30 & 40 \\ 15 & 15 & 15 \\ 10 & 5 & 10 \end{pmatrix}$$

First row represents the number of students in college *A*, college *B*, college *C* respectively.

Second row represents the number of students who got through the examination in three colleges respectively.

Third row represents the number of students who got full marks in the three colleges respectively.

Example 2.9: A publishing house has two branches. In each branch, there are three offices. In each office, there are 3 peons, 4 clerks and 5 typists. In one office of a branch, 6 salesmen are also working. In each office of other branch 2 head-clerks are also working. Using matrix notation find (*i*) the total number of posts of each kind in all the offices taken together in each branch, (*ii*) the total number of posts of each kind in all the offices taken together from both the branches.

Solution: (*i*) Consider the following row matrices,

$$A_1 = (3\ \ 4\ \ 5\ \ 6\ \ 0),\quad A_2 = (3\ \ 4\ \ 5\ \ 0\ \ 0),\quad A_3 = (3\ \ 4\ \ 5\ \ 0\ \ 0)$$

These matrices represent the three offices of the branch (say *A*) where elements appearing in the row represent the number of peons, clerks, typists, salesmen and head-clerks taken in that order working in the three offices.

Then, $A_1 + A_2 + A_3 = (3+3+3\ \ 4+4+4\ \ 5+5+5\ \ 6+0+0\ \ 0+0+0)$

$$= (9\ \ 12\ \ 15\ \ 6\ \ 0)$$

Thus, total number of posts of each kind in all the offices of branch A are the elements of matrix $A_1 + A_2 + A_3 = (9 \;\; 12 \;\; 15 \;\; 6 \;\; 0)$

Now consider the following row matrices,

$$B_1 = (3 \;\; 4 \;\; 5 \;\; 0 \;\; 2), \quad B_2 = (3 \;\; 4 \;\; 5 \;\; 0 \;\; 2), \quad B_3 = (3 \;\; 4 \;\; 5 \;\; 0 \;\; 2)$$

Then B_1, B_2, B_3 represent three offices of other branch (say B) where the elements in the row represents number of peons, clerks, typists, salesmen and head-clerks respectively.

Thus, total number of posts of each kind in all the offices of branch B are the elements of the matrix $B_1 + B_2 + B_3 = (9 \;\; 12 \;\; 15 \;\; 0 \;\; 6)$

(*ii*) The total number of posts of each kind in all the offices taken together from both branches are the elements of matrix,

$$(A_1 + A_2 + A_3) + (B_1 + B_2 + B_3) = (18 \;\; 24 \;\; 30 \;\; 6 \;\; 6)$$

Example 2.10: Let $A = \begin{pmatrix} 10 & 20 \\ 30 & 40 \end{pmatrix}$ where first row represents the number of table fans and second row represents the number of ceiling fans which two manufacturing units A and B make in one day. The first and second column represent the manufacturing units A and B. Compute $5A$ and state what it represents.

Solution: $5A = \begin{pmatrix} 50 & 100 \\ 150 & 200 \end{pmatrix}$

It represents the number of table fans and ceiling fans that the manufacturing units A and B produce in five days.

Example 2.11: Let $A = \begin{pmatrix} 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{pmatrix}$ where rows represent the number of items of type I, II, III, respectively. The four columns represents the four shops A_1, A_2, A_3, A_4 respectively.

Let, $B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \end{pmatrix}$, $C = \begin{pmatrix} 1 & 2 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 4 \end{pmatrix}$

Where elements in B represent the number of items of different types delivered at the beginning of a week and matrix C represent the sales during that week. Find,

(*i*) The number of items immediately after delivery of items.

(*ii*) The number of items at the end of the week.

(*iii*) The number of items needed to bring stocks of all items in all shops to 6.

Solution: (*i*) $A + B = \begin{pmatrix} 3 & 5 & 7 & 9 \\ 5 & 5 & 7 & 9 \\ 7 & 7 & 7 & 9 \end{pmatrix}$

Represent the number of items immediately after delivery of items.

(*ii*) $(A + B) - C = \begin{pmatrix} 2 & 3 & 5 & 6 \\ 4 & 3 & 4 & 5 \\ 5 & 4 & 3 & 5 \end{pmatrix}$

Represent the number of items at the end of the week.

(*iii*) We want that all elements in $(A + B) - C$ should be 6.

NOTES

Let $D = \begin{pmatrix} 4 & 3 & 1 & 0 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 1 \end{pmatrix}$

Then $(A + B) - C + D$ is a matrix in which all elements are 6. So, D represents the number of items needed to bring stocks of all items of all shops to 6.

Example 2.12: The following matrix represents the results of the examination of *B*. Com. class:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

The rows represent the three sections of the class. The first three columns represent the number of students securing 1st, 2nd, 3rd divisions respectively in that order and fourth column represents the number of students who failed in the examination.

(*i*) How many students passed in three sections respectively?

(*ii*) How many students failed in three sections respectively?

(*iii*) Write down the matrix in which number of successful students is shown.

(*iv*) Write down the column matrix where only failed students are shown.

(*v*) Write down the column matrix showing students in 1st division from three sections.

Solution: (*i*) The number of students who passed in three sections respectively are 1 + 2 + 3 = 6, 5 + 6 + 7 = 18, 9 + 10 + 11 = 30.

(*ii*) The number of students who failed from three sections respectively are 4, 8, 12.

(*iii*) $\begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ 9 & 10 & 11 \end{pmatrix}$

(*iv*) $\begin{pmatrix} 4 \\ 8 \\ 12 \end{pmatrix}$ represents column matrix where only failed students are shown.

(*v*) $\begin{pmatrix} 1 \\ 5 \\ 9 \end{pmatrix}$ represents column matrix of students securing 1st division.

2.3.4 Multiplication of Two Matrices

The product AB of two matrices A and B is defined only when the number of columns of A is same as the number of rows in B and by definition the product AB is a matrix G of order $m \times p$ if A and B were of order $m \times n$ and $n \times p$, respectively. The following example will give the rule to multiply two matrices:

Let, $A = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$ $B = \begin{pmatrix} d_1 & e_1 \\ d_2 & e_2 \\ d_3 & e_3 \end{pmatrix}$

Order of $A = 2 \times 3$, Order of $B = 3 \times 2$

So, AB is defined as,

$$G = AB = \begin{pmatrix} a_1d_1 + b_1d_2 + c_1d_3 & a_1e_1 + b_1e_2 + c_1e_3 \\ a_2d_1 + b_2d_2 + c_2d_3 & a_2e_1 + b_2e_2 + c_2e_3 \end{pmatrix}$$
$$= \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

g_{11} : Multiply elements of the first row of A with corresponding elements of the first column of B and add.

g_{12} : Multiply elements of the first row of A with corresponding elements of the second column of B and add.

g_{21} : Multiply elements of the second row of A with corresponding elements of the first column of B and add.

g_{22} : Multiply elements of the second column of A with corresponding elements of the second column and add.

Notes: 1. In general, if A and B are two matrices then AB may not be equal to BA. For example, if

$A = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ then $AB = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

and $BA = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$. So, $AB \neq BA$

2. If product AB is defined, then it is not necessary that BA must also be defined. For example, if A is of order 2×3 and B is of order 3×1, then AB can be defined but BA cannot be defined (as the number of columns of $B \neq$ the number of rows of A).

It can be easily verified that,

(*i*) $A(BC) = (AB)C$.

(*ii*) $A(B + C) = AB + AC$.

(*iii*) $(A + B)C = AC + BC$.

Example 2.13: If $A = \begin{pmatrix} 2 & -1 \\ 0 & 3 \end{pmatrix}$ and $B = \begin{pmatrix} 7 & 0 \\ -2 & -3 \end{pmatrix}$ write down AB.

Solution:
$$AB = \begin{pmatrix} 2\times 7 + (-1)\times(-2) & 2\times 0 + (-1)\times(-3) \\ 0\times 7 + 3\times(-2) & 0\times 0 + 3\times(-3) \end{pmatrix}$$
$$= \begin{pmatrix} 16 & 3 \\ -6 & -9 \end{pmatrix}$$

Example 2.14: Verify the associative law $A(BC) = (AB)C$ for the following matrices.

$$A = \begin{pmatrix} -1 & 0 \\ 7 & -2 \end{pmatrix}, \quad B = \begin{pmatrix} -1 & 5 \\ 7 & 0 \end{pmatrix}, \quad C = \begin{pmatrix} -1 & -1 \\ 2 & 0 \end{pmatrix}$$

Solution:
$$AB = \begin{pmatrix} -1 & 0 \\ 7 & -2 \end{pmatrix}\begin{pmatrix} -1 & 5 \\ 7 & 0 \end{pmatrix} = \begin{pmatrix} 1+0 & -5+0 \\ -7-14 & 35+0 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & -5 \\ -21 & 35 \end{pmatrix}$$

$$BC = \begin{pmatrix} -1 & 5 \\ 7 & 0 \end{pmatrix}\begin{pmatrix} -1 & -1 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 1+10 & 1+0 \\ -7+0 & -7+0 \end{pmatrix} = \begin{pmatrix} 11 & 1 \\ -7 & -7 \end{pmatrix}$$

$$A(BC) = \begin{pmatrix} -1 & 0 \\ 7 & -2 \end{pmatrix}\begin{pmatrix} 11 & 1 \\ -7 & -7 \end{pmatrix} = \begin{pmatrix} -11+0 & -1+0 \\ 77+14 & 7+14 \end{pmatrix}$$

$$= \begin{pmatrix} -11 & -1 \\ 91 & 21 \end{pmatrix}$$

$$(AB)C = \begin{pmatrix} 1 & -5 \\ -21 & 35 \end{pmatrix}\begin{pmatrix} -1 & -1 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} -1-10 & -1+0 \\ 21+70 & 21+0 \end{pmatrix}$$

$$= \begin{pmatrix} -11 & -1 \\ 91 & 21 \end{pmatrix}$$

Thus, $A(BC) = (AB)C$

Example 2.15: If A is a square matrix, then A can be multiplied by itself. Define $A^2 = A.A$ which is called power of a matrix. Compute A^2 for the following matrix:

$$A = \begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}$$

Solution: $$A^2 = \begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 15 & 16 \end{pmatrix}$$

Similarly, we can define $A^3, A^4, A^5, \ldots$ for any square matrix A.

2.3.5 Transpose of a Matrix

Let A be a matrix. The matrix obtained from A by interchange of its rows and columns, is called the *transpose* of A. For example,

If, $A = \begin{pmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \end{pmatrix}$ then transpose of $A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 2 & 0 \end{pmatrix}$

Transpose of A is denoted by A'.

It can be easily verified that,

(*i*) $(A')' = A$

(*ii*) $(A + B)' = A' + B'$

(*iii*) $(AB)' = B'A'$

Example 2.16: For the following matrices A and B verify $(A + B)' = A' + B'$.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 2 & 3 & 4 \\ 1 & 8 & 6 \end{pmatrix}$$

Solution: $$A' = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \quad B' = \begin{pmatrix} 2 & 1 \\ 3 & 8 \\ 4 & 6 \end{pmatrix}$$

So, $$A' + B' = \begin{pmatrix} 3 & 5 \\ 5 & 13 \\ 7 & 12 \end{pmatrix}$$

Again, $A + B = \begin{pmatrix} 3 & 5 & 7 \\ 5 & 13 & 12 \end{pmatrix}$

So, $(A+B)' = \begin{pmatrix} 3 & 5 \\ 5 & 13 \\ 7 & 12 \end{pmatrix}$

Therefore, $(A+B)' = A' + B'$

2.4 SYMMETRIC AND SKEW-SYMMETRIC MATRICES

Symmetric and Skew-Symmetric Matrices

A square matrix $A = [a_{ij}]$ is called a symmetric matrix if $a_{ij} = a_{ji}$ for all i and j.

For example,

$$\begin{bmatrix} 2 & -1 & 4 \\ -1 & 0 & 3 \\ 4 & 3 & 1 \end{bmatrix}, \begin{bmatrix} 6 & 3 \\ 3 & 2 \end{bmatrix}$$

A square matrix $A = [a_{ij}]$ is said to be skew-symmetric if $a_{ij} = -a_{ji}$ for all i and j.

Remember that $a_{ii} = -a_{ii}$ (for $i = j$)

$\Rightarrow$ $a_{ii} = 0$, i.e., all the diagonal elements of a skew-symmetric matrix are zero.

For example,

$$\begin{bmatrix} 0 & -4 \\ 4 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 2 & -3 \\ -2 & 0 & 4 \\ 3 & -4 & 0 \end{bmatrix}$$

If A' denotes the transpose of a square matrix A, then A is symmetric when $A = A'$, and A is skew-symmetric when $A' = -A$.

2.4.1 Adjoint Matrix

The adjoint matrix of A is obtained by replacing the elements of A by their respective cofactors and then transposing.

If $A = [a_{ij}]$ and $B = [A_{ij}]$ where A_{ij} is the cofactor of a_{ij} in A then we have the adjoint matrix of A, written as

$$Adj\ A = [A_{ij}]' = [A_{ji}]$$

If $$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, Adj\ A = \begin{bmatrix} A_{11} & A_{21} & A_{23} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{bmatrix}$$

where $$A_{11} = + \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}, A_{12} = - \begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix}, \text{ etc.}$$

NOTES

Example 2.17: Find the adjoint of the following given matrix.

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

Solution:

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

$$A_{11} = +\begin{vmatrix} 2 & 2 \\ 5 & 1 \end{vmatrix} = 4 - 5 = -1$$

$$A_{12} = -\begin{vmatrix} 0 & 1 \\ -4 & 2 \end{vmatrix} = -4$$

$$A_{13} = +\begin{vmatrix} 0 & 2 \\ -4 & 5 \end{vmatrix} = 8$$

$$A_{21} = -\begin{vmatrix} -2 & 3 \\ 5 & 2 \end{vmatrix} = 19$$

$$A_{22} = +\begin{vmatrix} 1 & 3 \\ -4 & 2 \end{vmatrix} = 14$$

$$A_{23} = -\begin{vmatrix} 1 & -2 \\ -4 & 5 \end{vmatrix} = 3$$

$$A_{31} = +\begin{vmatrix} -2 & 3 \\ 2 & 1 \end{vmatrix} = -8$$

$$A_{32} = -\begin{vmatrix} 1 & 3 \\ 0 & 1 \end{vmatrix} = -1$$

$$A_{33} = +\begin{vmatrix} 1 & -2 \\ 0 & 2 \end{vmatrix} = 2$$

$$Adj\ A = \begin{bmatrix} A_{11} & A_{21} & A_{31} \\ A_{12} & A_{22} & A_{32} \\ A_{13} & A_{23} & A_{33} \end{bmatrix} = \begin{bmatrix} -1 & 19 & -8 \\ -4 & 14 & -1 \\ 8 & 3 & 2 \end{bmatrix}$$

Example 2.18: Find the adjoint of the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 1 & 0 & 6 \end{bmatrix}$$

Solution:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 1 & 0 & 6 \end{bmatrix}$$

$$A_{11}=\begin{vmatrix}4 & 5\\ 0 & 6\end{vmatrix}=24 \qquad A_{12}=-\begin{vmatrix}0 & 5\\ 1 & 6\end{vmatrix}=5 \qquad A_{13}=\begin{vmatrix}0 & 4\\ 1 & 0\end{vmatrix}=-4$$

$$A_{21}=-\begin{vmatrix}2 & 3\\ 0 & 6\end{vmatrix}=-12 \qquad A_{22}=\begin{vmatrix}1 & 3\\ 1 & 6\end{vmatrix}=3 \qquad A_{23}=-\begin{vmatrix}1 & 2\\ 1 & 0\end{vmatrix}=2$$

$$A_{31}=\begin{vmatrix}2 & 3\\ 4 & 5\end{vmatrix}=-2 \qquad A_{32}=-\begin{vmatrix}1 & 3\\ 0 & 5\end{vmatrix}=-5 \qquad A_{33}=\begin{vmatrix}1 & 2\\ 0 & 4\end{vmatrix}=4$$

$$Adj\ A = \begin{bmatrix}A_{11} & A_{21} & A_{31}\\ A_{12} & A_{22} & A_{32}\\ A_{13} & A_{23} & A_{33}\end{bmatrix}$$

Therefore,

$$Adj\ A = \begin{bmatrix}24 & -12 & -2\\ 5 & 3 & -5\\ -4 & 2 & 4\end{bmatrix}$$

2.4.2 Inverse of a Matrix

Consider the matrices,

$$A=\begin{pmatrix}2 & 0 & -1\\ 5 & 1 & 0\\ 0 & 1 & 3\end{pmatrix}, \quad B=\begin{pmatrix}3 & -1 & 1\\ -15 & 6 & -5\\ 5 & -2 & 2\end{pmatrix}$$

It can be easily seen that,

$$AB = BA = I \text{ (unit matrix)}$$

In this case, we say, B is inverse of A. Infact, we have the following definition.

'If A is a square matrix of order n, then a square matrix B of the same order n is said to be inverse of A if $AB = BA = I$ (unit matrix).'

***Notes*:**

1. Inverse of a matrix is defined only for square matrices.
2. If B is an inverse of A, then A is also an inverse of B. [Follows clearly by definition.]
3. If a matrix A has an inverse, then A is said to be invertible.
4. Inverse of a matrix is unique.

   For, let B and C be two inverses of A.

   Then, $AB = BA = I$ and $AC = CA = I$

   So, $B = BI = B(AC) = (BA)C = IC = C$

   ***Notation*:** Inverse of A is denoted by A^{-1}
5. Square matrix is not invertible.

NOTES

For, let $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

If A is invertible, let $B = \begin{pmatrix} x & x' \\ y & y' \end{pmatrix}$ be inverse of A.

Then $AB = I$ implies $\begin{pmatrix} x+y & x'+y' \\ x+y & x'+y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

$\Rightarrow\ x+y=1,\ x'+y'=0,\ x+y=0,\ x'+y'=1$, which is absurd.

This proves our assertion.

In the present section, we give a method to determine the inverse of a matrix. Consider the identity $A = IA$.

We reduce the matrix A on left hand side to the unit matrix I by elementary row operations only and apply all those operations in same order to the prefactor I on the right hand side of the above identity. In this way, unit matrix I is reduced to some matrix B such that $I = BA$. Matrix B is then the inverse of A.

We illustrate the above method by the following examples.

Example 2.19: Find the inverse of the matrix,

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

Solution: Consider the identity,

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

Applying $R_2 \to R_2 - R_1$, then $R_3 \to R_3 - R_1$, we have,

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

Applying $R_1 \to R_1 - 3R_2 - 3R_3$, we have,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 7 & -3 & -3 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & 3 \\ 1 & 4 & 3 \\ 1 & 3 & 4 \end{pmatrix}$$

So, the desired inverse is,

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

Example 2.20: Find the inverse of the matrix,

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

Solution: Consider the identity,

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

Applying $R_2 \to R_2 + 3R_1$, $R_3 \to R_3 - 2R_1$, we have,

$$\begin{pmatrix} 1 & 3 & -2 \\ 0 & 9 & -11 \\ 0 & -1 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ -3 & 0 & -5 \\ 2 & 5 & 0 \end{pmatrix}$$

Applying $R_3 \to 9R_3$ and then $R_3 \to R_3 + R_2$, we have,

$$\begin{pmatrix} 1 & 3 & -2 \\ 0 & 9 & -11 \\ 0 & 0 & 25 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -15 & 1 & 9 \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ -3 & 0 & -5 \\ 2 & 5 & 0 \end{pmatrix}$$

Applying $R_3 \to \frac{1}{25} R_3$, we have,

$$\begin{pmatrix} 1 & 3 & -2 \\ 0 & 9 & -11 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -\frac{3}{5} & \frac{1}{25} & \frac{9}{25} \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ -3 & 0 & -5 \\ 2 & 5 & 0 \end{pmatrix}$$

Applying $R_2 \to R_2 + 11R_3$, $R_1 \to R_1 + 2R_3$, we have,

$$\begin{pmatrix} 1 & 3 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{5} & \frac{2}{25} & \frac{18}{25} \\ -\frac{18}{5} & \frac{36}{25} & \frac{99}{25} \\ -\frac{3}{5} & \frac{1}{25} & \frac{9}{25} \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ -3 & 0 & -5 \\ 2 & 5 & 0 \end{pmatrix}$$

Applying $R_2 \to \frac{1}{9} R_2$, we have,

$$\begin{pmatrix} 1 & 3 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{5} & \frac{2}{25} & \frac{18}{25} \\ -\frac{2}{5} & \frac{4}{25} & \frac{11}{25} \\ -\frac{3}{5} & \frac{1}{25} & \frac{9}{25} \end{pmatrix} \begin{pmatrix} 1 & 3 & -2 \\ -3 & 0 & -5 \\ 2 & 5 & 0 \end{pmatrix}$$

Applying $R_1 \to R_1 - 3R_2$, we have,

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&-\frac{2}{5}&-\frac{3}{5}\\-\frac{2}{5}&\frac{4}{25}&\frac{11}{25}\\-\frac{3}{5}&\frac{1}{25}&\frac{9}{25}\end{pmatrix}\begin{pmatrix}1&3&-2\\-3&0&-5\\2&5&0\end{pmatrix}$$

So, the desired inverse is,

$$\begin{pmatrix}1&-\frac{2}{5}&-\frac{3}{5}\\-\frac{2}{5}&\frac{4}{25}&\frac{11}{25}\\-\frac{3}{5}&\frac{1}{25}&\frac{9}{25}\end{pmatrix}$$

Example 2.21: Find the inverse of the matrix $\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}$

Solution: Consider the identity,

$$\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}$$

Applying $R_2 \to R_2 + 4R_1$, $R_3 \to R_3 + 4R_1$, we have,

$$\begin{pmatrix}1&2&-1\\0&1&0\\0&-1&1\end{pmatrix}=\begin{pmatrix}1&0&0\\4&1&0\\4&0&1\end{pmatrix}\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}$$

Applying $R_1 \to R_1 + R_3$ then $R_3 \to R_3 + R_2$, we have,

$$\begin{pmatrix}1&1&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}5&0&1\\4&1&0\\8&1&1\end{pmatrix}\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}$$

Applying $R_1 \to R_1 - R_2$, we have

$$\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}=\begin{pmatrix}1&-1&1\\4&1&0\\8&1&1\end{pmatrix}\begin{pmatrix}1&2&-1\\-4&-7&4\\-4&-9&5\end{pmatrix}$$

So, the desired inverse is,

$$\begin{pmatrix}1&-1&1\\4&1&0\\8&1&1\end{pmatrix}$$

2.5 DETERMINANTS

If A is a square matrix with entries from the field of complex numbers, then determinant of A is some complex number. This will be denoted by det A or $|A|$. If

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

then det A will be denoted by

$$\begin{vmatrix} a_{11} & a_{12} & a_{1n} \\ a_{n1} & a_{n2} & a_{nn} \end{vmatrix}$$

NOTES

Notes: 1. det A or $|A|$ is defined for square matrix A only.

2. det A or $|A|$ will be defined in such a way that A is invertible if det $A \neq 0$.
3. The determinant of an $n \times n$ matrix will be called determinant of order n.

Determinant of Order One

Let $A = (a_{11})$ be a square matrix of order one. Then det $A = a_{11}$

By definition, if A is invertible, then $a_{11} \neq 0$ and so, det $A \neq 0$. Also, conversely if det $A \neq 0$, then $a_{11} \neq 0$ and so, A is invertible.

Determinant of Order Two

Let $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ be a square matrix of order two. Then we define

det $A = a_{11}a_{22} - a_{12}a_{21}$

For example, if $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ then det $A = 4 - 6 = -2$

Suppose $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is invertible.

Then by definition there exists a matrix

$B = \begin{pmatrix} x & y \\ z & w \end{pmatrix}$ where x, y, z, w are complex numbers such that $AB = I = BA$

The above identity implies,

$a_{11}x + a_{12}z = 1,\ a_{11}y + a_{12}w = 0$

$a_{21}x + a_{22}z = 0,\ a_{21}y + a_{22}w = 1$

which in turn implies

$$\Delta x = a_{22}, \quad \Delta y = -a_{12}$$
$$\Delta z = -a_{21}, \quad \Delta w = a_{11}$$

where $\Delta = a_{11}a_{22} - a_{12}a_{21}$

Clearly $\Delta \neq 0$, for otherwise x, y, z, w will be indeterminate. This means that det $A \neq 0$. Conversely, if A is a square matrix of order 2 such that det $A \neq 0$, then A is invertible as

$$x = \frac{a_{22}}{\Delta}, \quad y = \frac{-a_{12}}{\Delta}, \quad z = \frac{-a_{21}}{\Delta}, \quad w = \frac{a_{11}}{\Delta}$$

will determine B uniquely satisfying $AB = I = BA$

Determinant of Order Three

Let $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ be a 3×3 matrix.

Then we define $\det A = a_{11}(a_{22}a_{33} - a_{32}a_{23})$
$- a_{12}(a_{21}a_{33} - a_{31}a_{23})$
$+ a_{13}(a_{21}a_{32} - a_{31}a_{22})$

The above definition may be explained as follows:

The first bracket is determinant of matrix obtained after removing first row and first column.

The second bracket is determinant of matrix obtained after removing first row and second column.

The third bracket is determinant of matrix obtained after removing first row and third column.

The elements before three brackets are first, second, third element respectively of first row with alternate positive and negative signs.

For example, let $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$

To find $\det A$.

The first bracket in the definition of $\det A$ is determinant of

$$\begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix} = 45 - 48 = -3$$

The second bracket is determinant of

$$\begin{pmatrix} 4 & 6 \\ 7 & 9 \end{pmatrix} = 36 - 42 = -6$$

The third bracket is determinant of

$$\begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix} = 32 - 35 = -3$$

So, $\det A = 1(-3) - 2(-6) + 3(-3) = -3 + 12 - 9 = 0$

It can be seen that if A is a square matrix of order 3, then A is invertible if $\det A \neq 0$

Determinant of Order Four

Let $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$

Then we define $\det A = a_{11} \times \det \begin{pmatrix} a_{22} & a_{23} & a_{24} \\ a_{32} & a_{33} & a_{34} \\ a_{42} & a_{43} & a_{44} \end{pmatrix}$

$$-a_{12} \times \det \begin{pmatrix} a_{21} & a_{23} & a_{24} \\ a_{31} & a_{33} & a_{34} \\ a_{41} & a_{43} & a_{44} \end{pmatrix}$$

$$+ \ a_{13} \times \det \begin{pmatrix} a_{21} & a_{22} & a_{24} \\ a_{31} & a_{32} & a_{34} \\ a_{41} & a_{42} & a_{44} \end{pmatrix}$$

$$-a_{14} \times \det \begin{pmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix}$$

Remark: A determinant $\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}$ of order 2 can also be obtained when we eliminate x, y from $a_1x + b_1y = 0$, $a_2x + b_2y = 0$ provided one of x, y is non-zero. Similarly determinant of order 3 can be obtained by eliminating x, y, z from,

$$a_1x + b_1y + c_1z = 0$$
$$a_2x + b_2y + c_2z = 0$$
$$a_3x + b_3y + c_3z = 0$$

provided one of x, y, z is non-zero.

2.5.1 Properties of Determinants

We list below some imortant properties of determinants.

1. If two rows (or columns) are interchanged in a determinant it retains its absolute value but changes its sign.

   i.e. $$\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = -\begin{vmatrix} b_1 & b_2 & b_3 \\ a_1 & a_2 & a_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$$

2. If rows are changed into columns and columns into rows the determinant remains unchanged.

   i.e. $$\begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}$$

3. If two rows (or columns) are identical in a determinant it vanishes.

   i.e. $$\begin{vmatrix} a_1 & a_2 & a_3 \\ a_1 & a_2 & a_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = 0$$

4. If any row (or column) is multiplied by a complex number k, the determinant so obtained is k times the original determinant.

   i.e. $$\begin{vmatrix} a_1 & a_2 & a_3 \\ kb_1 & kb_2 & kb_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = k \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$$

5. If to any row (or column) is added k times the corresponding elements of another row (or column), the determinant remains unchanged.

NOTES

i.e. $\begin{vmatrix} a_1 + kb_1 & a_2 + kb_2 & a_3 + kb_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$

6. If any row (or column) is the sum of two or more elements, then the determinant can be expressed as sum of two or more determinants.

i.e. $\begin{vmatrix} a_1 + k_1 & a_2 + k_2 & a_3 + k_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} + \begin{vmatrix} k_1 & k_2 & k_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix}$

7. If determinant vanishes by putting $x = a$, then $(x - a)$ is a factor of the determinant.

e.g., $\begin{vmatrix} 1 & 1 & 1 \\ a & b & c \\ a^2 & b^2 & c^2 \end{vmatrix}$ has $(a - b)$ as one of its factors (by putting $a = b$, first and second columns become identical).

8. If k rows or columns become identical by putting $x = a$ then $(x - a)^{k-1}$ is a factor of the determinant.

For example, consider in the following determinant:

$$\begin{vmatrix} (b+c)^2 & a^2 & a^2 \\ b^2 & (c+a)^2 & b^2 \\ c^2 & c^2 & (a+b)^2 \end{vmatrix}$$

all the three rows become identical by putting $a + b + c = 0$. So, $(a + b + c)^2$ is one of the factors of the given determinant.

Example 2.22: *Show that*

$$\begin{vmatrix} 1 & a & b+c \\ 1 & b & c+a \\ 1 & c & a+b \end{vmatrix} = 0$$

Solution: Now

$$\begin{vmatrix} 1 & a & b+c \\ 1 & b & c+a \\ 1 & c & a+b \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 & 1 \\ a & b & c \\ b+c & c+a & a+b \end{vmatrix}$$ [interchanging rows and columns]

Applying $C_2 \to C_2 - C_1$, $C_3 \to C_3 - C_1$

$$= \begin{vmatrix} 1 & 0 & 0 \\ a & b-a & c-a \\ b+c & a-b & a-c \end{vmatrix}$$

$$= (a-b)(a-c)\begin{vmatrix} 1 & 0 & 0 \\ a & -1 & -1 \\ b+c & 1 & 1 \end{vmatrix} = 0, \text{ by property 3}$$

NOTES

Example 2.23: *Show that*

$$\begin{vmatrix} a-b & b-c & c-a \\ b-c & c-a & a-b \\ c-a & a-b & b-c \end{vmatrix} = 0$$

Solution: $\begin{vmatrix} a-b & b-c & c-a \\ b-c & c-a & a-b \\ c-a & a-b & b-c \end{vmatrix}$

Applying $R_1 \to R_1 + R_2 + R_3$

$$= \begin{vmatrix} 0 & 0 & 0 \\ b-c & c-a & a-b \\ c-a & a-b & b-c \end{vmatrix}$$

$$= 0$$

Example 2.24: *Prove that*

$$\begin{vmatrix} 1 & 1 & 1 \\ a & b & c \\ a^2 & b^2 & c^2 \end{vmatrix} = (a-b)(b-c)(c-a)$$

Solution: $\begin{vmatrix} 1 & 1 & 1 \\ a & b & c \\ a^2 & b^2 & c^2 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ a & b-a & c-a \\ a^2 & b^2-a^2 & c^2-a^2 \end{vmatrix}$

Applying $C_2 \to C_2 - C_1$ and $C_3 \to C_3 - C_1$

$$= (b-a)(c-a)\begin{vmatrix} 1 & 0 & 0 \\ a & 1 & 1 \\ a^2 & b+a & c+a \end{vmatrix}$$

$$= (b-a)(c-a)(c+a-b-a)$$

$$= (b-a)(c-a)(c-b)$$

$$= (a-b)(b-c)(c-a)$$

Example 2.25: *Prove that*

$$\begin{vmatrix} a-b-c & 2a & 2a \\ 2b & b-c-a & 2b \\ 2c & 2c & c-a-b \end{vmatrix} = (a+b+c)^3$$

Solution: $\begin{vmatrix} a-b-c & 2a & 2a \\ 2b & b-c-a & 2b \\ 2c & 2c & c-a-b \end{vmatrix}$

Applying $R_1 \to R_1 + R_2 + R_3$

NOTES

$$= \begin{vmatrix} a+b+c & a+b+c & a+b+c \\ 2b & b-c-a & 2b \\ 2c & 2c & c-a-b \end{vmatrix}$$

$$= (a+b+c)\begin{vmatrix} 1 & 1 & 1 \\ 2b & b-c-a & 2b \\ 2c & 2c & c-a-b \end{vmatrix}$$

Applying $C_2 \to C_2 - C_1$, $C_3 \to C_3 - C_1$

$$= (a+b+c)\begin{vmatrix} 1 & 0 & 0 \\ 2b & (a+b+c) & 0 \\ 2c & 0 & -(a+b+c) \end{vmatrix}$$

$$= (a+b+c)(a+b+c)^2 = (a+b+c)^3$$

Example 2.26: $\begin{vmatrix} 1+a & 1 & 1 \\ 1 & 1+b & 1 \\ 1 & 1 & 1+c \end{vmatrix} = abc\left(1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c}\right)$

Solution: $\begin{vmatrix} 1+a & 1 & 1 \\ 1 & 1+b & 1 \\ 1 & 1 & 1+c \end{vmatrix} = abc\begin{vmatrix} 1+\frac{1}{a} & \frac{1}{a} & \frac{1}{a} \\ \frac{1}{b} & 1+\frac{1}{b} & \frac{1}{b} \\ \frac{1}{c} & \frac{1}{c} & 1+\frac{1}{c} \end{vmatrix}$

Applying $R_1 \to R_1 + R_2 + R_3$

$$= abc\begin{vmatrix} 1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c} & 1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c} & 1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c} \\ \frac{1}{b} & 1+\frac{1}{b} & \frac{1}{b} \\ \frac{1}{c} & \frac{1}{c} & 1+\frac{1}{c} \end{vmatrix}$$

$$= abc\left(1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c}\right)\begin{vmatrix} 1 & 1 & 1 \\ \frac{1}{b} & 1+\frac{1}{b} & \frac{1}{b} \\ \frac{1}{c} & \frac{1}{c} & 1+\frac{1}{c} \end{vmatrix}$$

Applying $C_2 \to C_2 - C_1$, $C_3 \to C_3 - C_1$

$$= abc\left(1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c}\right)\begin{vmatrix} 1 & 0 & 0 \\ \frac{1}{b} & 1 & 0 \\ \frac{1}{c} & 0 & 1 \end{vmatrix}$$

$$= abc\left(1+\frac{1}{a}+\frac{1}{b}+\frac{1}{c}\right)$$

Example 2.27: *Prove that x = 2 and x = 3 are roots of the equation*

$$\begin{vmatrix} x-5 & 2 \\ -3 & x \end{vmatrix} = 0$$

Solution: Now $\begin{vmatrix} x-5 & 2 \\ -3 & x \end{vmatrix} = 0$

$\Rightarrow \quad x^2 - 5x + 6 = 0$

$\Rightarrow \quad (x-3)(x-2) = 0$

$\Rightarrow \quad x = 3, x = 2$ are roots of the given equation.

Check Your Progress

6. What do you understand by modular constraint?
7. What is a matrix?
8. What is matrix equality?
9. How is adjoint matrix obtained?
10. Define the term determinants.

2.6 VECTORS

We come across different quantities in the study of physical phenomena, such as mass or volume of a body, time, temperature, speed, etc. All these quantities are such that they can be expressed completely by their magnitude, i.e., by a single number. For example, mass of a body can be specified by the number of grams and time by minutes, etc. Such quantities are called scalars. There are certain other quantities which cannot be expressed completely by their magnitude alone, such as velocity, acceleration, force, displacement, momentum, etc. These quantities can be expressed completely by their magnitude and direction and are called vectors.

Representation of Vectors

The best way to represent a vector is with the help of directed line segment. Suppose A and B are two points, then by the vector $\vec{AB}$, we mean a quantity whose magnitude is the length AB and whose direction is from A to B (See Figure 2.13).

A and B are called the end points of the vector $\vec{AB}$. In particular, A is called the initial point and B is called the terminal point. Sometimes a vector $\vec{AB}$ is expressed by a single letter **a**, which is always written in bold type to distinguish it from a scalar. Sometimes, however, we write the vector **a** as $\vec{a}$ or $\overline{a}$.

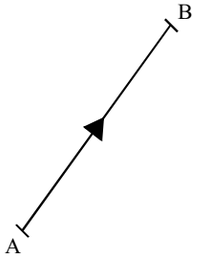


Fig. 2.13 *Vector* $\vec{AB}$

NOTES

Definitions

Modulus of a vector: The modulus or magnitude of a vector is the positive number measuring the length of the line representing it. It is also called the vector's absolute value. Modulus of a vector **a** is denoted by $|\mathbf{a}|$ or by the corresponding letter *a* in italics.

Unit vector: A vector whose magnitude is unity is called a unit vector and is generally denoted by $\hat{\mathbf{a}}$. We will always use the symbols $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$ to denote the unit vectors along the *x*, *y* and *z* axis respectively in three dimensions.

If **a** is any vector, then $\mathbf{a} = a\hat{\mathbf{a}}$, where $\hat{\mathbf{a}}$ is a unit vector having the same direction as **a** (The idea would become clearer when we define the product of a vector with a scalar).

Zero vector: A vector with zero magnitude and any direction is called a zero vector or a null vector. For example, if in Figure 1.1 the point *B* coincides with the point *A*, the vector $\vec{AB}$ becomes the zero vector $\vec{AB}$. The zero vector is denoted by the symbol **0**.

Equality of two vectors: Two vectors are said to be equal if and only if they have the same magnitude and the same direction.

Negative of a vector: The vector which has the same magnitude as the vector **a**, but has the opposite direction is called negative of **a** and is denoted by –**a**.

Thus, $\vec{AB} = -\vec{BA}$ for any vector $\vec{AB}$.

Free vectors: A vector is said to be a free vector or a sliding vector if its magnitude and direction are fixed but position in space is not fixed.

Note: When we defined equality of vectors, it was assumed that the vectors are free vectors. Thus, two vectors $\vec{AB}$ and $\vec{CD}$ can be equal if *AB* = *CD* and *AB* is parallel to *CD*, although they are not coincident (see Figure 2.14).

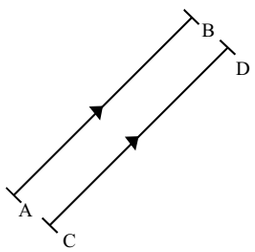


Fig. 2.14 *Equality of Two Vectors*

So, equality of two vectors does not mean that the two vectors are equivalent in all respects. For example, suppose we apply a certain force in a certain direction at two different points of a body, then although the vectors are same still they may have varying effects on the body.

Localized vector is that whose position in space is also fixed.

Coinitial vectors: Vectors having the same initial point are called coinitial vectors or concurrent vectors.

NOTES

Angle Between Two Vectors

Let $\vec{AB} = \mathbf{a}$ and $\vec{CD} = \mathbf{b}$ be any two vectors. Through a point O, take lines OA_1 and OC_1 parallel to $\vec{AB}$ and $\vec{CD}$ respectively (see Figure 2.15).

The angle θ between the lines OA_1 and OC_1 is called the angle between the vectors **a** and **b**, where $0 \le \theta \le \pi$.

If $\theta = 0$ or π, the two vectors are said to be parallel. Thus, parallelism only requires that the two vectors have the same or opposite direction and there is no necessity of relation between their magnitudes.

The vectors are said to be perpendicular if, $\theta = \frac{\pi}{2}$.

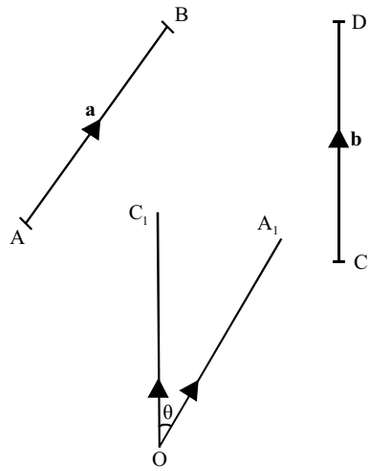


Fig. 2.15 *Angle Between Two Vector*

Vector Spaces

The motivating factor in rings was set of integers and in groups the set of all permutations of a set. A vector space originates from the notion of a vector that we are familiar with in mechanics or geometry. Our aim in this volume is not to go into details of that. Reader would recall that a vector is defined as a directed line segment, which in algebraic terms is defined as an ordered pair (a, b), being coordinates of the terminal point relative to a fixed coordinate system. Addition of vectors is given by the rule,

$$(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$$

One can easily verify that set of vectors under this, forms an abelian group. An abelian group denotes a set A, with an operation '.' and written as $<A, .>$ and should satisfy the condition of closure, associativity, commutativity, identity element and inverse element. Also, scalar multiplication is defined by the rule $a(a, b) = (aa, ab)$ which satisfies certain properties. This concept is extended similarly to three dimensions. We generalise the whole idea through definition of a vector space and vary the scalars not only in the set of real numbers but in any field F. A vector space, thus, differs from groups and rings in as much as it also involves elements from outside itself.

NOTES

Definition. Let $< V, + >$ be an abelian group and $< F, +, . >$ be a field. Define a function (called scalar multiplication) from $F \times V \to V$, for all $\alpha \in F$, $v \in V$, $\alpha \,.\, v \in V$. Then, V is said to form a vector space over F if for all $x, y \in V$, $\alpha, \beta \in F$, the following hold:

(*i*) $(\alpha + \beta)\, x = \alpha x + \beta x$

(*ii*) $\alpha\,(x + y) = \alpha x + \alpha y$

(*iii*) $(\alpha\beta)\, x = \alpha\,(\beta x)$

(*iv*) $1 \cdot x = x$, 1 being unity of F.

Also, members of F are called scalars and those of V are called vectors.

Note: We have used the same symbol '+' for the two different binary compositions of V and F, for convenience. Similarly, same symbol '•' is used for scalar multiplication and product of the field F.

Since $< V, + >$ is a group, its identity element is denoted by 0. Similarly, the field F would also have zero element which will also be represented by 0. In case of doubt one can use different symbols like 0_V and 0_F, etc.

Since we generally work with a fixed field, we shall only be writing V as a vector space (or sometimes $V(F)$ or V_F). It would always be understood that it is a vector space over F (unless stated otherwise).

We defined the scalar multiplication from $F \times V \to V$. One can also define it from $V \times F \to V$ and have a similar definition. The first one is called a left vector space and the second a right vector space. It is easy to show that if V is a left vector space over F then it is a right vector space over F and vice-versa. In the view of this result it becomes redundant to talk about left or right vector spaces. We shall thus talk of only vector spaces over F.

One can also talk about the above system when the scalars are allowed to take values in a ring instead of a field, which leads us to the definition of modules.

Theorem 2.1. In any vector space $V(F)$, the following results hold:

(*i*) $0 \,.\, x = 0$

(*ii*) $\alpha \,.\, 0 = 0$

(*iii*) $(-\alpha)x = -(\alpha x) = \alpha(-x)$

(*iv*) $(\alpha - \beta)x = \alpha x - \beta x,\ \alpha, \beta \in F,\ x \in V$

Proof: (*i*) $0 \,.\, x = (0 + 0) \,.\, x = 0 \,.\, x + 0 \,.\, x$

$\Rightarrow\ 0 \,.\, x = 0$ (Cancellation in V)

(*ii*) $\alpha \,.\, 0 = \alpha \,.\, (0 + 0) = \alpha \,.\, 0 + \alpha \,.\, 0 \Rightarrow \alpha \,.\, 0 = 0$

(*iii*) $(-\alpha)x + \alpha x = [(-\alpha) + \alpha]x = 0 \,.\, x = 0$

$\Rightarrow\ (-\alpha x) = -\alpha x$

(*iv*) Follows from above.

If $< F, +, . >$ be a field, then F is a vector space over F as $< F, + >$ and $< V, + >$ is an additive abelian group. Scalar multiplication can be taken as the product of F. All properties are seen to hold. Thus, $F(F)$ is a vector space.

For example, let $< F, +, . >$ be a field.

NOTES

Let, $V = \{(\alpha_1, \alpha_2) \mid \alpha_1, \alpha_2 \in F\}$

Define + and . (Scalar multiplication) by,

$$(\alpha_1, \alpha_2) + (\beta_1, \beta_2) = (\alpha_1 + \beta_1, \alpha_2 + \beta_2)$$

And, $\alpha(\alpha_1, \alpha_2) = (\alpha\alpha_1, \alpha\alpha_2)$

One can check that all conditions in the definition are satisfied.

Here,

$$V = F \times F = F^2$$

One can extend this to F^3 and so on. In general, we can take n–tuples $(\alpha_1, \alpha_2, ..., \alpha_n)$, $\alpha_i \in F$ and define F^n or $F^{(n)} = \{(\alpha_1, \alpha_2, ..., \alpha_n) \mid \alpha_i \in F\}$ as a vector space over F.

If $F \subseteq K$ be two fields, then $K(F)$ will form a vector space, where addition of $K(F)$ is + of K and F and for any $\alpha \in F$, $x \in K$, $\alpha \,.\, x$ is taken as product of α and x in K.

Thus, $C(R)$, $C(C)$, $R(Q)$ would be some examples of vector spaces, where C = Complex numbers, R = Reals and Q = Rationals.

Let V be a set of all real valued continuous functions defined on [0, 1]. Then V forms a vector space over the field R of real numbers under addition and scalar multiplication defined by,

$$(f + g)x = f(x) + g(x), \qquad f, g \in V$$
$$(\alpha f)x = \alpha f(x), \qquad \alpha \in R$$
$$\text{for all } x \in [0, 1]$$

It may be recalled here that the sum of two continuous functions is continuous and the scalar multiple of a continuous function is continuous.

The set $F[x]$ of all polynomials over a field F in an indeterminate x forms a vector space over F with respect to the usual addition of polynomials and the scalar multiplication defined by:

For, $f(x) = a_0 + a_1x + ... + a_nx^n \in F(x), \quad \alpha \in F$

$$\alpha \,.\, (f(x)) = \alpha a_0 + \alpha a_1x + ... + \alpha a_nx^n$$

$M_{m \times n}(F)$, the set of all $m \times n$ matrices with entries from a field F forms a vector space under addition and scalar multiplication of matrices.

We use the notation $M_n(F)$ for $\mathrm{M}_{m \times n}(F)$.

Let F be a field and X a non-empty set.

Let $F^X = \{f \mid f : X \to F\}$, the set of all mappings from X to F. Then F^X forms a vector space over F under addition and scalar multiplication defined as follows:

For, $f, g \in F^X, \alpha \in F$

Define, $f + g : X \to F$, $\alpha f : X \to F$ such that

$$(f + g)(x) = f(x) + g(x)$$
$$(\alpha f)(x) = \alpha f(x) \ \forall \ x \in X$$

Let V be the set of all vectors in three dimensional space. Addition in V is taken as the usual addition of vectors in geometry and scalar multiplication is defined as

$\alpha \in R$, $v \in V \Rightarrow \alpha v$ is a vector in V with magnitude $|\alpha|$ times that of v. Then V forms a vector space over R.

The Triangle Law

NOTES

If there are two vectors **a** = $\overrightarrow{OA}$ and **b** = $\overrightarrow{AB}$, represented as two sides of a triangle, as shown in Figure 2.16, then the third side $\overrightarrow{OB}$, shown as **c** shows the resultant.

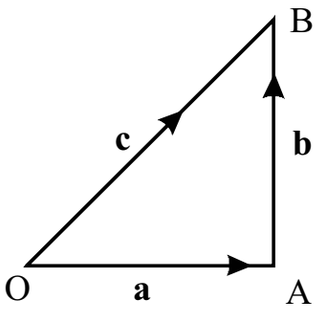


Fig. 2.16 *Triangle Law*

The Parallelogram Law

Let **a** and **b** be any two vectors. Through a point O, take a line OA parallel to the vector **a** and of length equal to a. Then, $\overrightarrow{OA}$ = **a**. Again through A, take a line AB the vector ***b*** having length b, then $\overrightarrow{AB}$ = **b**.

We define the sum of **a** and **b** as (**a** + **b**) to be the vector $\overrightarrow{OB}$ and write,

a + **b** = $\overrightarrow{OB}$.

Similarly, the sum of three or more vectors can be obtained by repeated application of this definition.

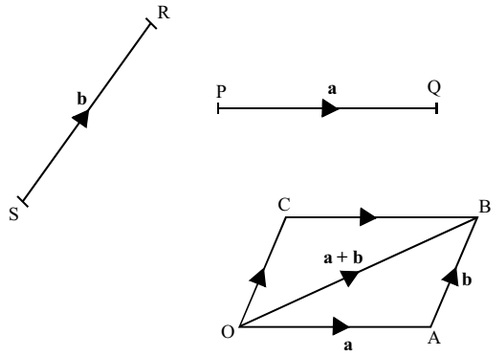


Fig. 2.17 *Parallelogram Law*

Notes: 1. The process by which we obtained an equal vector $\overrightarrow{OA}$ from **a** is sometimes referred to as translation of vectors. It is obtained by moving the line segment of **a** from its original position to the new position OA, without disturbing the direction.
2. This method of addition is called parallelogram law of addition.

NOTES

Derivative of Sum

Vector Addition is Commutative

Let **a** and **b** be any two vectors. We get,

$$\vec{OB} = \mathbf{a} + \mathbf{b}$$ [See Figure 1.5]

Now complete the parallelogram $OABC$, then

$$\vec{OC} = \mathbf{b} \quad \text{and} \quad \vec{CB} = \mathbf{a}$$

Also, $\vec{OC} + \vec{CB} = \vec{OB}$ [By definition of addition]

$\Rightarrow$ $\mathbf{b} + \mathbf{a} = \vec{OB}$

Hence, $\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a} = \vec{OB}$

This proves the result.

Vector Addition is Associative

Let, $\mathbf{a} = \vec{OA}$

$\mathbf{b} = \vec{AB}$

$\mathbf{c} = \vec{BC}$

be three vectors.

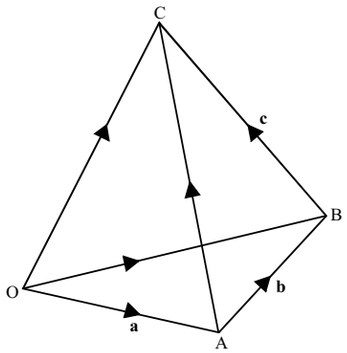


Fig. 2.18 *Vector Addition*

Then, $(\mathbf{a} + \mathbf{b}) + \mathbf{c} = (\vec{OA} + \vec{AB}) + \vec{BC}$

$= \vec{OB} + \vec{BC}$

$= \vec{OC}$ ... (2.1)

And, $\mathbf{a} + (\mathbf{b} + \mathbf{c}) = \vec{OA} + (\vec{AB} + \vec{BC})$

$= \vec{OA} + \vec{AC}$

$= \vec{OC}$ ... (2.2)

Thus, from Equations (2.1) and (2.2), we have,

$$(\mathbf{a} + \mathbf{b}) + \mathbf{c} = \mathbf{a} + (\mathbf{b} + \mathbf{c})$$

Hence proved.

Existence of Identity

NOTES

If **a** is any vector and **0** is the zero vector then,

$$\mathbf{a} + \mathbf{0} = \mathbf{a}$$

In Figure 1.5, if B coincides with A then,

$$\overrightarrow{AB} = \overrightarrow{AA} = \mathbf{0}$$

By definition

$$\overrightarrow{OB} = \overrightarrow{OA} + \overrightarrow{AB}$$

$$\Rightarrow \quad \overrightarrow{OA} = \overrightarrow{OA} + \overrightarrow{AA}$$

$$\Rightarrow \quad \mathbf{a} = \mathbf{a} + \mathbf{0}$$

Hence, proved.

Existence of Inverse

If **a** is any vector, then a vector $-\mathbf{a}$, is called inverse of **a** such that,

$$\mathbf{a} + (-\mathbf{a}) = \mathbf{0}$$

Let, $\mathbf{a} = \overrightarrow{OA}$ [By definition of addition]

Then, by definition, $-\mathbf{a} = \overrightarrow{AO}$

Thus, $\overrightarrow{OA} + \overrightarrow{AO} = \overrightarrow{OO}$

$$\Rightarrow \quad \mathbf{a} + (-\mathbf{a}) = \mathbf{0}$$

In view of the above properties, we can say that the set V of vectors with addition of vectors as a binary composition forms an abelian group.

Subtraction of Vectors: By $\mathbf{a} - \mathbf{b}$ we mean $\mathbf{a} + (-\mathbf{b})$, where $-\mathbf{b}$ is inverse of **b** is also called negative of **b** as defined earlier.

Multiplication of a Vector by a Scalar

Suppose **a** is a vector and n is a scalar. By $n\mathbf{a}$ we mean a vector whose magnitude is $|n||\mathbf{a}|$, i.e., $|n|$ times the magnitude of **a** and whose direction is that of **a** or opposite to that of **a** depending on n being positive or negative.

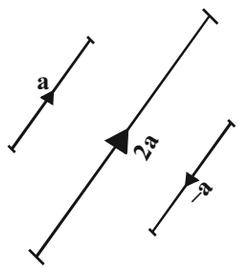


Fig. 2.19 *Scalar Multiplication and Negation of a Vector*

Generally, the scalar is written on the left of the vector, although one could write it on the right too.

Note: We do not put any sign (. or ×) between n and **a** when we write $n\mathbf{a}$.

The following results can be proved:

(*i*) $(mn)\mathbf{a} = m(n\mathbf{a})$

(*ii*) $0\mathbf{a} = \mathbf{0}$

(*iii*) $n(\mathbf{a} + \mathbf{b}) = n\mathbf{a} + n\mathbf{b}$

(*iv*) $(m + n)\mathbf{a} = m\mathbf{a} + n\mathbf{a}$

Proofs: (*i*) and (*ii*) are direct consequence of the definition and hardly need any further proof.

(*iii*) $n(\mathbf{a} + \mathbf{b}) = n\mathbf{a} + n\mathbf{b}$. Let n be positive.

Suppose $\mathbf{a} = \vec{OA}$, $\mathbf{b} = \vec{AB}$

Then, $\vec{OB} = \mathbf{a} + \mathbf{b}$

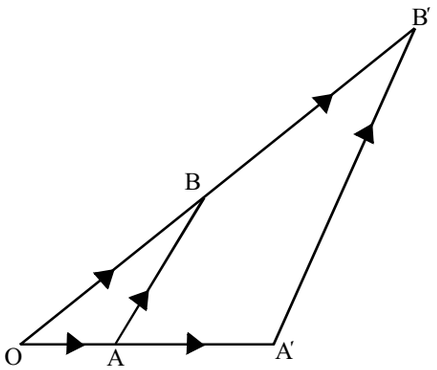


Fig. 2.20 *Graphical Representation of na + nb*

Let A', B' be points on OA and OB (or OA and OB produced) respectively such that,

$$OA' = n \,.\, OA$$

$$OB' = n \,.\, OB$$

Then, $\vec{OA'} = n\vec{OA} = n\mathbf{a}$

$$\vec{OB'} = n\vec{OB} = n(\mathbf{a} + \mathbf{b})$$

Also, $A'B' = nAB$ (Where, OAB and $OA'B'$ are similar triangles)

$\Rightarrow$ $\vec{A'B'} = n\vec{AB}$

$= n\mathbf{b}$

Now, $\vec{OB'} = \vec{OA'} + \vec{A'B'}$

$\Rightarrow$ $n(\mathbf{a} + \mathbf{b}) = n\mathbf{a} + n\mathbf{b}$

This proves our assertion.

When n is negative, the figure would change in this case as now A and A' will lie on the opposite sides of O.

Proceeding as before, we get,

$$\vec{OA'} = n\mathbf{a}$$

NOTES

$$\overrightarrow{A'B'} = n\mathbf{b}$$

$$\overrightarrow{OB'} = n(\mathbf{a} + \mathbf{b})$$

Which proves the result as $\overrightarrow{OB'} = \overrightarrow{OA'} + \overrightarrow{A'B'}$

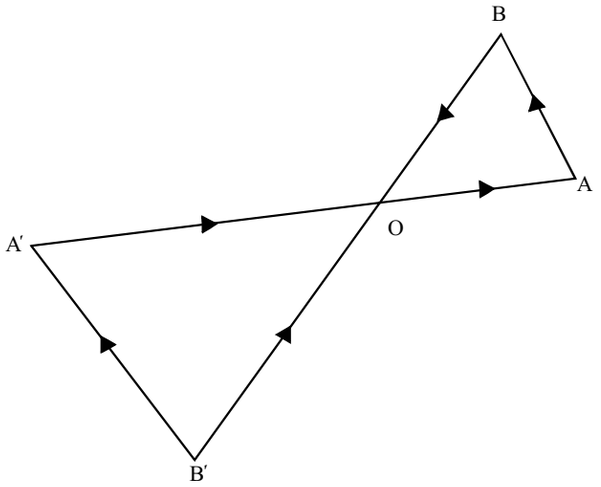


Fig. 2.21 *Graphical Representation of na + nb when n is Negative*

(*iv*) Suppose m and n are positive.

We show that, $(m + n)\mathbf{a} = m\mathbf{a} + n\mathbf{a}$

Let, $m + n = k$

Then,

$$\text{LHS} = \mathbf{a} + \mathbf{a} + \mathbf{a} + \ldots + \mathbf{a} \quad (k \text{ times})$$

Also, $$\text{RHS} = m\mathbf{a} + n\mathbf{a}$$

$$= \underbrace{(\mathbf{a} + \mathbf{a} + \ldots + \mathbf{a})}_{(m \text{ times})} + \underbrace{(\mathbf{a} + \mathbf{a} + \ldots + \mathbf{a})}_{(n \text{ times})}$$

$$= \mathbf{a} + \mathbf{a} + \ldots + \mathbf{a} \quad (m + n \text{ times})$$

$\Rightarrow$ $$\text{LHS} = \text{RHS} \quad (k \text{ times})$$

One can easily prove the result even if m or n is negative.

Aliter: Direction of the vector $(m + n)\mathbf{a}$ is same as that of $\mathbf{a}$, since $m + n > 0$.

Also, directions of the vectors $m\mathbf{a}$ and $n\mathbf{a}$ are same as that of $\mathbf{a}$ and, therefore, direction of $m\mathbf{a} + n\mathbf{a}$ is also same as that of $\mathbf{a}$.

Now, magnitude of the vector $(m + n)\mathbf{a}$ is,

$$|m + n| \ |\boldsymbol{a}| = (m + n)a$$

$$= ma + na$$

$$= |m| \ |\mathbf{a}| + |n| \ |\mathbf{a}|$$

$$= |m \ \mathbf{a}| + |n \ \mathbf{a}| \text{ as they have same direction.}$$

$$= |m\mathbf{a} + n\mathbf{a}|$$

This is the magnitude of the vector $m\mathbf{a} + n\mathbf{a}$, i.e., the vector on the RHS.

Thus, the two vectors have same direction and magnitude and hence they are equal.

The different cases when either m or n is negative or both m and n are negative can be dealt with similarly.

Theorem 2.2: Two non-zero vectors **a** and **b** are parallel if and only if a scalar t is such that, $\mathbf{a} = t\mathbf{b}$.

Proof: Let **a** be parallel to **b**. Then the direction of **a** and **b** is same or opposite.

Suppose direction of **a** and **b** is same.

If $a = b$, where a, b are the magnitudes of **a** and **b** respectively, then $t = 1$ serves our purpose, because then,

$$a = 1.\ b \ \Rightarrow\ \mathbf{a} = 1\mathbf{b}$$

If $a \neq b$, then we can always find a scalar t such that, $a = tb$

(Property of real numbers, indeed we take $t = a/b$)

For this t, we have,

$$\mathbf{a} = t\mathbf{b}$$

So, when direction of **a** and **b** is same, the result is true.

Now, let direction of **a** and **b** be opposite. The same scalar will do the job except that in this case we will take t with the negative sign.

Conversely, let **a** and **b** be two vectors such that, $\mathbf{a} = t\mathbf{b}$ for some scalar t.

By definition of equality of vectors this implies that **a** and t**b** have same direction.

Again, **a** and t**b** have same or opposite direction

[By definition of t**b**]

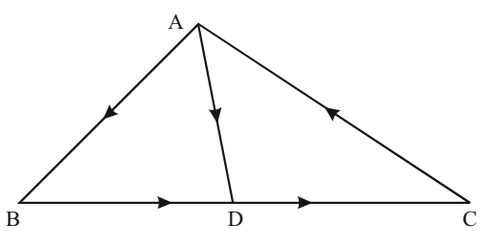
$\Rightarrow$ **a** and **b** are two vectors having same or opposite direction.

$\Rightarrow$ **a** and **b** are parallel.

Hence proved.

Example 2.28: If ABC is a triangle and D is middle point of BC, show that $\overrightarrow{BD} = \frac{1}{2}\overrightarrow{BC}$.

Solution: In figure below, $\overrightarrow{BD}$ is a vector with magnitude $BD = \frac{1}{2}BC$ and $\overrightarrow{BC}$ is a vector with magnitude BC.



Since directions of $\overrightarrow{BC}$ and $\overrightarrow{BD}$ vectors are same, it follows that $\overrightarrow{BD} = \frac{1}{2}\overrightarrow{BC}$

NOTES

Example 2.29: Show that the vector equation, $\mathbf{a} + \mathbf{x} = \mathbf{b}$ has a unique solution.

Solution: We know that,

$$\mathbf{a} + [-\mathbf{a} + \mathbf{b}] = [\mathbf{a} + (-\mathbf{a})] + \mathbf{b} \quad \text{[By Associativity Law]}$$
$$= \mathbf{0} + \mathbf{b} \quad \text{[By Identity Law]}$$
$$= \mathbf{b}$$

$\Rightarrow$ $-\mathbf{a} + \mathbf{b}$ is a solution of $\mathbf{a} + \mathbf{x} = \mathbf{b}$.

Suppose that $\mathbf{y}$ is any other solution of this equation.

Then, $\mathbf{y} = \mathbf{0} + \mathbf{y}$
$= [(-\mathbf{a}) + \mathbf{a}] + \mathbf{y}$
$= (-\mathbf{a}) + (\mathbf{a} + \mathbf{y})$
$= -\mathbf{a} + \mathbf{b}$, as $\mathbf{y}$ is a solution.

$\Rightarrow$ $-\mathbf{a} + \mathbf{b}$ is the unique solution.

Position Vector

Let O be a fixed point, called origin. If P is any point in space and the vector $\overrightarrow{OP} = \mathbf{r}$, we say that position vector of P is $\mathbf{r}$ with respect to the origin O, and express this as $P(\mathbf{r})$.

Whenever we talk about some points with position vectors it is to be understood that all those vectors are expressed with respect to the same origin. To prove that:

If A and B are any two points with position vectors $\mathbf{a}$ and $\mathbf{b}$ then,

$$\overrightarrow{AB} = \mathbf{b} - \mathbf{a}$$

If O is the origin, then it is given that,

$$\overrightarrow{OA} = \mathbf{a}$$
$$\overrightarrow{OB} = \mathbf{b}$$

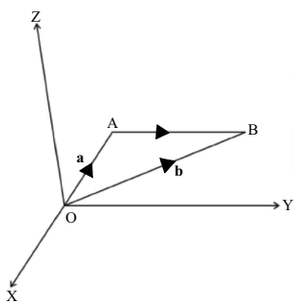


Fig. 2.22 *Position Vector*

Also, $\overrightarrow{OA} + \overrightarrow{AB} = \overrightarrow{OB}$ [By Addition Law]

$\Rightarrow$ $\overrightarrow{AB} = \overrightarrow{OB} - \overrightarrow{OA}$

$\Rightarrow \quad \overrightarrow{AB} = \mathbf{b} - \mathbf{a}$

Components of a Vector

Let P be any point in space with co-ordinates x, y, z. Complete the parallelopiped as shown in Figure 2.23.

NOTES

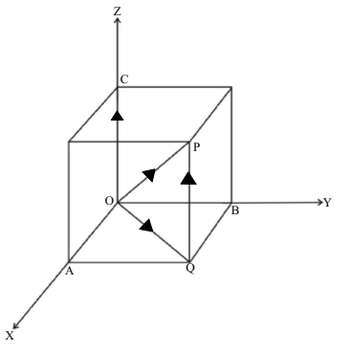


Fig. 1.11 *Components of a Vector*

Then, co-ordinates of the points A, B, C are $(x, 0, 0)$, $(0, y, 0)$, $(0, 0, z)$ respectively.

Suppose that position vector of P is $\mathbf{r}$,

i.e., $\overrightarrow{OP} = \mathbf{r}$

Also, let $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$, be the unit vectors along the three co-ordinates x, y and z-axis.

Now, $OA = x$

$\Rightarrow \quad \overrightarrow{OA} = x\,\hat{\mathbf{i}}$

Note: $x\ \hat{\mathbf{i}}$ is a vector whose magnitude is 1. $x = x$ and whose direction is that of the x-axis and this is precisely the vector $\overrightarrow{OA}$.

Similarly, $\overrightarrow{OB} = y\,\hat{\mathbf{j}}$

$\overrightarrow{OC} = z\hat{\mathbf{k}}$

From the triangle OPQ,

$$\mathbf{r} = \overrightarrow{OP} = \overrightarrow{OQ} + \overrightarrow{QP}$$
$$= \overrightarrow{OQ} + \overrightarrow{OC}$$

Also from the triangle OAQ,

$$\overrightarrow{OQ} = \overrightarrow{OA} + \overrightarrow{AQ}$$
$$= \overrightarrow{OA} + \overrightarrow{OB}$$

Thus, $\mathbf{r} = \overrightarrow{OA} + \overrightarrow{OB} + \overrightarrow{OC} = x\,\hat{\mathbf{i}} + y\,\hat{\mathbf{j}} + z\,\hat{\mathbf{k}}$.

Hence, if P is any point with position vector $\mathbf{r}$ and co-ordinates x, y, z then,

$$\mathbf{r} = x\,\hat{\mathbf{i}} + y\,\hat{\mathbf{j}} + z\,\hat{\mathbf{k}} \qquad \ldots (2.3)$$

Which can be expressed by writing,

$$\mathbf{r} = (x, y, z) \qquad \ldots (2.4)$$

NOTES

Thus, Equations (2.3) and (2.4) mean exactly the same. x, y, z are called the components of the vector **r**.

Note: $OP = |\mathbf{r}| = r$

Using geometry we find,

$$r^2 = OQ^2 + QP^2 = OA^2 + AQ^2 + QP^2$$
$$= OA^2 + OB^2 + OC^2$$
$$\Rightarrow \quad r^2 = x^2 + y^2 + z^2$$

i.e., the square of the modulus of a vector is equal to the sum of the squares of its rectangular components.

Note: The vectors $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, $\hat{\mathbf{k}}$ are said to form an orthonormal triad.

Angle Between Two Vectors: Let A and B be two points in space with position vectors **a** and **b**, and the co-ordinates of A and B be respectively (a_1, a_2, a_3) and (b_1, b_2, b_3).

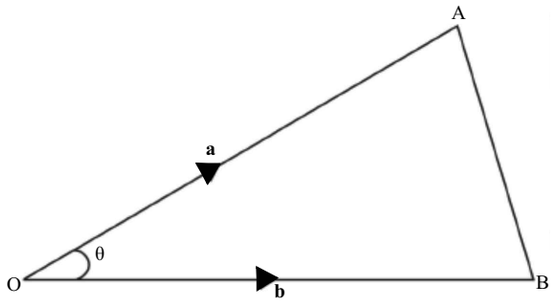


Fig. 2.24 *Angle Between Two Vectors*

Then,

$$\mathbf{a} = \vec{OA} = a_1\,\hat{\mathbf{i}} + a_2\,\hat{\mathbf{j}} + a_3\,\hat{\mathbf{k}}$$

$$\mathbf{b} = \vec{OB} = b_1\,\hat{\mathbf{i}} + b_2\,\hat{\mathbf{j}} + b_3\,\hat{\mathbf{k}}$$

$$\Rightarrow \quad \mathbf{b} - \mathbf{a} = (b_1 - a_1)\,\hat{\mathbf{i}} + (b_2 - a_2)\,\hat{\mathbf{j}} + (b_3 - a_3)\,\hat{\mathbf{k}}$$

Also, $\vec{AB} = \mathbf{b} - \mathbf{a}$

Thus, $AB^2 = (b_1 - a_1)^2 + (b_2 - a_2)^2 + (b_3 - a_3)^2$

Let θ be the angle between **a** and **b**. When θ is the angle between OA and OB, we have,

$$AB^2 = OA^2 + OB^2 - 2\,OA\,.\,OB\cos\theta$$

$$\Rightarrow \quad \cos\theta = \frac{OA^2 + OB^2 - AB^2}{2\,OA\,.\,OB}$$

$$= \frac{(a_1^2 + a_2^2 + a_3^2) + (b_1^2 + b_2^2 + b_3^2) - \Sigma(b_1 - a_1)^2}{2\,a\,.\,b}$$

$$\Rightarrow \qquad \cos\theta = \frac{a_1b_1 + a_2b_2 + a_3b_3}{\sqrt{a_1^2 + a_2^2 + a_3^2}\sqrt{b_1^2 + b_2^2 + b_3^2}}$$

$$\because \qquad a^2 = a_1^2 + a_2^2 + a_3^2$$

And, $$b^2 = b_1^2 + b_2^2 + b_3^2$$

Example 2.30: Find the angle between the vectors

$$\mathbf{a} = \hat{\mathbf{i}} + 2\hat{\mathbf{j}} + 3\hat{\mathbf{k}}$$

$$\mathbf{b} = \hat{\mathbf{i}} - \hat{\mathbf{j}} + 2\hat{\mathbf{k}}$$

Solution: $$\mathbf{a} = (1, 2, 3)$$

$$\mathbf{b} = (1, -1, 2)$$

$$\Rightarrow \qquad a^2 = 1 + 4 + 9 = 14$$

$$b^2 = 1 + 1 + 4 = 6$$

If θ is the angle between **a** and **b**, then

$$\cos\theta = \frac{1\times1 + 2\times(-1) + 3\times2}{\sqrt{14}\sqrt{6}} = \frac{5}{\sqrt{84}}$$

Hence, $$\theta = \cos^{-1}\frac{5}{\sqrt{84}}$$

Section Formula

To find the position vector of a point dividing the join of two given points in a given ratio.

Let A and B be the two given points with position vectors **a** and **b** respectively. Suppose the point $R(\mathbf{r})$ divides AB in the ratio $m : n$.

i.e., $AR : RB = m : n$

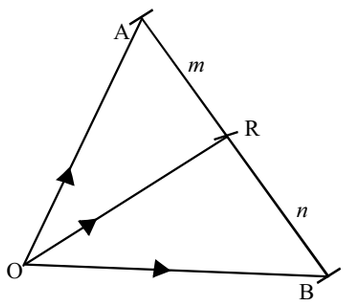


Fig. 2.25 *Section Formula*

Then, $$\frac{AR}{m} = \frac{RB}{n}$$

Or, $$nAR = mRB$$

$$\Rightarrow \qquad n\overrightarrow{AR} = m\overrightarrow{RB}$$

$$\Rightarrow \qquad n(\mathbf{r} - \mathbf{a}) = m(\mathbf{b} - \mathbf{r})$$

NOTES

$\Rightarrow \quad n\mathbf{r} + m\mathbf{r} = m\mathbf{b} + n\mathbf{a}$

$\Rightarrow \quad \mathbf{r} = \dfrac{n\mathbf{a} + m\mathbf{b}}{n+m}$

This gives the required value of **r**.

Corollary. If R is the middle point of AB, then $\mathbf{r} = \dfrac{\mathrm{a} + \mathrm{b}}{2}$ as in this case $m = n$.

Theorem 2.3: Three distinct points A, B, R with position vectors a, b, r are collinear if and only if there exist three numbers x, y, z (not all zero) such that

$$x\mathbf{a} + y\mathbf{b} + z\mathbf{c} = 0$$

and $$x + y + z = 0$$

Proof: Let the three points A, B, R be collinear

Then, R divides AB in some ratio, say $m : n$.

where, $\mathbf{r} = \dfrac{n\mathbf{a} + m\mathbf{b}}{n+m}$

$\Rightarrow \quad (n+m)\mathbf{r} = n\mathbf{a} + m\mathbf{b}$

Or, $n\mathbf{a} + m\mathbf{b} - (n+m)\mathbf{r} = 0$

Let, $x = n, y = m, z = -(n+m)$

Then, $x\mathbf{a} + y\mathbf{b} + z\mathbf{r} = 0$

Where, $x + y + z = n + m - (n+m) = 0$

Thus, all x, y, z can't be zero. Hence proved.

Conversely, Suppose $\exists\, x, y, z$, not all zero such that,

$$x\mathbf{a} + y\mathbf{b} + z\mathbf{r} = 0$$

And, $$x + y + z = 0$$

Let, $z \neq 0$

Then, $x + y + z = 0$

$\Rightarrow \quad x + y = -z$

$\Rightarrow \quad (x+y)\,\mathbf{r} = -z\,\mathbf{r}$

Also, $-z\,\mathbf{r} = x\mathbf{a} + y\mathbf{b}$

Thus, $x\mathbf{a} + y\mathbf{b} = (x+y)\,\mathbf{r}$

$\Rightarrow \quad \mathbf{r} = \dfrac{x\mathbf{a} + y\mathbf{b}}{x+y}$, as $x + y \neq 0$, otherwise $z = 0$

$\Rightarrow$ R divides AB in the ratio $x : y$

$\Rightarrow$ R lies on AB

$\Rightarrow$ A, B, R are collinear.

Hence proved.

Example 2.31: Show that the points with position vectors $3\mathbf{a} - 2\mathbf{b} + 4\mathbf{c}$, $\mathbf{a} + \mathbf{b} + \mathbf{c}$, and $-\mathbf{a} + 4\mathbf{b} - 2\mathbf{c}$ are collinear.

NOTES

Solution: The three points will be collinear if and only if we can find x, y, z (not all zero) such that,

$$x(3\mathbf{a} - 2\mathbf{b} + 4\mathbf{c}) + y(\mathbf{a} + \mathbf{b} + \mathbf{c}) + z(-\mathbf{a} + 4\mathbf{b} - 2\mathbf{c}) = 0 \quad \ldots(1)$$

And,
$$x + y + z = 0 \quad \ldots(2)$$

Equation (1) can be written as,

$$(3x + y - z)\,\mathbf{a} + (-2x + y + 4z)\,\mathbf{b} + (4x + y - 2z)\mathbf{c} = 0$$

This gives,

$$3x + y - z = 0$$
$$-2x + y + 4z = 0$$
$$4x + y - 2z = 0$$

Also, we should have $x + y + z = 0$. One non-zero solution of these four equations is,

$$x = z = 1, \quad y = -2$$

We find that the three given points are collinear.

Symmetry in **a**, **b**, **c** implies that R will also be the point that divides BE and CF in the ratio 2 : 1. This in turn yields that R is the required point where the three medians meet and it also trisects them.

Coplanar Points

It can be proved that four points with position vectors **a**, **b**, **c** and **d** are coplanar if and only if we can find scalars (not all zero) x, y, z and t such that,

$$x\mathbf{a} + y\mathbf{b} + z\mathbf{c} + t\mathbf{d} = 0$$

And,
$$x + y + z + t = 0$$

Example 2.32: Show that the points with position vectors $6\mathbf{a} - 4\mathbf{b} + 4\mathbf{c}$, $-\mathbf{a} - 2\mathbf{b} - 3\mathbf{c}$, $\mathbf{a} + 2\mathbf{b} - 5\mathbf{c}$, $-4\mathbf{c}$ are coplanar.

Solution: The four points will be coplanar if we can find scalars x, y, z and t (not all zero) such that,

$$x(6\mathbf{a} - 4\mathbf{b} + 4\mathbf{c}) + y(-\mathbf{a} - 2\mathbf{b} - 3\mathbf{c}) + z\,(\mathbf{a} + 2\mathbf{b} - 5\mathbf{c}) + t(-4\mathbf{c}) = 0 \quad \ldots(1)$$

And,
$$x + y + z + t = 0 \quad \ldots(2)$$

Rearranging Equation (1), we get,

$$(6x - y + z)\mathbf{a} + (-4x - 2y + 2z)\mathbf{b} + (4x - 3y - 5z - 4t)\,\mathbf{c} = 0$$

This equation suggests that,

$$6x - y + z = 0$$
$$-4x - 2y + 2z = 0$$
$$4x - 3y - 5z - 4t = 0$$

Also, we should have, $x + y + z + t = 0$

Which gives, $x = 0, y = 1, z = 1, t = -2$

This is a non-zero solution of the equations and thus the four points are coplanar.

NOTES

Dot and Cross Product of Vectors

Product of two vectors is defined in two ways, the scalar product and the vector product.

Scalar Product or Dot Product

If **a** and **b** are two vectors, then their scalar product **a . b** (read as **a** dot **b**) is defined by,

$$\mathbf{a} \cdot \mathbf{b} = ab \cos \theta$$

Where, a and b are the magnitudes of the vectors **a** and **b** respectively and θ is the angle between the vectors **a** and **b**.

It is clear from definition that dot product of two vectors is a scalar quantity.

Hence proved.

Scalar Product is Commutative

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= ab \cos \theta \\ &= ba \cos \theta \\ &= \mathbf{b} \cdot \mathbf{a} \end{aligned}$$

Theorem 2.4: Two non-zero vectors **a** and **b** are perpendicular if and only if,

$$\mathbf{a} \cdot \mathbf{b} = 0$$

Proof: Let **a** and **b** be two non-zero perpendicular vectors. Then,

$$\mathbf{a} \cdot \mathbf{b} = ab \cos\left(\frac{\pi}{2}\right) = 0$$

Conversely, Let $\mathbf{a} \cdot \mathbf{b} = 0$

$\Rightarrow$ $ab \cos \theta = 0$

Where, θ is the angle between **a** and **b**.

$\Rightarrow$ $\cos \theta = 0$ [As **a** and **b** are non-zero]

$\Rightarrow$ $\theta = \frac{\pi}{2}$

$\Rightarrow$ **a** and **b** are perpendicular.

The following results are trivial:

$$\hat{\mathbf{i}} \cdot \hat{\mathbf{i}} = \hat{\mathbf{j}} \cdot \hat{\mathbf{j}} = \hat{\mathbf{k}} \cdot \hat{\mathbf{k}} = 1$$

$$\hat{\mathbf{i}} \cdot \hat{\mathbf{j}} = \hat{\mathbf{j}} \cdot \hat{\mathbf{k}} = \hat{\mathbf{k}} \cdot \hat{\mathbf{i}} = 0$$

Definition. By $\mathbf{a}^2$ we will always mean **a . a**.

Thus, $\mathbf{a} \cdot \mathbf{a} = a\,a \cos 0 = \mathbf{a}^2$.

Example 2.33: Show that $\mathbf{a}.(-\mathbf{b}) = -\mathbf{a} \cdot \mathbf{b}$

Solution: We have,

$$\begin{aligned} \mathbf{a}.(-\mathbf{b}) &= ab \cos(\pi - \theta), \text{ where } \theta \text{ is angle between } \mathbf{a} \text{ and } \mathbf{b}. \\ &= -ab \cos \theta \\ &= (-\mathbf{a}) \cdot \mathbf{b}. \end{aligned}$$

Distributive Law

Prove that, $\mathbf{a}.(\mathbf{b}+\mathbf{c}) = \mathbf{a}\,.\,\mathbf{b} + \mathbf{a}\,.\,\mathbf{c}$.

Proof: Let, $\overrightarrow{OA} = \mathbf{a}$

$\overrightarrow{OB} = \mathbf{b}$

$\overrightarrow{BC} = \mathbf{c}$

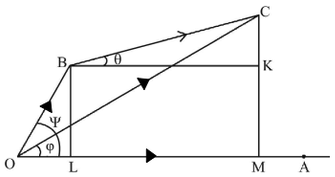


Fig. 2.26s *Distributive Law*

Let BL and CM be perpendiculars from B and C on $\overrightarrow{OA}$ respectively and BK be perpendicular from B on CM.

Then, $\mathbf{a}\,.\,\mathbf{b} = ab\cos\psi = a\,.\,OL$

$\mathbf{a}\,.\,\mathbf{c} = ac\cos\theta = a\,.\,BK = a\,.\,LM$

$\Rightarrow$ RHS $= \mathbf{a}\,.\,\mathbf{b} + \mathbf{a}\,.\,\mathbf{c} = a\,.\,OL + a\,.\,LM = a(OL + LM)$

$= a\,.\,OM$

Again, $\overrightarrow{OB} + \overrightarrow{BC} = \overrightarrow{OC}$

$\Rightarrow$ $\mathbf{b} + \mathbf{c} = \overrightarrow{OC}$

$\Rightarrow$ LHS $= \mathbf{a}\,(\mathbf{b}+\mathbf{c}) = \mathbf{a}\,.\,\overrightarrow{OC}$

$= a\,.\,OC\cos\varphi$

$= a\,.\,OM$

Hence, the result is analysed as follows:

An immediate consequence of the above result is,

$(\mathbf{a}+\mathbf{b})^2 = (\mathbf{a}+\mathbf{b})\,.\,(\mathbf{a}+\mathbf{b})$

$= \mathbf{a}\,.\,\mathbf{a} + \mathbf{a}\,.\,\mathbf{b} + \mathbf{b}\,.\,\mathbf{a} + \mathbf{b}\,.\,\mathbf{b}$

$= \mathbf{a}^2 + \mathbf{a}\,.\,\mathbf{b} + \mathbf{a}\,.\,\mathbf{b} + \mathbf{b}^2$

$= \mathbf{a}^2 + 2\mathbf{ab} + \mathbf{b}^2$

Similarly, we can prove that,

$(\mathbf{a}-\mathbf{b})^2 = \mathbf{a}^2 - 2\mathbf{ab} + \mathbf{b}^2$

$(\mathbf{a}+\mathbf{b})\,.\,(\mathbf{a}-\mathbf{b}) = \mathbf{a}^2 - \mathbf{b}^2$

Scalar Product in Terms of the Components

Let, $\mathbf{a} = (a_1, a_2, a_3) = a_1\,\hat{\mathbf{i}} + a_2\,\hat{\mathbf{j}} + a_3\,\hat{\mathbf{k}}$

$\mathbf{b} = (b_1, b_2, b_3) = b_1\,\hat{\mathbf{i}} + b_2\,\hat{\mathbf{j}} + b_3\,\hat{\mathbf{k}}$

If $\mathbf{a}$ and $\mathbf{b}$ be any two vectors, then

NOTES

$$\mathbf{a} \cdot \mathbf{b} = (a_1 \hat{\mathbf{i}} + a_2 \hat{\mathbf{j}} + a_3 \hat{\mathbf{k}}) \cdot (b_1 \hat{\mathbf{i}} + b_2 \hat{\mathbf{j}} + b_3 \hat{\mathbf{k}})$$

$$= a_1 b_1 \hat{\mathbf{i}} \cdot \hat{\mathbf{i}} + a_2 b_2 \hat{\mathbf{j}} \cdot \hat{\mathbf{j}} + a_3 b_3 \hat{\mathbf{k}} \cdot \hat{\mathbf{k}}$$

[Other terms being zero]

$$= a_1 b_1 + a_2 b_2 + a_3 b_3$$

Angle Between Two Vectors

Since, $\mathbf{a}.\mathbf{b} = ab \cos \theta$

$\Rightarrow$ $$\cos \theta = \frac{\mathbf{a}.\mathbf{b}}{ab}$$

$$= \frac{a_1 b_1 + a_2 b_2 + a_3 b_3}{\sqrt{a_1^2 + a_2^2 + a_3^2}\sqrt{b_1^2 + b_2^2 + b_3^2}}$$

This formula has been proved earlier.

Example 2.34: Show that the vectors **a**, **b**, **c** given by,

$$7\mathbf{a} = 2\hat{\mathbf{i}} + 3\hat{\mathbf{j}} + 6\hat{\mathbf{k}}$$

$$7\mathbf{b} = 3\hat{\mathbf{i}} - 6\hat{\mathbf{j}} + 2\hat{\mathbf{k}}$$

$$7\mathbf{c} = 6\hat{\mathbf{i}} + 2\hat{\mathbf{j}} - 3\hat{\mathbf{k}}$$

are of unit length and are mutually perpendicular.

Solution: The three vectors are,

$$\mathbf{a} = \left(\frac{2}{7}, \frac{3}{7}, \frac{6}{7}\right)$$

$$\mathbf{b} = \left(\frac{3}{7}, \frac{-6}{7}, \frac{2}{7}\right)$$

$$\mathbf{c} = \left(\frac{6}{7}, \frac{2}{7}, \frac{-3}{7}\right)$$

Now, $$a = \sqrt{\left(\frac{2}{7}\right)^2 + \left(\frac{3}{7}\right)^2 + \left(\frac{6}{7}\right)^2} = \frac{1}{7}\sqrt{4+9+36} = 1$$

$$b = \sqrt{\left(\frac{3}{7}\right)^2 + \left(\frac{-6}{7}\right)^2 + \left(\frac{2}{7}\right)^2} = 1$$

$$c = \sqrt{\left(\frac{6}{7}\right)^2 + \left(\frac{2}{7}\right)^2 + \left(\frac{-3}{7}\right)^2} = 1$$

This shows that the given vectors are of unit length.

Again,

$$\mathbf{a} \cdot \mathbf{b} = \frac{2}{7} \times \frac{3}{7} + \frac{3}{7} \times \left(\frac{-6}{7}\right) + \frac{6}{7} \times \frac{2}{7} = \frac{1}{49}(6 - 18 + 12) = 0$$

Thus, **a** and **b** are perpendicular:

Similarly, $$\mathbf{b} \cdot \mathbf{c} = \frac{3}{7}\times\frac{6}{7}+\left(\frac{-6}{7}\right)\times\frac{2}{7}+\frac{2}{7}\times\left(\frac{-3}{7}\right)=\frac{1}{49}(18-12-6)=0$$

$$\mathbf{c} \cdot \mathbf{a} = \frac{6}{7}\times\frac{2}{7}+\frac{2}{7}\times\frac{3}{7}+\left(\frac{-3}{7}\right)\times\frac{6}{7}=\frac{1}{49}(12+6-18)=0$$

This implies that **b** is perpendicular to **c** and **c** is perpendicular to **a**.

Thus, **a, b, c** are mutually perpendicular.

Cross Product of Two Vectors

Vector product is also termed as cross product. If **a** and **b** are two vectors, then their vector product **a** × **b** (read as **a** cross **b**) is defined by,

$$\mathbf{a} \times \mathbf{b} = ab \sin\theta\ \hat{\mathbf{n}}$$

Where, *a* and *b* are the magnitudes of the vectors **a** and **b**, θ is the angle between **a** and **b** and $\hat{\mathbf{n}}$ is a unit vector whose direction is along the normal to the plane of **a** and **b** in the direction from which rotation of **a** to **b** looks anti-clockwise. In other words, direction of $\hat{\mathbf{n}}$ is towards that side to which a right-handed screw will move if **a** is rotated towards **b**.

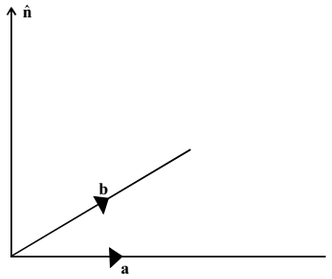


Fig. 2.27 *Cross Products of Two Vectors*

So, if **a** and **b** lie in the plane of the paper as shown in the Figure 2.27 then direction of $\hat{\mathbf{n}}$ is along the normal to the plane of the paper pointing towards the reader.

It is proved from definition that the vector or cross products of two vectors is a vector.

Also according to the definition that **a** × **b** and **b** × **a** will have the same magnitude *ab* sin θ, but opposite directions.

Hence, $\mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a}$

Thus, vector product is not commutative.

Also, for any vector **a**,

$$\mathbf{a} \times \mathbf{a} = a\,a \sin 0.\ \hat{\mathbf{n}} = 0$$

The following results are direct consequence of the definition:

$$\hat{\mathbf{i}} \times \hat{\mathbf{i}} = \hat{\mathbf{j}} \times \hat{\mathbf{j}} = \hat{\mathbf{k}} \times \hat{\mathbf{k}} = 0$$

NOTES

$$\hat{\mathbf{i}} \times \hat{\mathbf{j}} = \hat{\mathbf{k}}, \ \hat{\mathbf{j}} \times \hat{\mathbf{i}} = -\hat{\mathbf{k}}$$

$$\hat{\mathbf{j}} \times \hat{\mathbf{k}} = \hat{\mathbf{i}}, \ \hat{\mathbf{k}} \times \hat{\mathbf{j}} = -\hat{\mathbf{i}}$$

$$\hat{\mathbf{k}} \times \hat{\mathbf{i}} = \hat{\mathbf{j}}, \ \hat{\mathbf{i}} \times \hat{\mathbf{k}} = -\hat{\mathbf{j}}$$

Example 2.35: Show that $(m\mathbf{a}) \times \mathbf{b} = \mathbf{a} \times (m\mathbf{b}) = m(\mathbf{a} \times \mathbf{b})$

Solution: We have,

$$(m\mathbf{a}) \times \mathbf{b} = mab \sin\theta \ \hat{\mathbf{n}}$$

$$= amb \sin\theta \ \hat{\mathbf{n}}$$

$$= \mathbf{a} \times (m\mathbf{b})$$

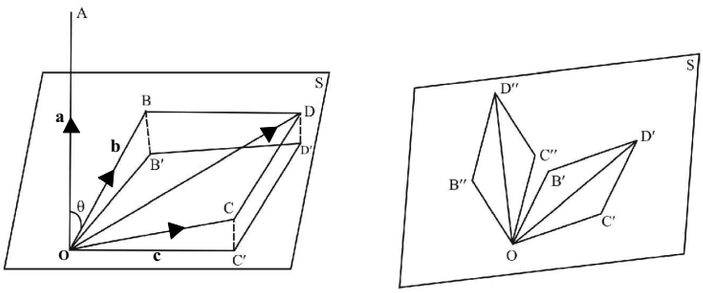
Similarly, $(m\mathbf{a}) \times \mathbf{b} = m(\mathbf{a} \times \mathbf{b})$

Distributive Law

Prove that, $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$

Proof: Through a point O, take vectors $\overrightarrow{OA} = \mathbf{a}$, $\overrightarrow{OB} = \mathbf{b}$ and $\overrightarrow{OC} = \mathbf{c}$. Let, S be a plane through O and perpendicular at $\overrightarrow{OA}$. Complete the parallelogram $OCDB$.

Such that, $\overrightarrow{OD} = \mathbf{b} + \mathbf{c}$



(*i*) (*ii*)

Fig. 2.28 *Distributive Law*

Let B', C', D' be the feet of perpendiculars from B, C, D respectively on the plane S.

Let, $\angle AOB = \theta$

Then, $\mathbf{a} \times \mathbf{b} = ab \sin\theta \ \hat{\mathbf{n}}$

Where, $\hat{\mathbf{n}}$ is a unit vector perpendicular to both OA and OB in the direction determined by motion of right-handed screw.

Now, $\angle BOB' = \frac{\pi}{2} - \theta$

Thus, $\mathbf{b}' = \overrightarrow{OB'}$

Then, $|\mathbf{b}'| = b' = OB' = OB \cos\left(\frac{\pi}{2} - \theta\right) = b \sin\theta$

NOTES

Again since OA, OB, OB' are in the same plane $AOB'B$, unit vector perpendicular to **a** and **b** is same as unit vector perpendicular to **a** and **b′**.

Thus, $$\mathbf{a} \times \mathbf{b}' = ab' \sin\left(\frac{\pi}{2}\right) \hat{\mathbf{n}}$$

$$= ab \sin\theta \ \hat{\mathbf{n}} = \mathbf{a} \times \mathbf{b}$$

Similarly, $$\mathbf{a} \times \mathbf{c} = \mathbf{a} \times \mathbf{c}'$$

Where, $$\mathbf{c}' = \vec{OC}'$$

And, $$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \vec{OD} = \mathbf{a} \times \vec{OD}'$$

Since OA is perpendicular to plane S and OD' lies in it, the unit vector perpendicular to both $\vec{OA}$ and $\vec{OD}'$ will lie in the plane S, say along OD''. So, OD'' lies in the plane S and is perpendicular to both OA and OD'.

Draw lines, $OB'' \perp OB'$ and $OC'' \perp OC'$,

Such that, $$OB'' = \vec{OA} \ . \ \vec{OB}'$$

$$OC'' = \vec{OA} \ . \ \vec{OC}'$$

Also cut off, $$OD'' = \vec{OA} \ . \ \vec{OD}'$$

Then,

$$\mathbf{a} \times \mathbf{b}' = \vec{OA} \times \vec{OB}' = \vec{OB}''$$

$$\mathbf{a} \times \mathbf{c}' = \vec{OA} \times \vec{OC}' = \vec{OC}''$$

$$\mathbf{a} \times \vec{OD}' = \vec{OA} \times \vec{OD}' = \vec{OD}''$$

Also, $OC''\ D''\ B''$ will be a parallelogram as $OC'D'B'$ is a parallelogram. Hence,

$$\vec{OD}'' = \vec{OB}'' + \vec{OC}''$$

i.e., $$\mathbf{a} \times \vec{OD}' = \mathbf{a} \times \mathbf{b}' + \mathbf{a} \times \mathbf{c}'$$

i.e., $$\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$$

(As proved earlier)

The above proof is sometimes referred to as the geometrical proof of the distributive law.

Vector Product in Terms of Components

Let, $$\mathbf{a} = (a_1, a_2, a_3) = a_1 \hat{\mathbf{i}} + a_2 \hat{\mathbf{j}} + a_3 \hat{\mathbf{k}}$$

$$\mathbf{b} = (b_1, b_2, b_3) = b_1 \hat{\mathbf{i}} + b_2 \hat{\mathbf{j}} + b_3 \hat{\mathbf{k}}$$

Where, **a** and **b** be any two vectors. Then,

$$\mathbf{a} \times \mathbf{b} = (a_1 \hat{\mathbf{i}} + a_2 \hat{\mathbf{j}} + a_3 \hat{\mathbf{k}}) \times (b_1 \hat{\mathbf{i}} + b_2 \hat{\mathbf{j}} + b_3 \hat{\mathbf{k}})$$

$$= a_1 b_1 \hat{\mathbf{i}} \times \hat{\mathbf{i}} + a_1 b_2 \hat{\mathbf{i}} \times \hat{\mathbf{j}} + a_1 b_3 \hat{\mathbf{i}} \times \hat{\mathbf{k}} + a_2 b_1 \hat{\mathbf{j}} \times \hat{\mathbf{i}} + a_2 b_2 \hat{\mathbf{j}} \times \hat{\mathbf{j}}$$

$$+ a_2 b_3 \hat{\mathbf{j}} \times \hat{\mathbf{k}} + a_3 b_1 \hat{\mathbf{k}} \times \hat{\mathbf{i}} + a_3 b_2 \hat{\mathbf{k}} \times \hat{\mathbf{j}} + a_3 b_3 \hat{\mathbf{k}} \times \hat{\mathbf{k}}$$

NOTES

$$= a_1b_2\hat{\mathbf{k}} - a_1b_3\hat{\mathbf{j}} - a_2b_1\hat{\mathbf{k}} + a_2b_3\hat{\mathbf{i}} + a_3b_1\hat{\mathbf{j}} - a_3b_2\hat{\mathbf{i}}$$

$$= \hat{\mathbf{i}}(a_2b_3 - a_3b_2) + \hat{\mathbf{j}}(a_3b_1 - a_1b_3) + \hat{\mathbf{k}}(a_1b_2 - a_2b_1)$$

$$= \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

Example 2.36: If $\mathbf{a} = 2\hat{\mathbf{i}} - \hat{\mathbf{j}} + \hat{\mathbf{k}}$, $\mathbf{b} = 3\hat{\mathbf{i}} + 4\hat{\mathbf{j}} - \hat{\mathbf{k}}$, verify that $\mathbf{a} \times \mathbf{b}$ represents a vector perpendicular to both **a** and **b**.

Solution: We have,

$$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 2 & -1 & 1 \\ 3 & 4 & -1 \end{vmatrix} = -3\hat{\mathbf{i}} + 5\hat{\mathbf{j}} + 11\hat{\mathbf{k}}$$

Now, $\mathbf{a}.(\mathbf{a} \times \mathbf{b}) = (2\hat{\mathbf{i}} - \hat{\mathbf{j}} + \hat{\mathbf{k}}) . (-3\hat{\mathbf{i}} + 5\hat{\mathbf{j}} + 11\hat{\mathbf{k}})$

$= 2 . (-3) + (-1) . 5 + 1 . 11 = 0$

$\Rightarrow$ **a** is perpendicular to $\mathbf{a} \times \mathbf{b}$.

Similarly, we can prove that **b** is also perpendicular to $\mathbf{a} \times \mathbf{b}$.

Theorem 2.5: Two vectors $\mathbf{a} = (a_1, a_2, a_3)$ and $\mathbf{b} = (b_1, b_2, b_3)$ are parallel if and only if,

$$\frac{a_1}{b_1} = \frac{a_2}{b_2} = \frac{a_3}{b_3}$$

Proof: Let **a** and **b** be parallel.

$\Rightarrow$ Angle θ between them is zero.

$\Rightarrow \quad \sin\theta = 0$

$\Rightarrow \quad \mathbf{a} \times \mathbf{b} = 0$

$$\Rightarrow \quad \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = 0$$

$$\Rightarrow (a_2b_3 - a_3b_2)\hat{\mathbf{i}} + (a_3b_1 - a_1b_3)\hat{\mathbf{j}} + (a_1b_2 - a_2b_1)\hat{\mathbf{k}} = 0 = 0\hat{\mathbf{i}} + 0\hat{\mathbf{j}} + 0\hat{\mathbf{k}}$$

$$\Rightarrow \quad a_2b_3 - a_3b_2 = 0$$
$$a_3b_1 - a_1b_3 = 0$$
$$a_1b_2 - a_2b_1 = 0$$

$$\Rightarrow \quad \frac{a_1}{b_1} = \frac{a_2}{b_2} = \frac{a_3}{b_3}$$

Converse follows by simply retracing the steps back.

Vector Product as Area

NOTES

Let $OABC$ be a parallelogram such that,

$$\overrightarrow{OA} = \mathbf{a}$$

$$\overrightarrow{OC} = \mathbf{b}$$

and let θ be the angle between **a** and **b**.

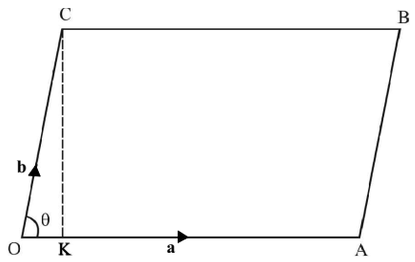


Fig. 2.29 *Vector Product as Area*

Now area of the parallelogram $OABC$,

$$= OA \,.\, CK \qquad (\text{Where, } CK \perp OA)$$

$$= ab \sin\theta$$

Also, $\mathbf{a} \times \mathbf{b} = (ab \sin\theta)\ \hat{\mathbf{n}}$

Comparing the two we note that, $\mathbf{a} \times \mathbf{b}$ = Area of the parallelogram $OABC$, i.e., the magnitude of the vector product of two vectors is the area of the parallelogram whose adjacent sides are represented by these vectors.

Corollary: It is easy to see that the area of the triangle OAC is $\frac{1}{2}\ \mathbf{a} \times \mathbf{b}$.

Example 2.37: Find the area of the parallelogram whose adjacent sides are determined by the vectors $\mathbf{a} = \hat{\mathbf{i}} + 2\hat{\mathbf{j}} + 3\hat{\mathbf{k}}$, $\mathbf{b} = 3\hat{\mathbf{i}} - 2\hat{\mathbf{j}} + \hat{\mathbf{k}}$.

Solution: Required area = $|\mathbf{a} \times \mathbf{b}|$

Now, $$\mathbf{a} \times \mathbf{b} = \begin{vmatrix} \hat{\mathbf{i}} & \hat{\mathbf{j}} & \hat{\mathbf{k}} \\ 1 & 2 & 3 \\ 3 & -2 & 1 \end{vmatrix}$$

$$= \hat{\mathbf{i}}\,(2+6) - \hat{\mathbf{j}}\,(1-9) + \hat{\mathbf{k}}\,(-2-6)$$

$$= 8(\hat{\mathbf{i}} + \hat{\mathbf{j}} - \hat{\mathbf{k}})$$

$\Rightarrow$ $|\mathbf{a} \times \mathbf{b}| = \sqrt{64+64+64} = 8\sqrt{3}$ is the required area.

Triple Product (Scalar, Vector)

Scalar Triple Product

The scalar triple product is defined as the dot product of one of the vectors with the cross product of the other two. Suppose **a**, **b**, **c** are three vectors. Then, $\mathbf{b} \times \mathbf{c}$ is again a vector and thus we can talk of $\mathbf{a}$. $(\mathbf{b} \times \mathbf{c})$, which would, of course, be a scalar. This is called scalar triple product of three vectors.

NOTES

Let, $\mathbf{a} = (a_1, a_2, a_3)$

$\mathbf{b} = (b_1, b_2, b_3)$

$\mathbf{c} = (c_1, c_2, c_3)$

Then, $\mathbf{b} \times \mathbf{c} = (b_2c_3 - b_3c_2,\ b_3c_1 - c_3b_1,\ b_1c_2 - b_2c_1)$

$= (d_1, d_2, d_3) = \mathbf{d}$ (say)

Thus, $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{a} \cdot \mathbf{d} = a_1d_1 + a_2d_2 + a_3d_3$

$= a_1(b_2c_3 - b_3c_2) + a_2(b_3c_1 - c_3b_1) + a_3(b_1c_2 - b_2c_1)$

$$= \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{vmatrix} \quad \ldots(2.5)$$

Hence, this is the value of the scalar triple product **a.** (**b** × **c**).

Suppose we had started with **b .** (**c** × **a**), it is easy to prove that the resulting determinant would have been,

$$\begin{vmatrix} b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \\ a_1 & a_2 & a_3 \end{vmatrix}$$

which is same as Equation (1.5) (As per the properties of determinants), and so,

$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{b} \cdot (\mathbf{c} \times \mathbf{a})$

Similarly, $\mathbf{b} \cdot (\mathbf{c} \times \mathbf{a}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b})$

$\Rightarrow$ $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b}) = (\mathbf{a} \times \mathbf{b}) \cdot \mathbf{c}$ [By Commutative Property]

Dot and cross can be interchanged in a scalar triple product and we write the scalar triple product as [**a, b, c**] or [**abc**], where it is upto the reader where to put cross and dot.

Note: It can be verified that,

$$[\mathbf{abc}] = -[\mathbf{bac}]$$

And, $$[\mathbf{abc}] = [\mathbf{bca}] = [\mathbf{cab}]$$

Example 2.38: Prove the distributive law $\mathbf{a} \times (\mathbf{b} + \mathbf{c}) = \mathbf{a} \times \mathbf{b} + \mathbf{a} \times \mathbf{c}$ using scalar triple product.

Solution: Let, $\mathbf{d} = \mathbf{a} \times (\mathbf{b} + \mathbf{c}) - \mathbf{a} \times \mathbf{b} - \mathbf{a} \times \mathbf{c}$

Then, it required to prove that, $\mathbf{d} = \mathbf{0}$

Let **e** be any vector, then,

$\mathbf{e} \cdot \mathbf{d} = \mathbf{e} \cdot [\mathbf{a} \times (\mathbf{b} + \mathbf{c})] - \mathbf{e} \cdot (\mathbf{a} \times \mathbf{b}) - \mathbf{e} \cdot (\mathbf{a} \times \mathbf{c})$

$= (\mathbf{e} \times \mathbf{a}) \cdot (\mathbf{b} + \mathbf{c}) - (\mathbf{e} \times \mathbf{a}) \cdot \mathbf{b} - (\mathbf{e} \times \mathbf{a}) \cdot \mathbf{c}$

[Interchanging dot and cross in the scalar triple products]

$= (\mathbf{e} \times \mathbf{a}) \cdot [(\mathbf{b} + \mathbf{c}) - \mathbf{b} - \mathbf{c}]$ [Distributivity of scalar product]

$= (\mathbf{e} \times \mathbf{a}) \cdot 0 = 0$

$\Rightarrow$ $\mathbf{e} \cdot \mathbf{d} = 0$ for all vectors $\mathbf{e}$

$\Rightarrow$ $\mathbf{d} = \mathbf{0}$ (We can take $\mathbf{e}$ to be a non-zero vector, not perpendicular to $\mathbf{d}$.)

Hence proved.

Vector Triple Product

A vector triple product is defined as the cross product of one vector with the cross product of the other two.

A product of the type $\mathbf{a} \times (\mathbf{b} \times \mathbf{c})$ is called a vector triple product.

We prove,

$$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\,\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\,\mathbf{c}$$

in the following way:

Let, $\mathbf{a} = (a_1, a_2, a_3)$

$\mathbf{b} = (b_1, b_2, b_3)$

$\mathbf{c} = (c_1, c_2, c_3)$

Then, $\mathbf{b} \times \mathbf{c} = (b_2c_3 - b_3c_2,\ b_3c_1 - b_1c_3,\ b_1c_2 - b_2c_1)$

$= (d_1, d_2, d_3) = \mathbf{d}$ (say)

Then,

$$\begin{aligned}
\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) &= \mathbf{a} \times \mathbf{d} \\
&= (a_2d_3 - a_3d_2,\ a_3d_1 - a_1d_3,\ a_1d_2 - a_2d_1) \\
&= (a_2d_3 - a_3d_2)\ \hat{\mathbf{i}} + (a_3d_1 - a_1d_3)\ \hat{\mathbf{j}} + (a_1d_2 - a_2d_1)\ \hat{\mathbf{k}} \\
&= \Sigma(a_2d_3 - a_3d_2)\ \hat{\mathbf{i}} \\
&= \Sigma[a_2(b_1c_2 - b_2c_1) - a_3(b_3c_1 - b_1c_3)]\ \hat{\mathbf{i}} \\
&= \Sigma[a_2b_1c_2 - a_2b_2c_1 - a_3b_3c_1 + a_3b_1c_3 + a_1b_1c_1 - a_1b_1c_1]\,\hat{\mathbf{i}} \\
&\qquad \text{[Adding and subtracting } a_1b_1c_1\text{]} \\
&= [b_1(a_1c_1 + a_2c_2 + a_3c_3) - c_1(a_1b_1 + a_2b_2 + a_3b_3)]\ \hat{\mathbf{i}} \\
&\quad + [b_2(a_1c_1 + a_2c_2 + a_3c_3) - c_2(a_1b_1 + a_2b_2 + a_3b_3)]\ \hat{\mathbf{j}} \\
&\quad + [b_3(a_1c_1 + a_2c_2 + a_3c_3) - c_3(a_1b_1 + a_2b_2 + a_3b_3)]\ \hat{\mathbf{k}} \\
&= (a_1c_1 + a_2c_2 + a_3c_3)\ (b_1\hat{\mathbf{i}} + b_2\hat{\mathbf{j}} + b_3\hat{\mathbf{k}}) \\
&\quad - (a_1b_1 + a_2b_2 + a_3b_3)\ (c_1\hat{\mathbf{i}} + c_2\hat{\mathbf{j}} + c_3\hat{\mathbf{k}}) \\
&= (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}
\end{aligned}$$

This proves our assertion.

Note: There is neither a cross nor a dot between $(\mathbf{a} \cdot \mathbf{c})$ and $\mathbf{b}$ in $(\mathbf{a} \cdot \mathbf{c})\mathbf{b}$ and between $(\mathbf{a} \cdot \mathbf{b})$ and $\mathbf{c}$ in $(\mathbf{a} \cdot \mathbf{b})\,\mathbf{c}$.

This is so, because $(\mathbf{a} \cdot \mathbf{c})$ and $(\mathbf{a} \cdot \mathbf{b})$ are scalars.

NOTES

Example 2.39: Prove that, $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) + \mathbf{b} \times (\mathbf{c} \times \mathbf{a}) + \mathbf{c} \times (\mathbf{a} \times \mathbf{b}) = \mathbf{0}$.

Solution: We know that,

$$\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{a} \cdot \mathbf{b})\mathbf{c}$$

$$\mathbf{b} \times (\mathbf{c} \times \mathbf{a}) = (\mathbf{b} \cdot \mathbf{a})\mathbf{c} - (\mathbf{b} \cdot \mathbf{c})\mathbf{a}$$

$$\mathbf{c} \times (\mathbf{a} \times \mathbf{b}) = (\mathbf{c} \cdot \mathbf{b})\mathbf{a} - (\mathbf{c} \cdot \mathbf{a})\mathbf{b}$$

The result is computed using addition.

2.6.1 Coordinate System

It is a method of representing points in a space of given dimensions by coordinates. Arrangement of reference lines or curves is used to identify the location of points in space. In two dimensions, the most common system is the Cartesian system, named after René Descartes. Points are designated by their distance along a horizontal (x) and vertical (y) axis from a reference point, the origin designated (0, 0). Cartesian coordinates can also be used for three or more dimensions. A polar coordinate system locates a point by its direction relative to a reference direction and its distance from a given point also called the origin. Such a system is used in radar or sonar tracking and is the basis of bearing-and-range navigation systems. In three dimensions, it leads to cylindrical and spherical coordinates.

Schemes for locating points in a given space by means of numerical quantities specified with respect to some frame of references are the coordinates of a point. To each set of coordinates there corresponds just one point in any coordinate system, but there are useful coordinate systems in which to a given point there may correspond more than one set of coordinates.

A coordinate system is a mathematical language that is used to describe geometrical objects analytically; that is, if the coordinates of a set of points are known, their relationships and the properties of figures determined by them can be obtained by numerical calculations instead of by other descriptions. It is the province of analytic geometry, aided chiefly by calculus, to investigate the means for these calculations.

The most familiar spaces are the plane and the three-dimensional Euclidean space. In the latter a point P is determined by three coordinates (x, y, z). The totality of points for which x has a fixed value constitutes a surface. The same is true for y and z so that through P, there are three coordinate surfaces. The totality of points for which x and y are fixed is a curve and through each point, there are three coordinate lines. If these lines are all straight, the system of coordinates is said to be rectilinear. If some or all of the coordinate lines are not straight, the system is curvilinear. If the angles between the coordinate lines at each point are right angles, then the system is rectangular.

A Cartesian coordinate system is one of the simplest and most useful systems of coordinates. It is constructed by choosing a point O designated as the origin. Through it three intersecting directed lines OX, OY, OZ, the coordinate axes, are constructed. The coordinates of a point P are x, the distance of P from the plane YOZ measured parallel to OX, and y and z, which are determined similarly (refer Figure 2.30). Usually the three axes are taken to be mutually perpendicular, in

NOTES

which case the system is a rectangular Cartesian one. Obviously a similar construction can be made in the plane, in which case a point has two coordinates (*x,y*).

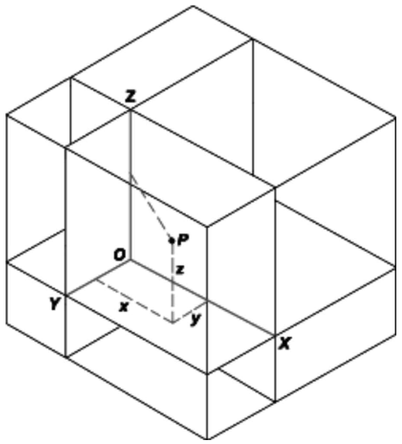


Fig. 2.30 *Cartesian Coordinate System*

A polar coordinate system is constructed in the plane by choosing a point *O* called the pole and through it a directed straight line, the initial line. A point *P* is located by specifying the directed distance *OP* and the angle through which the initial line must be turned to coincide with *OP* in position and direction. The coordinates of *P* are (r, θ). The radius vector *r* is the directed line *OP*, and the vectorial angle θ is the angle through which the initial line was turned (+ if turned counterclockwise, – if clockwise).

Spherical coordinates are constructed in three-dimensional Euclidean space by choosing a plane and in it constructing a polar coordinate system. At the pole *O* a polar axis *OZ* is constructed at right angles to the chosen plane. A point *P*, not on *OZ*, and *OZ* determine a plane. The spherical coordinates of *P* are then the directed distance *OP*, the angle θ through which the initial line is turned to lie in *ZOP* and the angle $\varphi = ZOP$ (refer Figure 2.31).

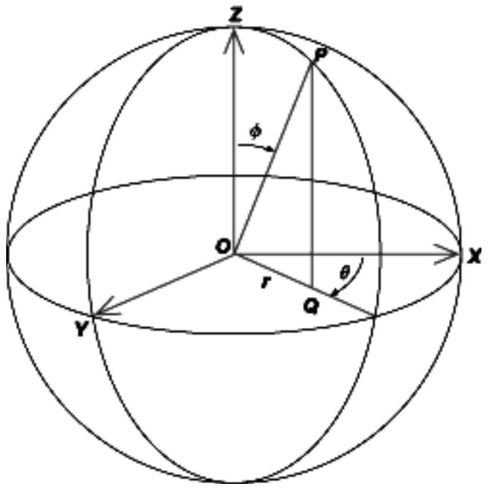


Fig. 2.31 *Spherical Coordinate System*

Cylindrical coordinates are constructed by choosing a plane with a pole *O*,

an initial line in it, and a polar axis *OZ*, as in spherical coordinates. A point *P* is projected onto the chosen plane. The cylindrical coordinates of *P* are (r,θ,z) where *r* and θ are the polar coordinates of *Q* and $z = QP$ (refer Figure 2.2). By means of a system of equations the description of a geometrical object in one coordinate system may be translated into an equivalent description in another coordinate system.

Check Your Progress

11. What is the best way to represent a vector?
12. What is the localized vector?
13. What are coinitial vectors?
14. What do you understand by scalar triple product?
15. What is a coordinate system?

2.7 ANSWERS TO 'CHECK YOUR PROGRESS'

1. A trackball is a pointing device consisting of a ball housed in a socket containing sensors to detect the rotation of the ball about two axes like an upside-down mouse with an exposed protruding ball.
2. A data glove is an interface device that uses position tracking sensors and fiber optic strands running down each finger and connected to a compatible computer.
3. When you place an object on the copyboard or glass surface (like a copier machine) and start scanning, the light source illuminates a thin horizontal strip of the object called a raster line.
4. The various types of locator devices are: absolute devices, relative devices, direct devices, indirect devices, continuous devices and discrete devices.
5. In the rubber-band technique of picture construction, the user generally has to drag the mouse pointer, i.e., to move the mouse cursor while the button is down. The entity shape and/or size is dynamically changed with every mouse movement while the button is down. When the button is released, the end point is frozen and the figure is there in its final shape, size and position.
6. The most common form of constraint is the modular constraint, which forces the input point to the nearest intersection on a grid.
7. A matrix is simply an arrangement of elements and has no numerical value.
8. Two matrices *A* and *B* are said to be equal if,

   (*i*) *A* and *B* are of same order.

   (*ii*) Corresponding elements in *A* and *B* are same. For example, the following two matrices are equal.

$$\begin{pmatrix} 3 & 4 & 9 \\ 16 & 25 & 64 \end{pmatrix} = \begin{pmatrix} 3 & 4 & 9 \\ 16 & 25 & 64 \end{pmatrix}$$

9. The adjoint matrix of A is obtained by replacing the elements of A by their respective cofactors and then transposing.
10. If A is a square matrix with entries from the field of complex numbers, then determinant of A is some complex number. This will be denoted by det A or $| A |$. If

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

then det A will be denoted by

$$\begin{vmatrix} a_{11} & a_{12} & a_{1n} \\ a_{n1} & a_{n2} & a_{nn} \end{vmatrix}$$

11. The best way to represent a vector is with the help of directed line segment.
12. Localized vector is that whose position in space is also fixed.
13. Vectors having the same initial point are called coinitial vectors or concurrent vectors.
14. The scalar triple product is defined as the dot product of one of the vectors with the cross product of the other two. Suppose a, b, c are three vectors. Then b × c is again a vector and thus we can talk of a. (b × c), which would, of course, be a scalar. This is called scalar triple product of three vectors.
15. A coordinate system is a mathematical language that is used to describe geometrical objects analytically; that is, if the coordinates of a set of points are known, their relationships and the properties of figures determined by them can be obtained by numerical calculations instead of by other descriptions.

2.8 SUMMARY

- Various devices are available for data input to general-purpose computer systems with graphic capabilities or sophisticated workstations designed for graphics applications.
- Using a keyboard, a person can type a document, use keystroke shortcuts, access menus, play games and perform a variety of other tasks.
- A mouse is basically a handheld pointing device, designed to sit under one hand of the user and to detect movement relative to its two-dimensional supporting surface.
- A trackball is a pointing device consisting of a ball housed in a socket containing sensors to detect the rotation of the ball about two axes like an upside-down mouse with an exposed protruding ball.
- Spaceball is a graphical input device that is based on a fixed spherical ball. It inputs six different values defined by the orientation of the ball and the pressure together with the direction that is applied to it.
- The image scanner is a graphic device which directly copies images from a paper or photograph and converts it into the digital format for display, storage and graphic manipulations.

- The logical classification of input devices decides the way to receive the graphical data to be processed by a graphic package.
- The various types of locator devices are: absolute devices, relative devices, direct devices, indirect devices, continuous devices and discrete devices.
- The most common form of constraint is the modular constraint, which forces the input point to the nearest intersection on a grid.
- A matrix which has exactly one row is called a row matrix.
- A matrix which has exactly one column is called a column matrix.
- A matrix in which the number of rows is equal to the number of columns is called a square matrix.
- The adjoint matrix of A is obtained by replacing the elements of A by their respective cofactors and then transposing.
- If A is a square matrix with entries from the field of complex numbers, then determinant of A is some complex number. This will be denoted by det A or $|A|$.
- A vector is a definite mathematical structure.
- There are numerous physical and geometric applications of vectors.
- The ability to represent magnitude and direction simultaneously facilitates its numerous applications.
- Vector, in mathematics, refers to a quantity, such as velocity, completely specified by a magnitude and a direction.
- A vector of n dimensions is an ordered collection of n elements, which are called components.
- Parallelism requires that the two vectors have either the same or the opposite direction and that there is no necessary relationship between their magnitudes.
- The vector or cross products of two vectors is always a vector.
- The scalar triple product is defined as the dot product of one of the vectors with the cross product of the other two.
- A vector triple product is defined as the cross product of one vector with the cross product of the other two.
- A coordinate system is a mathematical language that is used to describe geometrical objects analytically; that is, if the coordinates of a set of points are known, their relationships and the properties of figures determined by them can be obtained by numerical calculations instead of by other descriptions.

2.9 KEY TERMS

- Trackball: A pointing device consisting of a ball housed in a socket containing sensors to detect rotation of the ball about two axes like an upside-down mouse with an exposed protruding ball.

- Joystick: A personal computer peripheral or general control device consisting of a handheld stick that pivots about the base and steers the screen cursor around.
- Digitizer: A locator device used for drawing, painting, or interactively selecting coordinate positions on an object.
- Touch panel: A display device that accepts user input by means of a touch sensitive screen.
- Light pen: A pointing device shaped like a pen and connected to the computer.
- Data glove: An interface device that uses position tracking sensors and fiber optic strands running down each finger and connected to a compatible computer.
- Row matrix: A matrix which has exactly one row is called a row matrix.
- Column matrix: A matrix which has exactly one column is called a column matrix.
- Modulus of a vector: The modulus or magnitude of a vector is the positive number measuring the length of the line representing it.
- Coordinate system: It is a mathematical language that is used to describe geometrical objects analytically.

2.10 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is voice recognition system?
2. State about the input of graphical data.
3. Define the term graphical input functions.
4. Write the painting technique of picture construction.
5. When a matrix is termed as square matrix?
6. How is matrix addition commutative?
7. Define symmetric matrix.
8. How will you define the inverse of a matrix?
9. State about the two order of determinant.
10. What do you understand by vector product?
11. Define unit and free vectors.
12. How is the angle between two vectors evaluated?
13. State about the coplanar point.

Long-Answer Questions

1. Design an interface where there is provision for input from keyboard, scanner, mouse and joystick.
2. Describe the image scanners and its types with the help of diagram.

NOTES

3. Explain how the input function in computer graphics works.
4. Explain the various techniques for constructing interactive pictures.
5. Discuss about the various types of matrix.
6. Briefly explain the scalar multiplication giving appropriate example.
7. Explain the adjoint matrix with the help of appropriate example.
8. Discuss briefly determinants and its properties.
9. Discuss in detail the parallelogram law.
10. Show that the sum of three vectors determined by the medians of a triangle directed from the vertices is zero.
11. What is a position vector? Explain with an example.
12. How is the position of a point represented in spherical coordinates?

2.11 FURTHER READING

Hearn, Donald and M. Pauline Baker. *Computer Graphics*. New Jersey: Prentice Hall.

Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Higher Education.

Foley, James D., Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice*. London: Addison-Wesley Professional.

Mukhopadhyay, Anirban and Arup Chattopadhyay. *Introduction to Computer Graphics and Multimedia*. New Delhi: Vikas Publishing House Pvt Ltd.

UNIT 3 RASTER SCAN GRAPHICS, WINDOWS AND CLIPPING

NOTES

Structure

3.0 INTRODUCTION

The major objective of computer graphics is to model graphic objects in a scene. Such objects comprise several entity primitives, such as points, lines, etc. For displaying these entities on a raster display, the representative sets of pixels of the appropriate intensity or colour must be identified. Digital Differential Analyser (DDA) is used to

NOTES

rasterize lines, triangles and polygons. In the Bresenham's circle generation algorithm, only integer arithmetic is used. The algorithm is based on the differential equation of a circle. The midpoint circle algorithm is an algorithm which is used for determining the points needed to draw a circle. It is a variant of Bresenham's line algorithm. The characters display on a computer screen, there are three basic methods to generate characters on the computer screen: hardware-based, vector-based and bitmap-based methods.

The purpose of clipping procedure is to determine which part of a scene or specifically which points, lines (or curves) or portions of the lines (or curves) of a scene lie inside the clipping window. Line clipping, polygon clipping and text clipping algorithms are also explained in this unit.

In this unit you will study about the derivation of function, digital differential analyzer, Bresenhan's algorithm, circle generation-generation algorithm, real-time scan conversion, run-length encoding, cell-encoding, frame buffers, addressing the raster, line display, character display and polygon filling, simple ordered edge list algorithm, more efficient ordered edge list algorithms, edge fill and edge flag algorithms, seed fill algorithm, fundamentals of antialiasing, convolution integral and antialiasing, two-dimensional clipping, Sutherland-Cohen subdivision line clipping algorithm, midpoint subdivision algorithms.

3.1 OBJECTIVES

After going through this unit, you will be able to:

- State the derivation of function
- Discuss the digital differential analyzer and Bresenham's algorithm
- Explain the process of real-time scan conversion
- Describe the frame buffers
- Explain the character display and polygon filling
- Discuss the more efficient ordered edge list algorithms
- Explain the edge fill and edge flag algorithms
- Describe the seed fill algorithm
- Describe the fundamentals of antialiasing
- Describe clipping, such as line clipping, polygon clipping and text clipping
- Explain the Sutherland-Cohen subdivision line clipping algorithm
- Discuss the midpoint subdivision algorithms

3.2 DERIVATIVE OF A FUNCTION

Algebra of Differentiable Functions

We will now prove the following results for two differentiable functions $f(x)$ and $g(x)$.

NOTES

(1) $\frac{d}{dx}[f(x) \pm g(x)] = f'(x) \pm g'(x)$

(2) $\frac{d}{dx}[f(x) \cdot g(x)] = f'(x)\, g(x) + f(x)\, g'(x)$

(3) $\frac{d}{dx}\left[\frac{f(x)}{g(x)}\right] = \frac{f'(x)\, g(x) - f(x)\, g'(x)}{[g(x)]^2}$

(4) $\frac{d}{dx}[cf(x)] = cf'(x)$, where c is a constant

Where, of course, by $f'(x)$ mean $\frac{d}{dx} f(x)$.

Proof:

(1) $\frac{d}{dx}[f(x) + g(x)]$

$$= \lim_{\delta x \to 0} \frac{[f(x + \delta x) + g(x + \delta x)] - [f(x) + g(x)]}{\delta x}$$

$$= \lim_{\delta x \to 0} \left[\frac{f(x + \delta x) - f(x)}{\delta x} + \frac{g(x + \delta x) - g(x)}{\delta x}\right]$$

$$= \lim_{\delta x \to 0} \frac{f(x + \delta x) - f(x)}{\delta x} + \lim_{\delta x \to 0} \frac{g(x + \delta x) - g(x)}{\delta x}$$

$$= f'(x) + g'(x)$$

Similarly, it can be shown that

$$\frac{d}{dx}[f(x) - g(x)] = f'(x) - g'(x)$$

Thus, we have the following rule:

The derivative of the sum (or difference) of two functions is equal to the sum (or difference) of their derivatives.

(2) $\frac{d}{dx}[f(x)\, g(x)]$

$$= \lim_{\delta x \to 0} \frac{f(x + \delta x)\, g(x + \delta x) - f(x)\, g(x)}{\delta x}$$

$$= \lim_{\delta x \to 0} \frac{g(x + \delta x)\big[f(x + \delta x) - f(x)\big] + f(x)\big[g(x + \delta x) - g(x)\big]}{\delta x}$$

$$= \lim_{\delta x \to 0} \left[g(x + \delta x).\left[\frac{f(x + \delta x) - f(x)}{\delta x}\right] + f(x)\left[\frac{g(x + \delta x) - g(x)}{\delta x}\right]\right]$$

$$= [\lim_{\delta x \to 0} g(x + \delta x)] \lim_{\delta x \to 0} \left[\frac{f(x + \delta x) - f(x)}{\delta x}\right]$$

$$+\left[\lim_{\delta x\to 0} f(x)\right]\left[\lim_{\delta x\to 0}\frac{g(x+\delta x)-g(x)}{\delta x}\right]$$

$$= g(x)\,f'(x) + f(x)\,g'(x).$$

NOTES

Thus, we have the following rule for the derivative of a product of two functions:

The derivative of a product of two functions = (the derivative of first function × second function) + (first function × derivative of second function).

(3) $\dfrac{d}{dx}\left[\dfrac{f(x)}{g(x)}\right]$

$$= \lim_{\delta x\to 0}\frac{\dfrac{f(x+\delta x)}{g(x+\delta x)} - \dfrac{f(x)}{g(x)}}{\delta x}$$

$$= \lim_{\delta x\to 0}\frac{f(x+\delta x)\,g(x) - f(x)\,g(x+\delta x)}{\delta x.g(x+\delta x)\,g(x)}$$

$$= \lim_{\delta x\to 0}\frac{g(x)[f(x+\delta x)-f(x)] - f(x)[g(x+\delta x)-g(x)]}{\delta x.g(x+\delta x)g(x)}$$

$$= \left[\lim_{\delta_x\to 0}\frac{1}{g(x+\delta x)}\cdot\frac{1}{g(x)}\right]\left[\lim_{\delta x\to 0}\frac{g(x)[f(x+\delta x)-f(x)]}{\delta x} - \lim_{\delta x\to 0}\frac{f(x)[g(x+\delta x)-g(x)]}{\delta x}\right]$$

$$= \frac{1}{[g(x)]^2}\cdot[g(x)f'(x) - f(x)g'(x)]$$

$$= \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}.$$

The corresponding rule is stated as under:

The derivative of quotient of two functions=

$$\frac{(Derivative\ of\ Numerator \times Denominator) - (Numerator \times Derivative\ of\ Denominator)}{(Denominator)^2}$$

(4) $\dfrac{d}{dx}[cf(x)] = \lim_{\delta x\to 0}\dfrac{cf(x+\delta x) - cf(x)}{\delta x}$

$$= c\lim_{\delta x\to 0}\left[\frac{f(x+\delta x)-f(x)}{\delta x}\right] = cf'(x).$$

The derivative of a constant function is equal to the constant multiplied by the derivative of the function.

Differential Coefficients of Standard Functions

I. $\dfrac{d}{dx}(x^n) = nx^{n-1}$

Proof: Let, $y = x^n$

Then, $(y + \delta y) = (x + \delta x)^n$

$\Rightarrow$ $\delta y = (x + \delta x)^n - y = (x + \delta x)^n - x^n$

$$= x^n\left[\left(1+\frac{\delta x}{x}\right)^n - 1\right]$$

NOTES

$$= x^n\left[1 + n\left(\frac{\delta x}{x}\right) + \frac{n(n-1)}{2!}\left(\frac{\delta x}{x}\right)^2 + \ldots - 1\right]$$

$$= nx^{n-1}(\delta x) + \frac{n(n-1)}{2!}x^{n-2}(\delta x)^2 + \ldots$$

$$\frac{\delta y}{\delta x} = nx^{n-1} + \text{terms containing powers of } \delta x$$

$$\Rightarrow \quad \lim_{\delta x \to 0}\frac{\delta y}{\delta x} = nx^{n-1}$$

Hence, $\quad \frac{dy}{dx} = nx^{n-1}$.

II. (*i*) $\frac{d}{dx}(a^x) = a^x \log_e a$

(*ii*) $\frac{d}{dx}(e^x) = e^x$

Proof: (*i*) Let, $\quad y = a^x$

then, $\quad y + \delta y = a^{x+\delta x}$

$$\Rightarrow \quad \delta y = a^{x+\delta x} - a^x = a^x(a^{\delta x} - 1)$$

$$\Rightarrow \quad \frac{\delta y}{\delta x} = \frac{a^x(a^{\delta x} - 1)}{\delta x}$$

$$\Rightarrow \quad \frac{dy}{dx} = \lim_{\delta x \to 0}\frac{\delta y}{\delta x} = a^x \lim_{\delta x \to 0}\left(\frac{a^{\delta x} - 1}{\delta x}\right)$$

$$= a^x \lim_{\delta x \to 0}\frac{\left[1 + \delta x(\log a) + \frac{(\delta x)^2(\log a)^2}{2} + \ldots - 1\right]}{\delta x}$$

$$= a^x \lim_{\delta x \to 0}\left(\log a + \text{terms containing } \delta x\right)$$

$$= a^x \log a = a^x \log_e a$$

proves the first part.

(ii) Since $\log_e e = 1$, it follows from result (*i*) that $\frac{d}{dx}e^x = e^x$.

III. $\frac{d}{dx}\log_e x = \frac{1}{x}$

Proof: Let, $\quad y = \log x$

$$\Rightarrow \quad y + \delta y = \log(x + \delta x)$$

$$\Rightarrow \quad \delta y = \log(x + \delta x) - \log x = \log\left(\frac{x + \delta x}{x}\right)$$

$$\Rightarrow \quad \frac{\delta y}{\delta x} = \frac{\log\left(1 + \frac{\delta x}{x}\right)}{\delta x}$$

$$= \frac{1}{x}\cdot\frac{x}{\delta x}\log\left(1 + \frac{\delta x}{x}\right) = \frac{1}{x}\log\left(1 + \frac{\delta x}{x}\right)^{x/\delta x}$$

$$\Rightarrow \quad \lim_{\delta x \to 0}\frac{\delta y}{\delta x} = \frac{1}{x}\lim_{\delta x \to 0}\log_e\left(1 + \frac{\delta x}{x}\right)^{x/\delta x}$$

NOTES

$$= \frac{1}{x}\log_e e = \frac{1}{x}$$

as $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n = e$ and $\log_e e = 1$

Hence, $$\frac{dy}{dx} = \frac{1}{x}$$

IV. $\frac{d}{dx}(\sin x) = \cos x$

Proof: Now, $y = \sin x \Rightarrow y + \delta y = \sin(x + \delta x)$

$$\Rightarrow \quad \delta y = \sin(x+\delta x) - \sin x = 2\cos\left(x+\frac{\delta x}{2}\right)\sin\frac{\delta x}{2}$$

$$\Rightarrow \quad \frac{\delta y}{\delta x} = \frac{2\cos\left[x+\frac{\delta x}{2}\right]\sin\frac{\delta x}{2}}{\delta x} = \cos\left(x+\frac{\delta x}{2}\right)\cdot\left(\frac{\sin\frac{\delta x}{2}}{\frac{\delta x}{2}}\right)$$

$$\Rightarrow \quad \lim_{\delta x\to 0}\frac{\delta y}{\delta x} = \lim_{\delta x\to 0}\left[\cos\left(x+\frac{\delta x}{2}\right)\right]\left[\lim_{\delta x\to 0}\frac{\sin\frac{\delta x}{2}}{\frac{\delta x}{2}}\right]$$

$$= (\cos x)\left[\lim_{\delta x\to 0}\frac{\sin\frac{\delta x}{2}}{\frac{\delta x}{2}}\right] = (\cos x)(1) = \cos x$$

$$\Rightarrow \quad \frac{dy}{dx} = \cos x.$$

V. $\frac{d}{dx}(\cos x) = -\sin x$ [The proof is similar to that of (IV).]

Notes:

1. The technique employed in the proofs of (I) to (IV) above is known as '*ab initio*' technique. We have utilized (apart from simple formulas of Algebra and Trigonometry) the definition of differential coefficient only. We have nowhere used the algebra of differentiable functions.
2. In (VI) to (XII) we shall utilize the algebra of differentiable functions.

VI. $\frac{d}{dx}(c) = 0$, **where** c **is a constant.**

Proof: Let, $y = c = cx^0$.

Then, $$\frac{dy}{dx} = c\left(\frac{dx^0}{dx}\right) = c(0.x^{0-1}) = 0.$$

VII. $\frac{d}{dx}(\tan x) = \sec^2 x$

Proof: Let, $$y = \tan x = \frac{\sin x}{\cos x}$$

$$\frac{dy}{dx} = \frac{\frac{d}{dx}(\sin x)\cos x - \sin x\frac{d}{dx}(\cos x)}{(\cos x)^2}$$

$$= \frac{(\cos x)(\cos x) - \sin x(-\sin x)}{(\cos x)^2}$$

$$s = \frac{\cos^2 x + \sin^2 x}{\cos^2 x} = \frac{1}{\cos^2 x} = \sec^2 x.$$

VIII. $\frac{d}{dx}(\sec x) = \sec x \tan x$

Proof: Let, $y = \sec x = \frac{1}{\cos x}$

Then,
$$\frac{dy}{dx} = \frac{\frac{d}{dx}(1)\cos x - (1)\frac{d}{dx}(\cos x)}{(\cos x)^2}$$
$$= \frac{(0)(\cos x) - (-\sin x)}{\cos^2 x}$$
$$= \frac{\sin x}{\cos^2 x} = \frac{\sin x}{\cos x} \cdot \frac{1}{\cos x} = \tan x \sec x.$$

Before we proceed further, we will introduce hyperbolic functions.

We define hyperbolic sine of x as $\frac{e^x - e^{-x}}{2}$ and write it as

$\sin h\, x = \frac{e^x - e^{-z}}{2}$.

Hyperbolic cosine of x is defined to be $\frac{e^x + e^{-x}}{2}$ and is denoted by $\cos h\, x$. It can be easily verified that

$$\cos h^2 x - \sin h^2 x = 1$$

Since $(\cos h\, \theta, \sin h\, \theta)$ satisfies the equation $x^2 - y^2 = 1$ of a hyperbola, these functions are called hyperbolic functions.

In analogy with circular functions (i.e., $\sin x$, $\cos x$, etc.) we define $\tan h\, x$, $\cot h\, x$ $\sec h\, x$ and $\operatorname{cosec} h\, x$.

Thus, by definition, $\tan h\, x = \frac{\sin h\, x}{\cos h\, x}$, $\cot h\, x = \frac{1}{\tan h\, x}$,

$$\sec h\, x = \frac{1}{\cos h\, x} \text{ and } \operatorname{cosec} h\, x = \frac{1}{\sin h\, x}.$$

IX. $\frac{d}{dx}(\sin h\, x) = \cos h\, x$

Proof: Before proving this result, we say that

$$\frac{d}{dx}(e^{-x}) = -e^{-x},$$

Because,
$$e^{-x} = (e^{-1})^x$$

$\Rightarrow$
$$\frac{d(e^{-x})}{dx} = (e^{-1})^x \log_e (e^{-1}) = e^{-x}(-1) = -e^{-x}$$

Now, let $y = \sin h\, x = \frac{1}{2}(e^x - e^{-x})$

Then,
$$\frac{dy}{dx} = \frac{1}{2}\left(\frac{d}{dx}(e^x) - \frac{d}{dx}(e^{-x})\right)$$

NOTES

$$= \frac{1}{2}[e^x - (-e^{-x})] = \frac{1}{2}(e^x + e^{-x}) = \cos h\, x.$$

X. $\frac{d}{dx}(\cos h\, x) = \sin h\, x$

NOTES

Proof is similar to that of (IX).

XI. $\frac{d}{dx}(\tan h\, x) = \sec h^2 x$

Proof: Let, $y = \tan h\, x = \frac{\sin h\, x}{\cos h\, x}$

$$\frac{dy}{dx} = \frac{\frac{d}{dx}(\sin h\, x)\cos h\, x - \sin h\, x \frac{d}{dx}(\cos h\, x)}{(\cos h\, x)^2}$$

$$= \frac{(\cos h\, x)(\cos h\, x) - (\sin h\, x)(\sin h\, x)}{\cos h^2 x}$$

$$= \frac{\cos h^2 x - \sin h^2 x}{\cos h^2 x} = \frac{1}{\cos h^2 x} = \sec h^2 x.$$

XII. $\frac{d}{dx}(\sec h\, x) = -\sec h\, x \tan h\, x$

Proof: Let, $y = \sec h\, x = \frac{1}{\cos h\, x}$

$$\frac{dy}{dx} = \frac{\frac{d}{dx}(1)\cos h\, x - (1)\frac{d}{dx}(\cos h\, x)}{\cos h^2 x}$$

$$= \frac{(0)(\cos h\, x) - \sin h\, x}{\cos h^2 x}$$

$$= -\left(\frac{\sin h\, x}{\cos h\, x}\right)\left(\frac{1}{\cos h\, x}\right) = -\tan h\, x \sec h\, x.$$

Example 3.1: If $y = x^2 \sin x$, find $\frac{dy}{dx}$.

Solution: This is a problem of the type $\frac{d}{dx}(uv)$.

By applying the formula,

$$\frac{dy}{dx} = \frac{d}{dx}(x^2)\sin x + x^2 \frac{d}{dx}(\sin x)$$

$$= 2x \sin x + x^2 \cos x.$$

Example 3.2: If $y = x^2 \operatorname{cosec} x$, find $\frac{dy}{dx}$.

Solution: We can write y as $\frac{x^2}{\sin x}$

Applying the formula,

$$y = \frac{x^2}{\sin x}$$

$$\Rightarrow \quad \frac{dy}{dx} = \frac{\frac{d}{dx}(x^2)\sin x - x^2 \frac{d}{dx}(\sin x)}{\sin^2 x}$$

$$= \frac{2x \sin x - x^2 \cos x}{\sin^2 x}.$$

Chain Rule of Differentiation

This is the most important and widely used rule for differentiation.

The rule states that:

If y is a differentiable function of z, and z is a differentiable function of x, then y is a differentiable function of x, i.e.,

$$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx}.$$

Proof: Let $y = F(z)$ and $z = f(x)$.

If δx is change in x and corresponding changes in y and z are δy and δz respectively, then $y + \delta y = F(z + \delta z)$ and $z + \delta z = f(x + \delta x)$.

Thus, $\delta y = F(z + \delta z) - F(z)$ and $\delta z = f(x + \delta x) - f(x)$

Now, $$\frac{\delta y}{\delta x} = \frac{\delta y}{\delta z} \cdot \frac{\delta z}{\delta x}$$

$$\Rightarrow \quad \lim_{\delta x \to 0} \frac{\delta y}{\delta x} = \lim_{\delta x \to 0} \frac{\delta y}{\delta z} \lim_{\delta x \to 0} \frac{\delta z}{\delta x}$$

$$\Rightarrow \quad \frac{dy}{dx} = \left(\lim_{\delta z \to 0} \frac{\delta y}{\delta z} \right) \frac{dz}{dx},$$

Since $\delta x \to 0$ implies that $\delta z \to 0$

$$= \frac{dy}{dz} \cdot \frac{dz}{dx}.$$

Corollary: If y is a differentiable function of x_1, x_1 is a differentiable function of x_2, ... , x_{n-1} is a differentiable function of x, then y is a differentiable function of x.

And $$\frac{dy}{dx} = \frac{dy}{dx_1} \frac{dx_1}{dx_2} \ldots \frac{dx_{n-1}}{dx_n}.$$

Proof: Apply induction on n.

Example 3.3: Find the differential coefficient of sin log x with respect to x.

Solution: Put $z = \log x$, then, $y = \sin z$

Now, $$\frac{dy}{dx} = \frac{dy}{dz} \cdot \frac{dz}{dx} = \cos z \cdot \frac{1}{x} = \frac{1}{x} \cos (\log x).$$

Example 3.4: Find the differential coefficient of (*i*) $e^{\sin x^2}$ (*ii*) log sin x^2 with respect to x.

Solution: (*i*) Put $x^2 = y$, $\sin x^2 = z$ and $u = e^{\sin x^2}$

Then, $u = e^z$, $z = \sin y$ and $y = x^2$

By chain rule,

$$\frac{du}{dx} = \frac{du}{dz} \frac{dz}{dy} \frac{dy}{dx}$$

$$= e^z \cos y \, 2x = e^{\sin y} \cos y \, 2x = 2xe^{\sin x^2} \cos x^2.$$

(*ii*) Let, $u = x^2$

$v = \sin x^2 = \sin u$

Then, $y = \log \sin x^2 = \log \sin u = \log v$

So, $\frac{du}{dx} = 2x$, $\frac{dv}{du} = \cos u$ and $\frac{dy}{dv} = \frac{1}{v}$

Then, $$\frac{dy}{dx} = \frac{dy}{dv} \cdot \frac{dv}{du} \cdot \frac{du}{dx}$$

$$= \frac{1}{v} \cdot \cos u \cdot 2x$$

$$= \frac{1}{\sin u} \cos u \cdot 2x = 2x \cot u = 2x \cot x^2.$$

NOTES

Note: After some practice we can use the chain rule, without actually going through the substitutions. For example,

If $y = \log (\sin x^2)$, then $\frac{dy}{dx} = \frac{1}{\sin x^2} \cos x^2 \cdot 2x = 2x \cot x^2$.

Note that we have first differentiated log function according to the formula $\frac{d}{dt}(\log t) = \frac{1}{t}$. Since, here we have log $(\sin x^2)$, so the first term on differentiation is $\frac{1}{\sin x^2}$.

Now, consider $\sin x^2$ and differentiate it according to the formula $\frac{d}{du}(\sin u) = \cos u$. Thus, the second term is $\cos x^2$.

Finally, we differentiated x^2 with respect to x, so, the third term is $2x$.

Then, we multiplied all these three terms to get the answer $2x \cot x^2$. We will illustrate this quick method by few more examples.

Example 3.5: Find $\frac{dy}{dx}$, when $y = e^{(2x+3)^3}$.

Solution: Since, $\frac{d(e^t)}{dt} = e^t$ and $\frac{du^3}{du} = 3u^2$

And $\frac{d(2v)}{dv} = 2$

We get, $\frac{dy}{dx} = e^{(2x+3)^2} \cdot 3(2x+3)^2 \cdot (2.1+0) = 6(2x+3)^2 e^{(2x+3)^2}$

Example 3.6: Differentiate $y = \log [\sin (3x^2 + 5)]$ with respect to x.

Solution: $\frac{dy}{dx} = \frac{1}{\sin (3x^2+5)} \cos (3x^2+5)\, 6x = 6x \cot (3x^2 + 5)$.

Example 3.7: Differentiate $y = \tan^2 (\sqrt{x} + 3)$.

Solution: $\frac{dy}{dx} = 2 \tan (\sqrt{x}+3) \cdot \sec^2 (\sqrt{x}+3) \cdot \frac{1}{2\sqrt{x}} = \frac{\tan (\sqrt{x}+3) \sec^2 (\sqrt{x}+3)}{\sqrt{x}}$

3.3 DIGITAL DIFFERENTIAL ANALYZER

DDA Algorithm

Assume that a line is to be rasterized between given endpoints (*xstart*, *ystart*) and (*xend*, *yend*). Now let us consider the equation of the line as:

$$y = mx + c$$

where m represents the slope of the line and c is the y intercept. This slope can be expressed as:

$$m = \frac{yend - ystart}{xend - xstart}$$

NOTES

Infact any two consecutive points (x_i, y_i) and (x_{i+1}, y_{i+1}) lying on this line segment should satisfy the equation $\frac{y_{i+1} - y_i}{x_{i+1} - x_i} = m$

If we say $y_{i+1} - y_i = \Delta y$ and $x_{i+1} - x_i = \Delta x$, then $\frac{\Delta y}{\Delta x} = m \Rightarrow \Delta y = m\Delta x$

Thus for any given x interval Δx along the line, we can compute the corresponding y interval Δy. Now if

$\Delta x = 1$, i.e., $x_{i+1} - x_i = 1$, i.e., $x_{i+1} = x_i + 1$ then $\Delta y = m$

which implies $y_{i+1} - y_i = m$, i.e., $y_{i+1} = y_i + m$ ....(3.1)

Thus a unit change in x changes y by m which is a constant for a given line. We know that if $x_{i+1} = x_i + 1$, then $y_{i+1} = y_i + m$; the values of x and y (on the line) are defined in terms of their previous values. Initializing (x_i, y_i) with (*xstart*, *ystart*) the line can be generated by incrementing the previous x values by one unit and solving the corresponding y value at each step, till *xend* is reached. At each step, we make incremental calculations based on the previous step. This is what is defined as incremental algorithm and often referred to as the *Digital Differential Analyser (DDA)* algorithm.

While incrementing the values of y by constant m in this DDA method, we have to keep in mind that m, being the slope of the line, can be any real number, negative or positive – so that the calculated y values must be rounded off to the nearest integer for the sake of plotting on the screen.

But an obvious doubt at this point would be why not increment y values by 1 instead of x and accordingly calculate x values from the lines equation. The determining factor here is absolute value of the line's slope, i.e., $|m|$.

If $|m| \leq 1$ which implies $|\Delta y| \leq |\Delta x|$, we sample the line at unit x intervals, as done above.

But if $|m| > 1$ implying $|\Delta y| > |\Delta x|$, a unit step in x creates a step in y that is greater than 1, which is not desirable. In that case we reverse the roles of x and y by sampling at unit y intervals as:

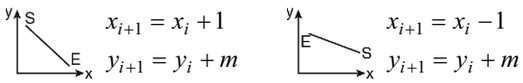
$$\Delta y = y_{i+1} - y_i = 1$$
$$\Rightarrow y_{i+1} = y_i + 1 \quad \text{and}$$
$$\Delta x = \frac{\Delta y}{m} \Rightarrow x_{i+1} - x_i = \frac{1}{m}$$
$$\Rightarrow x_{i+1} = x_i + \frac{1}{m} \qquad \text{....(3.2)}$$

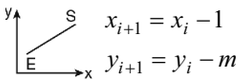
Equations (3.1) and (3.2) are based on the assumption that *xstart* < *xend* and *ystart* < *yend,* i.e., slope is positive. But if it is not so, then we have to apply negative increment as shown below for the other possible cases.

xstart < *xend* *ystart* > *yend*	*xstart* > *xend* *ystart* < *yend*	*xstart* > *xend* *ystart* > *yend*
$\Rightarrow m$ is negative	$\Rightarrow m$ is negative	$\Rightarrow m$ is positive

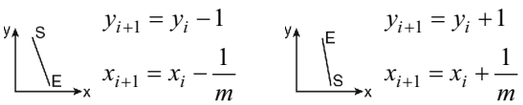
NOTES

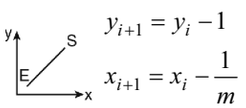
$|m| \le 1$





$|m| > 1$





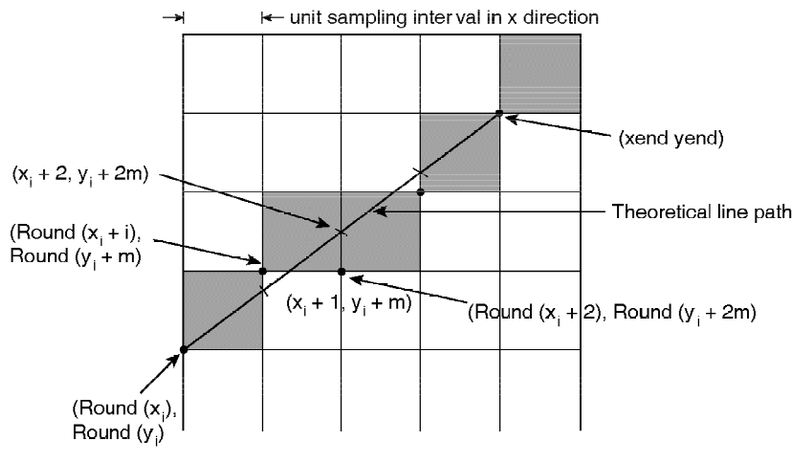


Fig. 3.1 *Incremental Calculation of* x_i, y_i *pixel Coordinates for Scan Converting a Theoretical Line Path*

The *Pseudocode* for rasterizing a line according to DDA logic is presented below. This code works for any line in all the four quadrants.

We assume that points are to be plotted on a bi-level intensity system and a function Setpixel () is such that Setpixel (*x*, *y*, 1), will load the intensity value 1 into the frame buffer at a position corresponding to column *x* along scanline *y* on the screen.

We also assume a function Round () which rounds off the argument to the nearest integer.

```
Input = xstart, ystart, xend, yend
if abs (xend - xstart)   ≥   : check whether | slope | ≤ 1 or > 1 then span
abs (yend - ystart)            = and accordingly set the
abc (xend - xstart)            sampling length
else span = abs (yend - ystart)
Δx  = (xend - xstart)/span   : set the larger of Δx and Δy as one raster unit
                               or unit sampling
Δy  = (yend - ystart)/span     interval
x = xstart                   : initialize x, y pixel coordinate with xstart, ystart
```

```
y = ystart
i = 1
while (i ≤ span)                    : continue loop till xend or yend is reached.
Setpixel (Round (x), Round
(y), 1)
x = x + Δx
y = y + Δy                          : incremental calculation to select next pixel.
i  = i + 1
end while
```


3.4 BRESENHAM'S ALGORITHM

One serious drawback of the DDA algorithm is that it is very time consuming as it deals with a rounding off operation and floating point arithmetic. Moreover successive addition of floating point increment (m or $1/m$) causes accumulation of round off error and eventually a drift away of the plotted pixels from the true line path in case of long line segments.

The algorithm developed by Bresenham is more accurate and efficient compared to DDA algorithm because it cleverly avoids the 'Round' function and scan converts lines using only incremental integer calculation.

This algorithm samples a line by incrementing by one unit either x or y depending on the slope of the line and then selects the pixel lying at least distance from the true line path at each sampling position.

To illustrate Bresenham's approach let us consider a line (L) with positive slope less than 1. So the line will be sampled at unit intervals in X direction. Assuming we have already determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot at next sampling position, i.e., at $x_k + 1$ grid line.

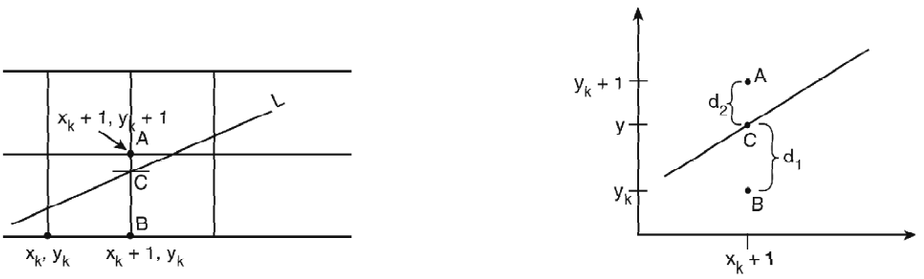


Fig. 3.2 *Bresenham's Line Algorithm*

Our choices are clearly the pixels at $(x_k + 1, y_k + 1)$ and $(x_k + 1, y_k)$, i.e., A and B respectively in the Figure 3.2. Let C be the intersection point of the line L with the gridline $x = x_k + 1$. In Bresenham's formulation, the difference between the vertical distances of A and B to C is computed, and the sign of the difference is used to select the pixel whose distance from C is smaller as the best approximation to the line.

NOTES

Let the vertical distance of pixel B from the true line path, i.e., BC be denoted as d_1 and the vertical distance of pixel A from the true line path, i.e., AC be denoted as d_2 at sampling position $x_k + 1$.

Now as C is a point on the true line path at sampling position $x_k + 1$, hence the coordinates (x, y) of C are given by

$x = x_k + 1$

$y = m(x_k + 1) + c$, considering the equation of the line in the standard slope intercept form, $y = mx + c$

Then,

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

and,

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

$\therefore$ The difference between these two distances is,

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2c - 1 \quad \ldots(3.3)$$

Substituting $m = \frac{\Delta y}{\Delta x}$, where Δy and Δx are the vertical and horizontal separations of the endpoint positions of the line, we get after rearranging

$$\Delta x(d_1 - d_2) = 2\Delta y(x_k) - 2\Delta x(y_k) + 2\Delta y + 2c\Delta x - \Delta x \quad \ldots(3.4)$$

Since $\Delta x > 0$ in this case, $\Delta x(d_1 - d_2)$ has the same sign as $(d_1 - d_2)$. Therefore $\Delta x(d_1 - d_2)$ which involves only integer calculations can be used as a decision parameter to decide the correct pixel, (x_{k+1}, y_{k+1}) next to (x_k, y_k).

We call this $\Delta x(d_1 - d_2)$ as parameter P_k.

If $P_k > 0$ then $d_1 > d_2$ hence choose A implying $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k + 1)$

If $P_k < 0$ then $d_1 < d_2$ hence choose B implying $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k)$

If $P_k = 0$ then $d_1 = d_2$ hence we can choose either A or B.

Now what happens to the value of P_k and therefore the location of pixel (x_{k+2}, y_{k+2}) for the next grid line; both depend of course on whether we choose A or B as (x_{k+1}, y_{k+1}).

The parameter that decides the pixel (x_{k+2}, y_{k+2}) is P_{k+1} which is evaluated similarly from Equation (3.4) as,

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

Subtracting P_k from P_{k+1} to get the incremental difference,

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

NOTES

If A is chosen previously then $y_{k+1} - y_k = 1$

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

Else if B is chosen previously then $y_{k+1} - y_k = 0$

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

Now checking the sign of P_{k+1} the choice is made between pixel at $(x_{k+1} + 1, y_{k+1} + 1)$ and $(x_{k+1} + 1, y_{k+1})$ as the pixel (x_{k+2}, y_{k+2}).

This process is continued till the endpoint of the line is reached.

Thus we see at each step that the algorithm chooses between two pixels based on the sign of the decision parameter calculated in the previous iteration; then it updates the decision parameter simply by incrementing the old value by a factor depending upon the choice of pixel.

This method of obtaining values of successive decision parameters (P_k) using incremental integer calculation avoids direct computation of P_k from Equation (3.5) involving floating point operations.

Since the first pixel is simply the first endpoint (x_0, y_0) of the line we can directly calculate the initial value of P_k, i.e., P_0 for fixing (x_1, y_1).

As the point (x_0, y_0) lies on the true line path, it satisfies the line equation $y = mx + c$.

$$\Rightarrow c = y_0 - mx_0$$
$$= y_0 - \left(\frac{\Delta y}{\Delta x}\right) x_0$$
$$\Rightarrow 2c\Delta x = 2y_0\Delta x - 2x_0\,\Delta y$$

Putting $k = 0$ and replacing $2c\Delta x$ with $2y_0\Delta x - 2x_0\,\Delta y$ in the expression of P_k, we get

$$P_0 = 2\Delta y(x_0) - 2\Delta x(y_0) + 2\Delta y + 2y_0\Delta x - 2x_0\Delta y - \Delta x$$
$$= 2\Delta y - \Delta x$$

We summarize below Bresenham's line drawing algorithm for a line with positive slope less than 1.

Step 1 *Calculate horizontal separation Δx and vertical separation Δy of the given line endpoints. Plot the pixel (x_0, y_0), i.e., the starting endpoint.*

Step 2 *Calculate* $P_0 = 2\Delta y - \Delta x$

Step 3 *At each x_k along the line, starting at k = 0, check the sign of the decision parameter P_k. If $P_k < 0$ the next point to plot (x_{k+1}, y_{k+1}) is (x_{k+1}, y_k) and $P_{k+1} = P_k + 2\Delta y$*

Otherwise the next point to plot is (x_{k+1}, y_{k+1}) and $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Set k = k + 1

NOTES

Step 4 *Repeat Step 3 as long as* $k < \Delta x$.

Example 3.8 To digitalize a line from point (0, 2) to point (4, 5)

Solution:

$$(x_0, y_0) = (0, 2)$$
$$\Delta y = 5 - 2 = 3$$
$$\Delta x = 4 - 0 = 4$$
$$\text{Slope } m = \left(\frac{5-2}{4-0}\right) = \frac{3}{4} < 1$$

Now calculate the successive decision parameters P_k and corresponding pixel positions (x_{k+1}, y_{k+1}) closest to the line path as follows:

$$P_0 = 2\Delta y - \Delta x = 2 > 0$$
$$\Rightarrow x_1 = 1,\ y_1 = y_0 + 1 = 3$$
$$\therefore\ P_1 = P_0 + 2\Delta y - 2\Delta x$$
$$= 2 + 2(3) - 2(4)$$
$$= 0$$
$$\Rightarrow\ x_2 = 2, y_2 = y_1 + 1 = 4$$
$$\therefore\ P_2 = P_1 + 2\Delta y - 2\Delta x$$
$$= 0 + 2(3) - 2(4)$$
$$= -2 < 0$$

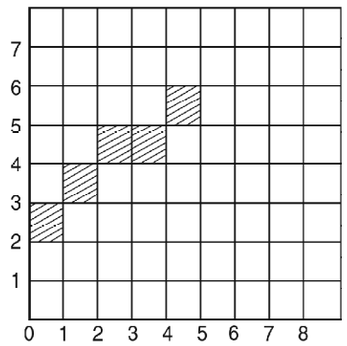
$$\Rightarrow x_3 = 3, y_3 = y_2 = 4$$
$$\therefore\ P_3 = P_2 + 2\Delta y$$
$$= -2 + 2(3)$$
$$= 4 > 0$$
$$\Rightarrow x_4 = 4, y_4 = y_3 + 1 = 5$$

We stop further calculation because the end point (4, 5) is reached.

The result is tabulated as follows:

k	P_k	Coordinate of Pixel to be Plotted	
		x_{k+1}	y_{k+1}
		$x_0 = 0$	y_0 =2 start pixel
0	$P_0 = 2$	$x_1 = 1$	$y_1 = 3$
1	$P_1 = 0$	$x_2 = 2$	$y_2 = 4$
2	$P_2 = -2$	$x_3 = 3$	y_3 =4
3	$P_3 = 4$	$x_4 = 4$	y_4 = 5 end pixel

The following is the shaded pixels (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) – representing the digitalised line, between (0, 2) and (4, 5)



Once again note that this version of Bresenhams algorithm works only for lines with slope between 0 and 1.

When the absolute magnitude of the slope of the line is greater than 1, we step along the direction of *Y* (instead of *X*) in unit steps and calculate successive *x* values using the Brenham's algorithm (refer Figure 3.14)

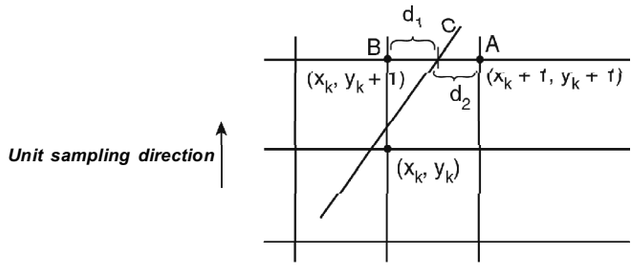


Fig. 3.3 *Calculating x Values when the Slope of the Line is Greater than 1*

However, for negative slopes the procedure is similar except that now one coordinate decreases as the other increases from start point to the endpoint of such lines. Therefore the decreasing coordinate is incremented by –1 instead of +1 in Bresenham's logic (refer Figure 3.4).

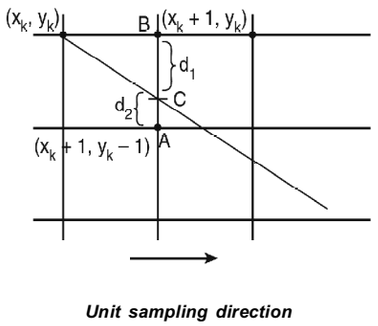


Fig. 3.4 *The Coordinate Decreases from Start Point to End Point*

Taking into account all the above cases, a generalized version of Bresenhams algorithm is given below which will work for any line. The algorithm yields the same result even if we reverse the processing direction (start point to end point) for a given line.

NOTES

Generalized Bresenham's Algorithm (Pseudocode)

Input The line endpoints (x_s, y_s) and (x_e, y_e) which are assumed to be not equal. All variables are assumed to be integers.

The sign( ) function returns –1, 0, 1 as its argument is < 0, = 0, or > 0; sign $(k) = \dfrac{k}{\text{abs}(k)}$

$x = x_s$: initialize variables
$y = y_s$
$\Delta x = \text{abs}\ (x_e - x_s)$
$\Delta y = \text{abs}\ (y_e - y_s)$
$s_1 = \text{sign}\ (x_e - x_s)$
$s_2 = \text{sign}\ (y_e - y_s)$
if $\Delta y > \Delta x$ then
temp $= \Delta x$
$\Delta x = \Delta y$: swap Δx and Δy if absolute slope $|m| > 1$
$\Delta y =$ temp
swap $= 1$: set the flag
else swap = 0
end if
$n = 1$
$P = 2\Delta y - \Delta x$: Initial value of decision parameter (P_0)
Setpixel $(x, y, 1)$: intensify the starting pixel (x_s, y_s)
while $(n \le \Delta x)$: till the endpoint is reached
if $P \ge 0$ then
$x = x + s_1$
$y = y + s_2$: choose the next pixel by incrementing x and y by + 1 or –1
$P = P + 2(\Delta y - \Delta x)$: update decision parameter
else : i.e., if $P < 0$
if swap = 1 then : i.e., | slope | > 1
$y = y + s_2$: choose the next pixel by incrementing y by + 1 or –1 while keeping x same
else : i.e., if | slope | < 1
$x = x + s_1$: choose the next pixel by incrementing x by +1 or –1 while keeping y same
end if : y by +1 or –1 while keeping x same
$P = P + 2\Delta y$: update decision parameter
end if
Setpixel $(x, y, 1)$: set the next pixel
$n = n + 1$
end while : while loop end

NOTES

If you are thinking how we have generalized the decision parameter's (P_{k+1}) expression, here we present the derivations for enthusiastic readers.

Considering $\left.\begin{matrix} s_1 = \text{sign } (x_e - x_s) \\ s_2 = \text{sign } (y_e - y_s) \end{matrix}\right\}$ and $\left.\begin{matrix} \Delta x = \text{abs}\,(x_e - x_s) \\ \Delta y = \text{abs } (y_e - y_s) \end{matrix}\right\}$

Slope $m = \dfrac{(y_e - y_s)}{(x_e - x_s)} = \dfrac{(s_2\,\Delta y)}{(s_1\,\Delta x)}$

Let $\Delta y < \Delta x$, then sampling is along Δx. Considering (x_k, y_k) as the pixel chosen already, we need to choose the next pixel (x_{k+1}, y_{k+1}) between the two probables $(x_k \pm 1,\ y_k \pm 1)$, i.e., $(x_k + s_1,\ y_k + s_2)$ and $(x_k \pm 1,\ y_k)$, i.e., $(x_k + s_1,\ y_k)$. Let us also assume that the y coordinate of the point where the true line path intersects the gridline $x_k \pm 1$ (i.e., $x_k + s_1$) is y.

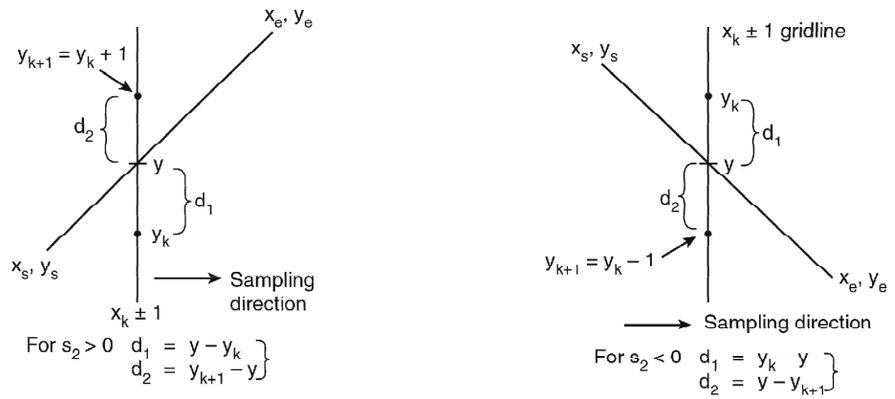


Fig. 3.5 *Bresenham's Algorithm with Positive and Negative Slope*

Then using the symbols as before we can write (refer Figure 3.5)

$y = m(x_k + s_1) + c$

$d_1 = s_2(y - y_k)$

$d_2 = s_2(y_{k+1} - y)$

Replacing the value of y in the above expressions of d_1, d_2, we get

$$\frac{d_1}{s_2} = m(x_k + s_1) + c - y_k$$

$$\frac{d_2}{s_2} = (y_k + s_2) - m(x_k + s_1) - c \qquad [\text{Since} \quad y_{k+1} = y_k + s_2]$$

$$\therefore \quad \frac{(d_1 - d_2)}{s_2} = 2mx_k + 2ms_1 + 2c - 2y_k - s_2$$

$$= 2\frac{s_2 \Delta y}{s_1 \Delta x} x_k + 2\frac{s_2}{s_1}\frac{\Delta y}{\Delta x} s_1 + 2c - 2y_k - s_2$$

$$\Rightarrow \quad \frac{s_1}{s_2}\ \Delta x(d_1 - d_2) = (2\Delta y \quad x_k)s_2 - 2(\Delta \quad xy_k)s_1 + (2\Delta y)s_1 s_2 + (2c\Delta x)s_1 - (\Delta x)\cdot s_1 s_2$$

NOTES

$$\Rightarrow \quad P_k = \Delta x(d_1 - d_2) = (2\Delta y \quad x_k) \quad \frac{s_2^2}{s_1} - (2\Delta x \quad y_k)s_2 + (2\Delta y)s_2^2 + \quad (2c\Delta x)s_2 - (\Delta x)s_2^2$$

$$= \frac{2\Delta y \quad x_k}{s_1} - (2\Delta x \quad y_k)s_2 + (2\Delta y) + (2c\Delta x)s_2 - \Delta x$$

$$\therefore \quad P_{k+1} = \frac{2\Delta y \quad x_{k+1}}{s_1} - (2\Delta x \quad y_{k+1}) \quad s_2 + (2\Delta y) + (2c\Delta x) - \Delta x \qquad [\text{since} \quad s_2^2 = 1]$$

$$\therefore \quad P_{k+1} - P_k = \frac{2\Delta y}{s_1}(x_{k+1} - x_k) - 2\Delta x \quad s_2(y_{k+1} - y_k)$$

For $P_k \geq 0$ $\quad x_{k=1} = x_k + s_1$, $y_{k+1} = y_k + s_2$

$$\Rightarrow \quad P_k = \Delta x(d_1 - d_2) = (2\Delta y \quad x_k) \quad \frac{s_2^2}{s_1} - (2\Delta x \quad y_k)s_2 + (2\Delta y)s_2^2 + \quad (2c\Delta x)s_2 - (\Delta x)s_2^2$$

$$= \frac{2\Delta y \quad x_k}{s_1} - (2\Delta x \quad y_k)s_2 + (2\Delta y) + (2c\Delta x)s_2 - \Delta x$$

$$\therefore \quad P_{k+1} = \frac{2\Lambda y \quad x_{k+1}}{s_1} - (2\Delta x \quad y_{k+1}) \quad s_2 + (2\Delta y) + (2c\Delta x) - \Delta x \qquad [\text{since} \quad s_2^2 = 1]$$

$$\therefore \quad P_{k+1} - P_k = \frac{2\Delta y}{s_1}(x_{k+1} - x_k) - 2\Delta x \quad s_2(y_{k+1} - y_k)$$

For $P_k \geq 0$ $\quad x_{k=1} = x_k + s_1$, $y_{k+1} = y_k + s_2$

$$\Rightarrow P_{k+1} = P_k + (2\Delta y)\frac{s_1}{s_2} - (2\Delta x)(s_2)^2 = P_k + 2(\Delta y - \Delta x)$$

for $P_k < 0$ $\quad x_{k+1} = x_k + s_1$, $y_{k+1} = y_k$

$$\Rightarrow P_{k+1} = P_k + (2\Delta y)\frac{s_1}{s_1} - (2\Delta x)\, s_2(0) = P_k + 2\Delta y$$

Putting $m = \frac{s_2\, \Delta y}{s_1\, \Delta x}$, $y = y_s$ and $x = x_s$ in $y = mx + c$

we get, $(2c\Delta x) = (2\Delta x\, y_s) - (2\Delta y\, x_s)\frac{s_2}{s_1}$

Now replacing this value of $(2c\,\Delta x)$ in the expression of P_k and putting $k = 0$, we get the initial value of decision parameter.

$$P_0 = \frac{2\Delta y \quad x_0}{s_1} - (2\Delta x \quad y_0)s_2 + 2\Delta y + (2\Delta x \quad y_s)s_2 - (2\Delta y \quad x_s) \quad \frac{s_2^2}{s_1} - \Delta x$$

$$= 2\Delta y - \Delta x \quad \left[\text{Since,} \quad x_0 = x_s,\ y_0 = y_s \right\} \quad \text{and} \quad s_2^2 = 1\right]$$

In similar manner the algorithm can be proved for $\Delta y > \Delta x$ case where sampling is done along Δy and $P_0 = 2\Delta x - \Delta y$

$$\left.\begin{aligned}\text{For } P_k \geq 0,\ x_{k+1} &= x_k + s_1 \\ y_{k+1} &= y_k + s_2 \text{ and} \\ P_{k+1} &= P_k + 2\Delta x - 2\Delta y\end{aligned}\right\}$$

$$\left.\begin{aligned}\text{For } P_k < 0,\ y_{k+1} &= y_k + s_2 \\ x_{k+1} &= x_k \text{ and} \\ P_{k+1} &= P_k + 2\Delta x\end{aligned}\right\}$$

Example 3.9

Given,

$xstart = 0$ $ystart = 0$

$xend = -4$ $yend = -8$

To Find out using generalized Bresenham's algorithm the pixel locations approximating a line between the given points.

Solution : $\Delta x = \text{abs}(-4-0) = 4$

$\Delta y = \text{abs}(-8-0) = 8$

$s_1 = \text{sign}\ (-4-0) = -1,\ s_2 = \text{sign}\ (-8-0) = -1$

Since $\Delta y > \Delta x$, i.e., slope $= \dfrac{\Delta y}{\Delta x} = \dfrac{8}{4} = 2 > 1$

Δy and Δx are interchanged, i.e., $\Delta x = 8$ and $\Delta y = 4$

and flag swap $= 1$

$n = 1$

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

$= 2 \times 4 - 8 = 0$ [Since swap $= 1$]

Setpixel (0, 0, 1)

Since $n \leq (\Delta x = 8)$

Since $P = 0$

$x = x + s_1 = 0 + (-1) = -1$

$y = y + s_2 = 0 + (-1) = -1$

$P = P + 2(\Delta y - \Delta x)$

$= 0 + 2(4-8) = -8$

Setpixel $(-1, -1, 1)$

$n = 1 + 1 = 2$

Since $n < 8$

Since $(P = -8) < 0$ and swap $= 1$ so

$y = y + s_2$

$= -1 + (-1) = -2$

$P = P + 2\Delta y$

$= -8 + 2 \times 4 = 0$

Setpixel $(-1, -2, 1) = 0$ [x remains unchanged as -1 in previous iteration]

$n = 2 + 1 = 3$

NOTES

Since $n < 8$

Since $P = 0$

$x = x + s_1 = -1 - 1 = -2$

$y = y + s_2 = -2 - 1 = -3$

$P = P + 2(\Delta y - \Delta x)$

$= 0 + 2(4 - 8) = -8$

Setpixel $(-2, -3, 1)$

$n = 3 + 1 = 4$

Since $n < 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -3 - 1 = -4$

$P = P + 2\Delta y$

$= -8 + 2 \times 4 = 0$

Setpixel $(-2, -4, 1)$ [x remains unchanged as -2 in previous iteration]

$n = 4 + 1 = 5$

Since $n < 8$

Since $P = 0$

$x = x + s_1 = -2 - 1 = -3$

$y = y + s_2 = -4 - 1 = -5$

$P = P + 2(\Delta y - \Delta x) = 0 + 2(4 - 8) = -8$

Setpixel $(-3, -5, 1)$

$n = n + 1 = 5 = 1 = 6$

Since $n < 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -5 - 1 = -6$

$P = P + 2\Delta y = -8 + 2 \times 4 = 0$

Setpixel $(-3, -6, 1)$ [$x = -3$ unchanged]

$n = n + 1 = 6 + 1 = 7$

Since $n < 8$

Since $P = 0$ so

$x = x + s_1 = -3 - 1 = -4$

$y = y + s_2 = -6 - 1 = -7$

$P = P + 2(\Delta y - \Delta x) = 0 + 2(4 - 8) = -8$

Setpixel $(-4, -7, 1)$

$n = n + 1 = 7 + 1 = 8$

Since $n = 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -7 - 1 = -8$

$P = P + 2\Delta y = -8 + 2 \times 4 = 0$

Setpixel $(-4, -8, 1)$ [$x = -4$ unchanged]

$n = n + 1 = 8 + 1 = 9$

Since $n > 8$ hence no more iteration is done.

NOTES

3.5 CIRCLE GENERATION-BRESENHAM'S ALGORITHM

As with Bresenham's line generation algorithm, the sign of a decision parameter is checked for finding the closest pixel to the circumference of a circle at each sampling step in Bresenham's circle rasterizing algorithm.This algorithm is better than the earlier two algorithms (Polynomial and Parametric method) because it avoids trigonometric and square root calculations by adopting only integer operation involving squares of the pixel separation distances. For a given radius R and screen centre position (x_c, y_c), we can first calculate pixel positions around a circle path centred at the origin (0, 0).Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y, as done in the Parametric method. Keeping in mind the advantage of generating a complete circle by only computing points on a single octant, we consider the first octant of an origin-centred circle.Notice that, if the algorithm begins at $x = 0, y = R$, then for clockwise generation of the circle y is a monotonically decreasing function of x in the first quadrant. For any known point (x_k, y_k) on the circle, for a clockwise generation of the circle, there are only three possible candidate pixels for selection as the next pixel which best represents the circle: (1) adjacent pixel horizontally to the right $–H(x_k+1, y_k)$, (2) adjacent pixel diagonally downward to the right $–D(x_k+1, y_k–1)$ and (3) adjacent pixel vertically downward $–V(x_k, y_k–1)$. Out of these three pixels, the algorithm chooses the one for which the distance from the true circle is minimum.

Now consider the pixel P w.r.t. the circular arc shown in Figure. 3.6.

Let d_P denote the quantity $|OP^2 – OX^2|$. In this case $d_P = |OP^2 - OX^2|$

$$= x^2 + y^2 - R^2$$

Similarly, if we introduce the terms d_H, d_D and d_V corresponding to pixels H, D and V respectively then we can write.

$$d_H = |OH^2 - R^2| = |(x_k + 1)^2 + (y_k)^2 - R^2|$$
$$d_D = |OD^2 - R^2| = |(x_k + 1)^2 + (y_k - 1)^2 - R^2|$$
$$d_V = |OV^2 - R^2| = |(x_k)^2 + (y_k - 1)^2 - R^2|$$

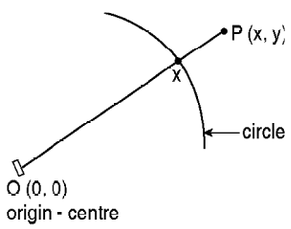


Fig. 3.6 *Bresenham's Method*

NOTES

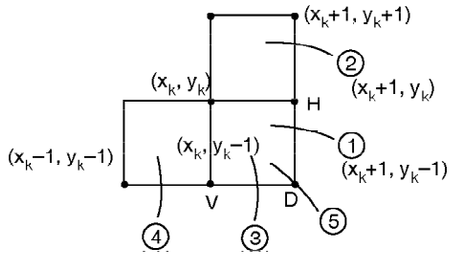


Fig. 3.7 *Five Possible Cases*

Though the (x_{k+1}, y_{k+1}) pixel has to be chosen among (x_k+1, y_k), (x_k+1, y_k-1) and (x_k, y_k-1), there are five possible cases to be considered regarding position of the true circle path in the vicinity of the point (x_k, y_k) as illustrated in Figure 3.7.

It is clear from Figure 2.3 that

For **case** (1) and (2) the diagonal pixel D is inside the circle, implying

$$OD^2 < R^2, \text{ i.e., } OD^2 - R^2 < 0$$

For **case** (3) and (4) the diagonal pixel D is outside the circle, implying

$$OD^2 > R^2, \text{ i.e., } OD^2 - R^2 > 0$$

For **case** (5) the diagonal pixel D is on the circle, implying

$$OD^2 = R^2, \text{ i.e., } OD^2 - R^2 = 0$$

Let us now study the five cases separately while referring to the Figure 3.7.

Case 1 To choose between $H(x_k + 1, y_k)$ and $D(x_k + 1, y_k - 1)$

$$\text{Let } \delta_{HD} = d_H - d_D$$

$$\therefore \delta_{HD} = |OH^2 - R^2| - |OD^2 - R^2|$$

$$= (OH^2 - R^2) - (R^2 - OD^2)$$ [since, $OD^2 < R^2$ $OH^2 > R^2$ as D is inside the circle while H is outside the circle]

$$= OH^2 + OD^2 - 2R^2$$

$$= \{(x_{k+1} + 1)^2 + (y_k)^2\} + \{(x_k + 1)^2 + (y_k - 1)^2\} - 2R^2$$

$$= 2(x_k + 1)^2 + 2(y_k - 1)^2 - 2R^2 + 2y_k - 1$$

$$= 2\{(x_k + 1)^2 + (y_k - 1)^2 - R^2\} + 2y_k - 1$$

$$= 2\Delta D_k + 2y_k - 1$$

$$\text{where } \Delta D_k = OD^2 - R^2 = (x_k + 1)^2 + (y_k - 1)^2 - R^2$$

If $\delta_{HD} < 0$, then $d_H < d_D$, which implies the horizontal pixel is closer to the actual circle than the diagonal pixel. So choose $H(x_k + 1, y_k)$,

Conversely, if $\delta_{HD} > 0$, then choose $D(x_k + 1, y_k - 1)$.

However, if $\delta_{HD} = 0$, then the distances are equal; conventionally choose H.

Case 2 Since y is a monotonically decreasing function of x along the clockwise generated circle-path in first quadrant, we cannot choose $(x_k + 1, y_k + 1)$ as the pixel next to (x_k, y_k) even if it is closer to the circle than $H(x_k + 1, y_k)$. So for case(2), always choose the horizontal pixel $H(x_k + 1, y_k)$ as $(k + 1)$ th pixel.

Also note that $\delta_{HD} < 0$ for this case.

Case 3 To choose between V(x_k, $y_k - 1$) and D($x_k + 1$, $y_k - 1$)

Let, $\delta_{VD} = d_V - d_D$

$$\therefore \quad \delta_{VD} = |OV^2 - R^2| - |OD^2 - R^2|$$

$$= (R^2 - OV^2) - (OD^2 - R^2) \qquad [\text{since, } OD^2 > R^2, OV^2 < R^2 \text{ as } D \text{ is}$$

$$= 2R^2 - OV^2 - OD^2 \qquad \text{outside the circle while } V \text{ is inside the circle}]$$

$$= 2R^2 - \{(x_k)^2 + (y_k - 1)^2\} - \{(x_k + 1)^2 + (y_k - 1)^2\}$$

$$= 2R^2 - 2(x_k + 1)^2 - 2(y_k - 1)^2 + 2x_k + 1$$

$$= 2x_k + 1 - 2\{(x_k + 1)^2 + (y_k - 1)^2 - R^2\}$$

$$= 2x_k - 2\Delta D_k + 1 \qquad [\text{since, } \Delta D_k = (x_k + 1)^2 + (y_k - 1)^2 - R^2]$$

If $\delta_{Vd} < 0 \Rightarrow d_V < d_D$ hence choose $V(x_k, y_k - 1)$

else if $\delta_{VD} < 0 \Rightarrow d_V > d_D$ hence choose $D(x_k + 1, y_k - 1)$

else if $\delta_{VD} < 0 \Rightarrow d_V = d_D$ then conventionally choose D.

Case 4 Since along the circle y monotonically decreases and x monotonically increases, we cannot choose the pixel $(x_k - 1, y_k - 1)$ with lower x-value than the preceding pixel (x_k, y_k), even if it is closer to the circle than the vertical pixel $V(x_k, y_k - 1)$. So our choice is V as the $(k + 1)$th pixel.

Also note that $\delta_{VD} < 0$ for this case.

Case 5 This is the case when the diagonal pixel D lies on the actual circle. So, the obvious choice is $D(x_k+1, y_k-1)$.

Also note that for this case,

$$\delta_{HD} = OH^2 - OR^2 > 0 \quad [\text{since, } OD^2 = R^2 \ \& \ OH^2 > R^2]$$

$$\delta_{VD} = R^2 - OV^2 > 0 \quad [\text{since, } OD^2 = R^2 \ \& \ R^2 > OV^2]$$

We will call ΔD as the *decision parameter* because by checking the sign (<0, >0, or, = 0) of $\Delta D = OD^2 - R^2$ at each step, we can determine the cases(s) it corresponds to and proceed accordingly.

We can update the value of decision parameter ΔD at every step, following the incremental approach, i.e. by adding a factor (depending on the choice of pixel) to the old value.

1. If $x_{k+1} = x_k + 1, \quad y_{k+1} = y_k,$ i.e., the horizontal pixel H

$$\Delta D_{k+1} = (x_{k+1} + 1)^2 + (y_{k+1} - 1)^2 - R^2 \qquad [\text{since, } \Delta D_k = (x_k + 1)^2 + (y_k - 1)^2 - R^2]$$

$$= (x_k + 2)^2 + (y_k - 1)^2 - R^2$$

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

$$= \Delta D_k + 2x_{k+1} + 1$$

NOTES

2. If $x_{k+1} = x_k + 1, \quad y_{k+1} = y_k - 1,$ i.e., the diagonal pixel D

$$\Delta D_{k+1} = (x_k + 2)^2 + (y_k - 2)^2 - R^2$$
$$= (x_k + 1)^2 + (y_k - 1)^2 - R^2 + 2x_{k+1} - 2y_{k+1} + 2$$
$$= \Delta D_k + 2x_{k+1} - 2y_{k+1} + 2$$

3. If $x_{k+1} = x_k, \quad y_{k+1} = y_k - 1,$ i.e., the vertical pixel D

$$\Delta D_{k+1} = (x_k + 1)^2 + (y_k - 2)^2 - R^2$$
$$= (x_k + 1)^2 + (y_k - 1)^2 - R^2 - 2y_{k+1} + 1$$
$$= \Delta D_k - 2y_{k+1} + 1$$

The pseudocode summarizing the above steps is discussed in the following sub-sections.

Bresenham Algorithm (Pseudocode)

The centre of the circle and the starting point are assumed to be located precisely at pixel elements. The radius is also assumed to be an integer.

```
input :   x_c , y_c   R
      x = 0 ,   y = R            : the starting pixel (x_0, y_0)
```

$\Delta D = 2(1 - R)$: initial value of decision parameter

$$\Delta D_0 = (0 + 1)^2 + (R - 1)^2 - R^2$$
$$= 2 - 2R$$

```
while (y > x)                    : continue till the diagonal axis in the
                                   first quadrant is reached
Setpixel ((x_c + x), (y_c + y), 1) : plot the computed pixel in the 1st
                                   octant
Setpixel ((x_c - x), (y_c + y), 1)
Setpixel ((x_c + x), (y_c - y), 1)
Setpixel ((x_c - x), (y_c - y), 1)
Setpixel ((x_c + y), (y_c + x), 1) : plot the symmetric pixels in the other
                                   octants
Setpixel ((x_c - y), (y_c + x), 1)
Setpixel ((x_c + y), (y_c - x), 1)
Setpixel ((x_c - y), (y_c - x), 1)
```

if $\Delta D < 0$ then : $OD^2 < R^2$

$\delta = 2\Delta D + 2y - 1$: $\delta = \delta_{HD}$

if $\delta \le 0$ then

$x = x + 1$: choose horizontal pixel and update ΔD accordingly

$\Delta D = \Delta D + 2x + 1$

NOTES

```
else
x = x + 1
y = y - 1                                   : choose diagonal pixel and
                                              update ΔD accordingly
```

$\Delta D = \Delta D + 2x - 2y + 2$ for $\delta_{HD} > 0$

```
endif
elseif ΔD  > 0 then                         : OD² > R²
```

$\delta = 2x - 2\Delta D + 1$: $\delta = \delta_{VD}$

```
if δ < 0 then
```

$y = y - 1$: choose vertical pixel and update ΔD accordingly

$\Delta D = \Delta D - 2y + 1$

```
else
x = x + 1                                   : choose diagonal pixel and
update                                        ΔD accordingly
y = y - 1                                   for δVD ≥ 0
```

$\Delta D = \Delta D + 2x - 2y + 2$

```
endif
else
x = x + 1
y = y - 1                                   : choose diagonal pixel and
update                                        ΔD accordingly
```

$\Delta D = \Delta D + 2x - 2y + 2$ for $\Delta D = 0$

```
endif
endwhile
```

Example 3.10: To find out (using Bresenham's algorithm) the pixel location approximating the first octant of a circle having centre at (4, 5) and radius 4.

$$x_c = 4,\ y_c = 5,\ R = 4$$
$$x = 0$$
$$y = R = 4$$
$$\Delta D = 2(1 - R) = 2(1 - 4) = -6$$

Iteration 1 Since $(y = 4) > (x = 0)$

`Setpixel ((xc + x), (yc + y),1)` $\Rightarrow$ `Setpixel (4, - 9,1)`

$$x = x + 1 = 0 + 1 = 1$$
$$y = 4 \text{ (unchanged)}$$
$$\Delta D = \Delta D + 2x + 1 = -6 + 2(1) + 1 = -3$$

Iteration 2 Since $(y = 4) > (x = 1)$

`Setpixel ((xc + x), (yc + y),1)` $\Rightarrow$ `Setpixel (5, 9, 1)`

NOTES

Since $(\Delta D = -3) < 0$

$\delta = \delta_{HD} = 2\Delta D + 2y - 1 = 2(-3) + 2(4) - 1 = 1$

since $\delta > 0$

$x = x + 1 = 1 + 1 = 2$

$y = y - 1 = 4 - 1 = 3$

$\Delta D = \Delta D + 2x - 2y + 2 = -3 + 2(2) - 2(3) + 2 = -3$

Iteration 3 Since $(y = 3) > (x = 2)$

`Setpixel ((`x_c + x)`, (`y_c + y)`,1)` $\Rightarrow$ `Setpixel (6, 8, 1)`

Since $(\Delta D = -3) < 0$

$\delta = \delta_{HD} = 2\Delta D + 2y - 1 = 2(-3) + 2(3) - 1 = -1$

since $\delta < 0$

$x = x + 1 = 2 + 1 = 3$

$y = 3$ (unchanged)

$\Delta D = \Delta D + 2x + 1 = -3 + 2(3) + 1 = 4$

Iteration 4 Since $(y = 3) = (x = 3)$

`Setpixel ((`x_c + x)`, (`y_c + y)`,1)` $\Rightarrow$ `Setpixel (7, 8, 1)`

Since $(\Delta D = 4) > 0$

$\delta = \delta_{VD} = 2x - 2\Delta D + 1 = 2(3) - 2(4) + 1 = -1$

since $\delta < 0$

$y = y - 1 = 3 - 1 = 2$

$x = 3$ (unchanged)

$\Delta D = \Delta D - 2y + 1 = 4 - 2(2) + 1 = 1$

Iteration 5 Since $(y = 2) \ngeq (x = 3)$ iteration 5 does not run. So, the pixel locations in the first octant of the given circle are (4, 9), (5, 9), (6, 8) and (7, 8).

Check Your Progress

1. How will you calculate the derivative of a product of two functions?
2. What do you understand by the chain rule of differentiation?
3. What is the function of the digital differential analyser?
4. What input values are taken in generalized Bresenham's algorithm?
5. What do you understand by Bresenham's circle generation algorithm?

3.6 SCAN CONVERSION-GENERATION OF THE DISPLAY

The process of converting the rasterized picture stored in a frame buffer to the rigid display pattern of video is called as scan conversion or rasterization. It is a continuous-to- discrete transformation. A display list contains the information about every pixel on the screen. There are four ways of doing this scan conversion, namely-

1. Solid Area Scan Conversion.

NOTES

2. Real-time Scan Conversion
3. Run-length encoding
4. Cell organization.

Let us discuss them one by one now.

3.6.1 Solid Area Scan Conversion

The process of generation of solid areas from simple edge or vertex descriptions is known as solid area scan conversion. It means filling the area or closed graph or polygon. This means that we are making the area solid. For this we have discussed algorithms like seed fill, boundary fill, scanline fill to fill the polygon. It is also called as contour filling.

3.6.2 Real-time Scan Conversion

In this technique, the picture is randomly represented in terms of visual attributes and geometric properties. Color, shade and intensity are the visual attributes whereas (x, y) coordinates, slopes are the geometric properties of the picture. Now, our display processor scans through both of these properties and then calculates the intensity of every pixel on the screen during the presentation of each frame. Here, lesser memory is required for scan conversion. It is required to hold only the display list and one scan line. It facilitates dynamic picture representation. The complexity of picture is limited by the speed of display processor.

How to implement this?

Firstly, the entire display list is processed to obtain the intersection of each line in the display list. Of each line in the display list, with a particular scanline each time a scanline is displayed. We need an active edge list for this. This list contains those lines in a picture which intersect scanline.

3.6.3 Run-length Encoding

This encoding considers the pictures linearly or one-dimensionally. In many pictures, we observe that the large number of pixels in the picture have the same intensity or color. So, a picture can be defined by pixel intensity and the number of pixels on given scanline with the same intensity. Such a representation of picture is called as run-length encoding. The number of pixels give us the run-length as:

Intensity	Run-length

Consider an example to understand this.

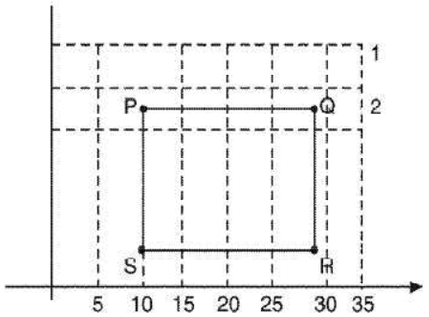


Fig. 3.8 *Run-length encoding.*

NOTES

In Fig. 3.8, scanline-1 has 35 pixels and all are zero because no line is intersected by an edge. Now, for this scanline-1, the run-length encoding will be as follows:

0	35

Similarly, for scanline-2, we get :

0	10	1	18	0	7

This means that-

(a) First ten pixels are zeros (OFF).

(b) Next 18 pixels are 1 (ON).

(c) Remaining 7 pixels are all zeros (OFF).

This process continues for all scanlines. With this scheme, lot of saving of memory can be achieved. But it has an overhead of encoding and decoding.

Extension of this scheme

We can further extent this scheme for colored and solid pictures also. Here, the run length is as follows-

Run length	Red Intensity	Green Intensity	Blue Intensity

For example, | 17 | 0 | 1 | 0 | would mean that 17 successive pixels will be drawn in green color only as we have 1 in Green Intensity field.

This technique is being used for internet images also.

Advantages of Run-length Encoding

1. Picture is stored in compressed form.
2. Saves storage space for animated sequences.
3. Saves transmission time.
4. Lesser memory requirement.

Disadvantages of Run-Length Encoding

1. Adding or deleting lines from the picture is difficult.
2. There is an extra overhead of encoding and decoding.
3. The storage requirement may approach twice that for pixel by pixel storage for short runs.

In a pixel storage, one piece of information for each pixel would require 225 intensity values for (15 × 15) raster. On the other hand, with non-length encoding the complete picture is encoded with only 102 numbers.

3.6.4 Cell Organization (or Encoding)

We can encode the raster as a set of rectangular or square area called as cell encoding. Our picture is considered as areawise cells. The entire screen is divided

into cells. Please understand that these cells will have minimum amount of information.

For example, the screen may be divided into area containing (8 × 8) pixels i.e., cell height and cell width, both are 8 pixels wide. This means that there are (64 × 64) cells for a (512 × 512) display.

These pixel patterns are stored in ROM. Our display processor accesses this information.

This scheme has been extended for-

(a) Line drawings.

(b) Color display and solid image representations.

The first extension (part-a) is achieved by storing the line segment patterns along with the character patterns in ROM. So, by combining the line segment pattern in adjacent cells, we can construct a complete line.

Note: As per Bresenham's line drawing algorithm there are at most $2n - 1$ cell patterns which represent line segments. For a (8 × 8) cell, we have to store 255 patterns of line segments.

These patterns of line segments can be further reduced by using techniques of translation, reflection and masking as given by Jordan and Barrett. According to these scientists, now we need only 108 line segment patterns for an (8 × 8) cell.

A cell encoded character 'C' is shown in Fig. 3.9.

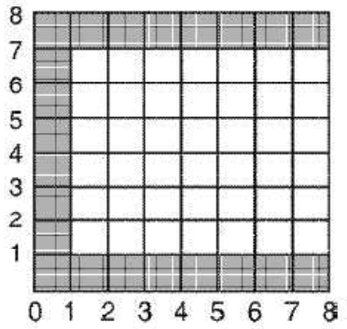


Fig. 3.9 *Character 'C'.*

3.7 FRAME BUFFERS

In the early days of personal computers, the amount of information displayed by the display unit such as a monitor was less because computer graphics did not have much scope. For example, a screen of monochrome (black and white) text requires only about 2 KB of Random Access Memory (RAM). The special parts of the Upper Memory Area (shortened as UMA) were devoted to hold this animation or video data. As the requirement for video memory increased into megabyte (MB) range, it made more sense to put the memory on the video card itself. In fact, this was necessary to defend the design limitations of personal computers. Frame buffer can be defined as the video memory (i.e., RAM) that holds the image that is displayed on the monitor. The amount of memory needed to hold the image depends mainly on the resolution (number of rows and columns

NOTES

in which the screen is divided) of the screen image and also the colour depth (brightness) used per pixel. The formula to calculate how much video memory is needed at a given resolution and bit depth is quite simple and determined as follows:

Memory in megabytes = (X-resolution × Y-resolution × Number-of-bits-per-pixel)/(8 × 1024 × 1024)

Practically we require more memory than this formula calculates. One major reason is that video cards are available only in certain memory configurations (in terms of whole megabyte). For example, we can not order a card with 1.65 megabytes of memory. We have to use a standard 2 megabytes card available in the market. Another reason is that several video interface cards, especially high end accelerators and 3-D cards use memory for frame buffer as well as for computation. Thus we need much more memory than is required strictly to hold the screen image.

3.7.1 Addressing the Raster

A raster addressed display (e.g., a CRT) works by scanning across the entire display in sequence while modulating control signal to activate each pixel as it is scanned. This display uses persistence of the pixel element (e.g., phosphor) to maintain the pixel state until the scan can visit that pixel again. There are only three control signals required for this to work: a horizontal scan control signal, a vertical scan control signal, and an intensity control signal. Timing between these signals is very important, else the image on the screen will show artifacts.

3.7.2 Line Display

This algorithm is an extension of the method of scan converting a solid polygon and can be treated as a special case of Z-buffer algorithm. The working principle of the algorithm can be summarized through following steps:

I. For each scan line (y = constant) of the pixel grid:

- **a.** Set the frame buffer to the background intensity.
- **b.** Set the Z-buffer to z_{min}.

II. For each polygon in the scene (in arbitrary order):

- **a.** Find intersection, if any, of the polygon's 2D projection with the scan line. Intersection occurs in pair as shown in Figure 3.10.
- **b.** For each pixel (in left to right order) on the scan line between the intersection pairs compare its depth (z value) to the depth recorded in the z-buffer at that location.
  - **(i)** If the pixel depth is greater than that in the Z-buffer then this line segment from left intersection to current pixel is the currently visible line segment. Hence, the polygon attribute for this line segment is written to the buffer and the Z-buffer is updated.

When all the polygons of the scene are processed, the scan line frame buffer contains the hidden surface solution for that scan line. Thus, the scan line Z-buffer algorithm creates an image one scan line at a time.

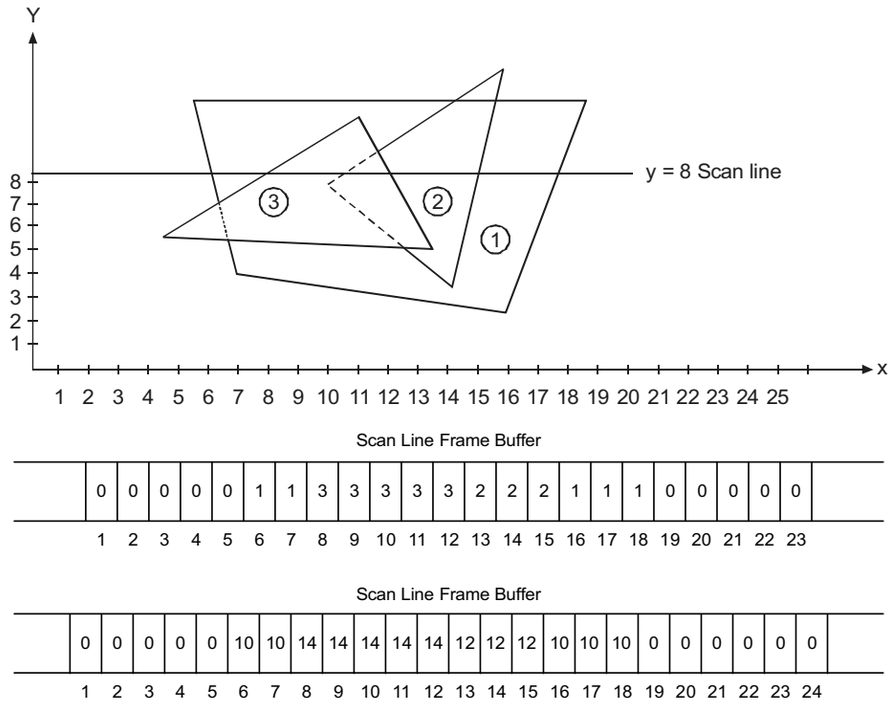


Fig. 3.10 *Frame Buffer and Z-Buffer Values*

In Figure 5.7, the frame buffer and Z-buffer values are shown after scanning polygons (1), (2), (3) in the scene along scan line $y = 4$. These polygons having uniform depths 10, 12 and 14 respectively, and are represented by intensity values 1, 2 and 3, respectively.

3.7.3 Character Display

There are three basic methods to generate characters in computer screen: (1) hardware based (2) vector based and (3) bitmap based methods. In the hardware based method the logic for generating character is built into the graphics terminal. Though the generation time is less but the typefaces are limited due to hardware restrictions.

In the vector based method the characters are developed using a set of polylines and splines that approximates the character outline (refer Figure 3.11). This form of character representation is completely device independent; memory requirement is less as boldface, italics or different size can be produced by manipulating the curves outlining the character shapes; it does not require separate memory blocks for each variation.

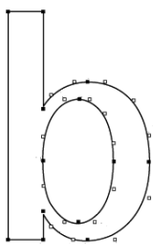


Fig. 3.11 *Vector-Based Font Generated with Line and Spline Passing Through Control Points*

In the bitmap based method small rectangular bitmap called *character mask* (containing binary values 1 and 0) is used to store pixel representation of each character in a frame buffer area known as *font cache* (refer Figure 3.12). Relative pixel locations corresponding to a character bitmap are marked depending on the size, face and style (font) of character. Size of each character masks ranges from 5 × 7 to 10 × 12. A single font in 10 different font sizes and 4 faces (normal, **bold**, *italic*, ***bold italic***) would require 40 font caches. Characters are actually generated on display by copying the appropriate bitmaps from the frame buffer to the desired screen positions. A mask is referenced by the coordinate of the origin (lower left corner) of the mask with respect to frame buffer addressing system.

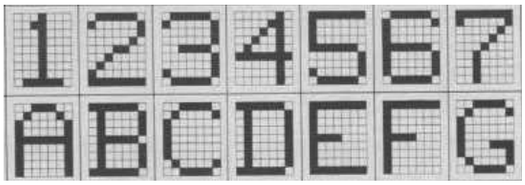


Fig. 3.12 *Bitmapped Font*

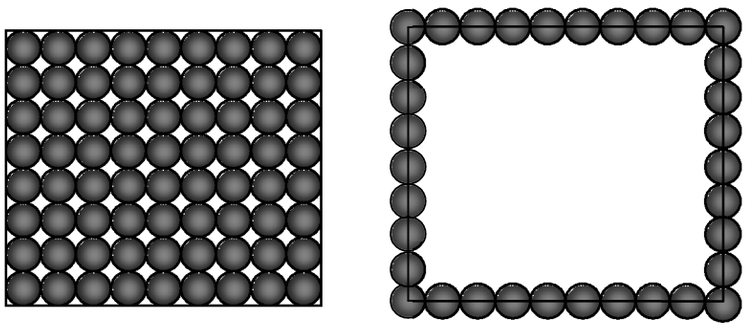
In bitmap based method a bold face character is obtained by writing the corresponding 'normal' character mask in consecutive frame buffer α locations. Italics character is produced by necessary skewing of 'normal' character mask while being written in the frame buffer. In fact a typeface designer can create from the scratch new fonts using a program like Windows Paint. The overall design style (font and face) for a set of characters is called a typeface.

3.7.4 Polygon Filling

Polygon filling is the process of colouring the area of a polygon. Area may be defined as the total number of pixel that outlines the polygon. The polygon defined by the total number of pixels is called interior defined, and the algorithms used for filling the area are known as flood fill algorithms. The polygon defined by the bounding pixels that outline the polygon is called as boundary defined, and the corresponding algorithms are termed as boundary fill algorithms.

It is clear that the flood fill algorithms and boundary fill need some starting point inside the polygon, which is called as seed, so they are also called as seed fill algorithms. These algorithms assumes that at least one point which is interior to the polygon is known to us. Then the algorithm tries to find all the other pixels interior to the polygon and subsequently colour them. There are three types of seed fill algorithms which are as follows:

1. Flood fill algorithm
2. Boundary fill algorithm
3. Scan-line polygon fill algorithm



(a) (b)

Figure 3.13 *(a) An Illustration of Flood Fill, (b) An Illustration of Boundary Fill*

Flood Fill Algorithm

In this method we start from the given initial interior pixel, i.e. the seed. From this seed, the algorithm inspects all the surrounding pixels. The surrounding pixels can be seen by the two ways.

4-Connected Approach: In this approach the pixel to the left, right top and below from the starting position, i.e. seed is checked.

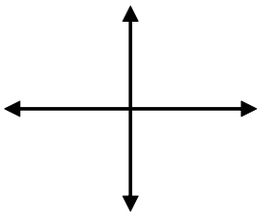


Fig. 3.14 *Four Connected Approach*

8-Connected Approach: In this approach we check the pixel to the left, right, top, below and including this we also check diagonally, as shown in Figure 3.15.

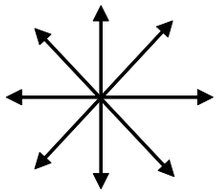


Fig. 3.15 *8-Connected Approach*

The following algorithm is shown for the 4-connected approach:

```
  Procedure Fill_ Flood (x, y, fill_colour, old_colour:
  integer)
begin
      if getpixel(x, y) = old_colour then
            setpixel(x, y, fill_colour)
            Fill_ Flood (x + 1, y, fill_colour, old_colour);
            Fill_ Flood (x - 1, y, fill_colour, old_colour);
            Fill_ Flood (x, y + 1, fill_colour, old_colour);
            Fill_ Flood (x, y - 1, fill_colour, old_colour);
```

NOTES

```
end
```

The algorithm will work for the 8-connected approach if we add the following calls also to the 4-connected approach

```
Fill_ Flood (x + 1, y - 1, fill_colour, old_colour);
Fill_ Flood (x + 1, y + 1, fill_colour, old_colour);
Fill_ Flood (x - 1, y + 1, fill_colour, old_colour);Fill_
Flood (x - 1, y - 1, fill_colour, old_colour);
```

The fill patterns for getfillsettings and setfillstyle are shown in Table 3.1.

Table 3.1 *Values for Various Fill Styles*

Names	*Value*	*Means Fill With...*
EMPTY_FILL	0	Background color
SOLID_FILL	1	Solid fill
LINE_FILL	2	---
LTSLASH_FILL	3	///
SLASH_FILL	4	///, thick lines
BKSLASH_FILL	5	\\\, thick lines
LTBKSLASH_FILL	6	\\\
HATCH_FILL	7	Light hatch
XHATCH_FILL	8	Heavy crosshatch
INTERLEAVE_FILL	9	Interleaving lines
WIDE_DOT_FILL	10	Widely spaced dots
CLOSE_DOT_FILL	11	Closely spaced dots
USER_FILL	12	User-defined fill pattern

Boundary Fill Algorithm

In this algorithm we start at a point inside the polygon and paint with a particular colour. The filling continues until a boundary colour is encountered. There are also two ways to do this:

1. 4-connected fill where we propagate: left, right, up, and down.
2. 8-connected fill where we propagate: left, right, up, down and diagonally also.

The algorithm for the 4-connected filling approach can be given as follows:

```
Procedure Fill_four (x, y, fillcolour, boundcol: integer);
Var
    currcolour: integer;
Begin
    currcolour: = inquirecolour(x, y)
    if(currcolour <> bound colour) and (curr_colour <>
fill_colour) then
     Begin
        Setpixel(x, y, fill_col)
       Fill_four (x + 1, y, fill_colour, bound_colour);
        Fill_four(x - 1, y, fill_colour, bound_colour);
       Fill_four (x, y + 1, fill_colour, bound_colour);
       Fill_four (x, y - 1, fill_colour, bound_colour);
end
```

```
end
```

The problem with four fill is shown in Figure 3.16.

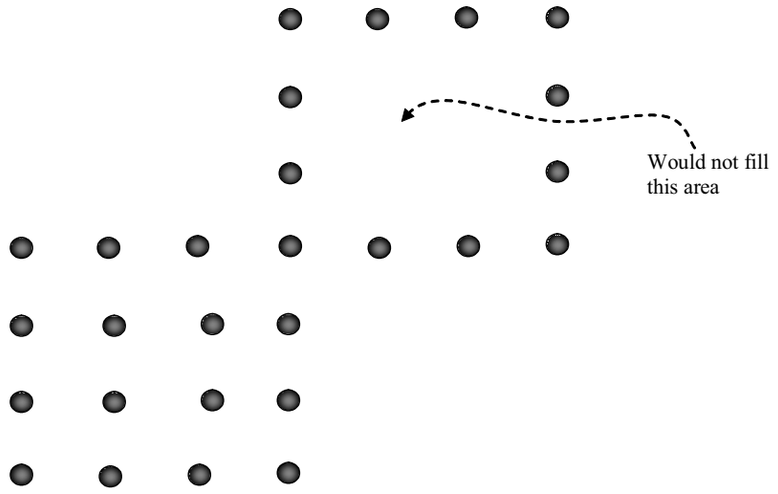


Fig. 3.16 *Boundary Filling*

This leads to the point (2). 8-connected fill algorithm where we test all eight adjacent pixels (Figure 3.17).

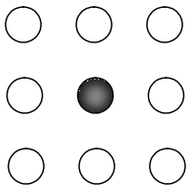


Fig. 3.17 *Eight Adjacent Pixels*

So we add the calls:

```
Eight_fill  (x  +  1,  y  -  1,  fill_colour,
bound_colour);eight_fill (x + 1, y + 1, fill_colour,
bound_colour);eight_fill (x - 1, y - 1, fill_colour,
bound_colour);eight_fill (x - 1, y + 1, fill_colour,
bound_colour);
```

Scan-Line Polygon Fill Algorithm

This algorithm tries to find the intersection point of the boundary of polygon and the scan lines [Figure 3.18(*a*) and (*b*)]. Then these pixels are used to define the pixels inside the polygon. These pixels are set to the required colour. The scan conversion algorithm locates the intersection points of the scan line with each edge of the polygon. This is done for all the scan lines starting from left to right, the intersection points are grouped and the interior pixel values points are set to the colour of the polygon.

NOTES

NOTES

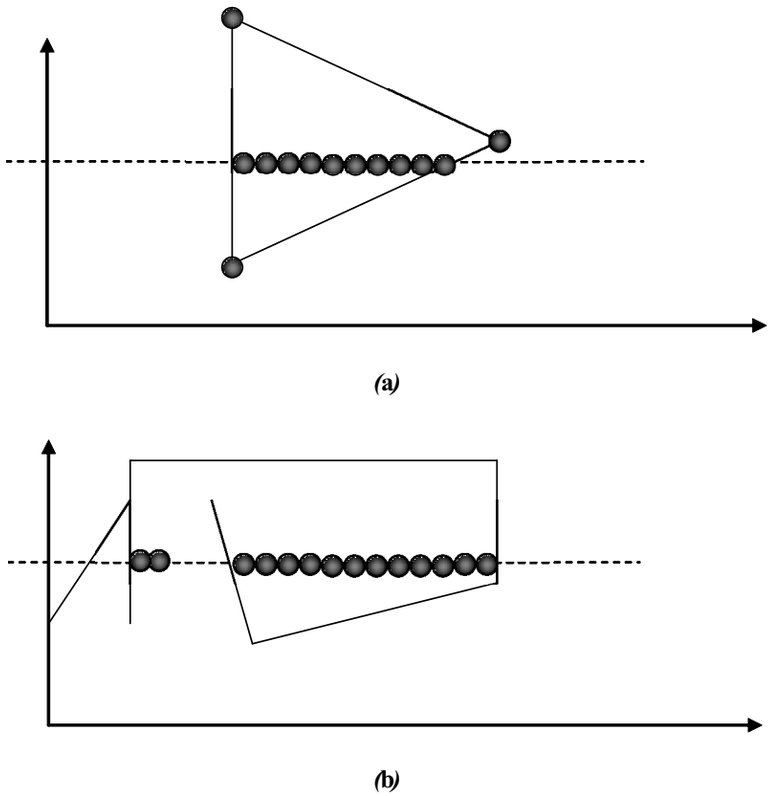


Fig. 3.18 *(a) Scan Line Convex Polygon Filling*
(b) Scan Line Concave Polygon Filling

When a scan line intersects a polygon vertex, it may require special handling.

3.7.5 Simple Ordered Edge List Algorithm

Alternate techniques for efficiently scan-converting solid area polygons know as ordered edge list algorithm. They depend upon sorting the polygon edge scan line intersections into scan line order. The efficiency of the algorithms depend on the efficiency of the sorting.

A Simple Ordered Edge List Algorithm

A simple ordered edge list algorithm:

Prepare the data:

Determine for each polygon edge the intersections with the half interval scan lines. A Bresenham or DDA algorithm can be used for this. Horizontal edges are ignored. Store each intersection $(x, y + ½)$ in a list.

Sort the list by scan line and increasing x on the scan line; i.e., (x_1, y_1) precedes (x_2, y_2) if $y_1 > y_2$ or $y_1 = y_2$ and $x_1 \leq x_2$.

Scan-convert the dat:

Extract pairs of element from the sorted list (x_1, y_1) and (x_2, y_2). The structure of the list ensures that $y = y_1 = y_2$ and $x_1 \leq x_2$. Activate pixels on the scan line y for integer values of x such that $x_1 \leq x + ½ \leq x_2$.

NOTES

3.7.6 More Efficient ordered Edge List Algorithms

The simple algorithm given in the previous section generates a large list which must be sorted. Making the sort more efficient improves the algorithm. This is accomplished by separating the vertical scan line sort in *y* from the horizontal scan line sort in *x* by using a *y* bucket sort.

A more efficient ordered edge list algorithm:

Prepare the data:

Determine for each polygon edge the intersections with the half interval scan lines, i.e., at y + ½. A Bresenham or DDA algorithm can be used for this. Ignore horizintal edges. Place the *x* coordinate of the intersection in the bucket corresponding to *y*.

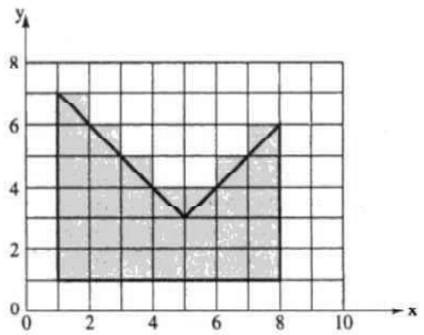


Fig. 3.19 *Results of solid area scan conversion*

As each scan line is addressed, i.e., for each *y* bucket, sort the list of *x* intersections into increasing order; i.e., x_1 precedes x_2 if $x_1 \leq x_2$.

This algorithm first sorts into scan line order with the y bucket sort, and then into order on the scan line. Thus, scan conversion begins prior to completion of the full sorting process. Also, with this algorithm it is easier to add or delete information from the display list, because it is only necessary to add or delete information from the appropriate *y* buckets. Hence, only the individual scan lines affected by the change need be resoted. An example further illustrates the algorithm.

First,, *y* buckets for scan lines 0 to 8 are established as shown in fig. 3.20. The intersections obtained by considering each edge in turn, counterclockwise from P1, are also shown in the buckets in Fig. 3.20, unsorted in *x*. The intersections were calculated using the half scan line technique. for illustrative purposes, they are also shown sorted in fig. 3.20b. In practice, a small scan line buffer such as is shown in fig. 3.20c can be used to contain the *x*-sorted intersection values. This allows more efficient additions to, or deletions from, the intersection list. They can simply be added to the end of each *y* bucket list, because the *x* sort does not take place until an individual scan line is moved to the scan line buffer. Hence, a completely sorted *y* bucket list does not to be maintained.

Extracting pairs of intersections from the *x*-sorted list and applying the algorithm above yields the pixel activation list for each scan line.

Although this second algorithm simplifies the sorting task, it either limits the number of intersections on a given scan line or requires the allocation of large

NOTES

amounts of storage, much of which may not be used. By using a linked list, this problem is overcome at the expense of additional data structure. The precalculation of the intersection of each scan line with each polygon edge is time-consuming. It also requires the storage of considerable data. By introducing an active edge list, as previously discussed for real-time scan conversion data storage is further reduced and scan line intersections can be calculated incrementally. The resulting algorithm is:

An ordered edge list algorithm using an active edge list:

Prepare the data:

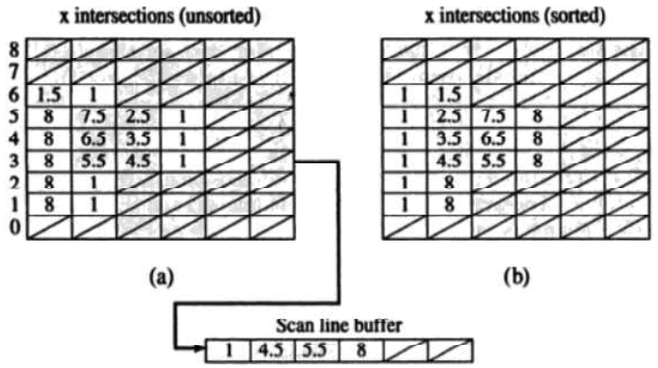


Fig. 3.20 *y buckets for the scan lines for the polygon of fig. 3.20*

For each polygon edge, determine using the half interval scan lines, i.e., $y + ½$, the highest scan lines intersected by the edge.

Place the polygon edge in the y bucket corresponding to the scan line.

Store the initial x intersection value, the number of scan lines crossed by the polygon edge, Δy, and the x increment, Δx, from scan line to scan line in a linked list.

Scan-convert the data:

For each scan line, examine the corresponding y bucket for any new edges; add any new edges to the active edge list.

Sort the x intercepts from the active edge list into increasing order, i.e., x_1 precedes x_2 if $x_1 \leq x_2$.

Extract pairs of intersections from the sorted x list. Activate pixels on the scan line y for integer values of x such that $x_1 \leq x + ½ \leq x_2$. For each edge on the active edge list, decrement Δy by 1. If $\Delta y < 0$, drop the edge from the active edge list. Calculate the new x intercept, $x_{new} = x_{old} + \Delta x$.

Advance to the next scan line.

This algorithm assumes that all data has been previously converted to a polygonal representation. Heckbert [Heck90a] presents code for a similar algorithj. Whitted [Whit81] gives a more general algorithm which removes this restriction.

3.7.7 The Egde Flag Algorithm

The edge flag algorithm is a two-step process algorithm, in first step is to outline the contour for establishes pairs of span-bounding pixels on each scan line. In

second step used to fill between these bounding pixels.

The edge flag algorithm is given

Contour olutline

Using the half scan line conveention for each edge intersecting the scan line, set the leftmost pixel whose midpoing lies to the right of the intersection, i.e., for x + ½ > $x_{intersection}$, to the boundary value.

Fill:

For each scan line intersecting the pologon

inside = false

for $x = 0$ (left) to $x = x_{max}$ (right)

if the pixel at x is set to the boundary value then negate inside

end if

if Inside = TRUE THEN

set the pixel at x to the polygon value

else

reset the pixel at x to the background value

end if

next x.

Check Your Progress

6. Define the term scan conversion process.
7. Name the basic methods to generate characters in computer screen.
8. What is polygon filling?
9. Name the three types of seed fill algorithms.
10. What is the function of the scan-line polygon fill algorithm?

3.8 SEED FILL ALGORITHM

The following methods and algorithms are used to fill the required area.

Stack Based Seed Fill Algorithms

So far as area filling is concerned we can identify two basic types of areas or regions viz. *Interior defined* and *Boundary defined*.

An interior defined region is a collection or patch of same color contiguous pixels. Pixels exterior to the region have different colors [refer Figure 3.21(a)].

For a boundary defined region pixels that mark the boundary or outline of the region have a unique color that is not same as the color of the interior pixels [refer Figure 3.21(b)].

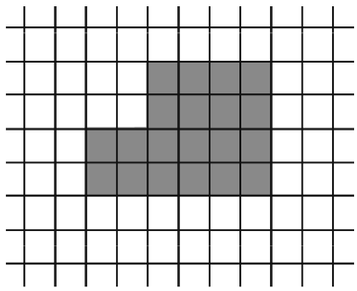


Fig. 3.21 (a) Interior-defined Region

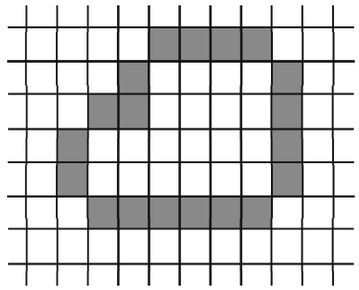


Fig. 3.21 (b) Boundary-defined Region

One approach to filling a boundary-defined area is to start at a point (pixel) inside the area and paint the area progressively towards the boundary. This method is called *Boundary fill algorithm*. This procedure accepts as input the coordinates of a sample pixel (called the seedpixel) from the area, a fill color value and a boundary color value. The seedpixel is first set to fill color while its neighbouring pixels are examined for boundary color. If boundary color is not found then these pixels are set to fill color. Similarly neighbours of these neighbouring pixels are examined and filled. Thus we continue to set the pixels in an increasing area until we encounter the boundary pixels. When we consider neighbours to the left-right, top and bottom only then this method is called *four connected method* whereas when the four diagonal pixels are also included as neighbours then this method is called *eight connected method* (refer Figure 3.22). Where the seed pixel S pixels marked 1, 2, 3, 4 are the 4-connected pixels; pixels marked 1, 2, 3, 4, 5, 6, 7, 8 are the 8-connected pixels

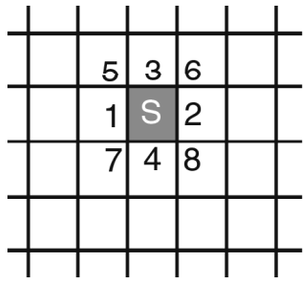


Fig. 3.22 4-and 8-Connected Pixels

3.8.1 Boundary Fill Algorithm

The following is the pseudocode of a procedure for filling up a certain boundary defined area using 8 connected fill method. The procedure boundary fill is a recursive procedure with fill color specified by parameter 'fill' and boundary color specified by parameter 'boundary'.

Input: the seed pixel (x, y), the fill colour and the 'boundary' color

Procedure boundary_fill $(x, y,$ fill, boundary)

if (Getpixel $(x, y) <>$ fill and Getpixel $(x, y) <>$ boundary) then

boundary_fill setpixel $(x, y,$ fill);

boundary_fill $(x + 1, y,$ fill, boundary);

boundary_fill ($x-1, y$, fill, boundary);

boundary_fill ($x, y+1$ fill, boundary);

boundary_fill ($x, y-1$, fill, boundary);

boundary_fill ($x+1, y+1$, fill, boundary);

boundary_fill ($x-1, y+1$, fill, boundary);

boundary_fill ($x-1, y-1$, fill, boundary);

boundary_fill ($x+1, y-1$, fill, boundary);

end if

Another 4 connected boundary fill algorithm is given below that uses stack based approach instead of a recursive approach.

Input: the seed pixel (x, y), the 'fill' color and the 'boundary' color.

Push pixel (x, y) : Initialize the stack with the seed pixel lying within the boundary-defined area and not having fill color.

while (stack not empty)

pop pixel (x, y) : Get a pixel from the stock

setpixel (x, y, fill)

if (Getpixel ($x+1, y$) < > fill and Getpixel ($x+1, y$) < > boundary) then

push pixel ($x+1, y$)

if (Getpixel ($x-1, y$) < > fill and Getpixel ($x-1, y$) < > boundary) then

push pixel ($x-1, y$)

if (Getpixel ($x, y+1$) < > fill and Getpixel ($x, y+1$) < > boundary) then

push pixel ($x, y+1$)

if (Getpixel ($x, y-1$) < > fill and Getpixel ($x, y-1$) < > boundary) then

push pixel ($x, y-1$) : The 4 neighbouring pixels are examined and pushed to stack if not already filled and not lying on the boundary endwhile

The above algorithm can be easily extended for 8 connected method by adding four more 'if' statements for four diagonal pixels at ($x+1, y+1$), ($x-1, y+1$), ($x+1, y-1$), ($x-1, y-1$).

Algorithms used for filling interior defined regions are generally known as *Flood Fill algorithms*. Such an algorithm starts with a seed pixel by replacing its existing color with fill color. Then using the 4 connected or 8 connected chain, the algorithm sets fill color to the other interior pixels. Assuming that all the pixels in an interior defined region have same color, the flood fill algorithm terminates when no more connected pixel have the old color.

3.8.2 Scan Line Seed Fill Algorithm

The seed fill algorithm makes the stack become quite large because in every loop the algorithm pushes atmost 4 to 8 pixels into the stack. The stack frequently contains duplicate pixels. A scan line seed fill algorithm minimizes stack size by

NOTES

pushing to the stack only one pixel in any uninterrupted unfilled span of pixels in a single scan line or row of pixels in a boundary defined region. Instead of proceeding along 4-connected or 8-connected chain this algorithm processes pixels in raster pattern; i.e. from left to right along each scanline in the region.

1. A seed pixel on a scan line within the area is popped from a stack containing the seed pixel.
2. The line or span containing the seed pixel is filled to the right and left of the seed pixel including the seed pixel itself until a boundary is found.
3. The extreme left and extreme right unprocessed pixel in the span are saved as x_{left} and x_{right} respectively.
4. The scan lines above and below the current scan line are examined in the range x_{left} to x_{right} for any contiguous span of either boundary pixels or previously filled pixels. If any such span is found it is simply crossed over. The extreme right pixel in all the unfilled spans on these scan lines within the same range is marked as a seed pixel and pushed onto the stack.

The pseudo code is given below (refer Figure 3.23)

Input: Pixel (x, y) as the seed pixel, fill as fill color and boundary as boundary color

Push pixel (x, y)	:	Initialize stack with seed pixel
while (stack not empty)		
Pop pixel (x, y)	:	Get the seed pixel
Setpixel $(x, y,$ fill)	:	Set it to fill color
save $x = x$	:	Save the x and y coordinates of seed pixel
save $y = y$		
$x = x + 1$		
while (Get pixel $(x, y) <>$ boundary)		
Setpixel $(x, y,$ fill)		: Fill the span to the right of the seed pixel
$x = x + 1$		
end while		
$x_{right} = x - 1$	:	Save the extreme right pixel of the span in the current scan line
$x =$ save x	:	Again come back to the seed pixel position
$x = x - 1$		
while Get pixel $(x, y) <>$ boundary)		
Setpixel $(x, y,$ fill)	:	fill the span to the left of the seed pixel
$x = x - 1$		
end while		
$x_{left} = x + 1$	:	Save the extreme left pixel of the span in the current scan line

$x = x_{left}$, y = save $y + 1$
while ($x <= x_{right}$)
if Get (pixel (x, y) = boundary or Get pixel (x, y) = fill)
: check the scan line immediately above the current scan line for filled pixel span or boundary filled span
$x = x + 1$: Simply cross over such span, if found
else
while (Get pixel (x, y) < > boundary and Get pixel (x, y) < > fill and $x <= x_{right}$)
$x = x + 1$: find the extreme right pixel of each unfilled span and push it onto the stack
end while
Push pixel ($x - 1, y$)
$x = x + 1$
end if
end while
$x = x_{left}$ y = save $y - 1$: move to the scanline immediately below the scanline containing the seed pixel.
while ($x <= x_{right}$)
check the scan line below and stack pixels accordingly
end while
end while

Note: Algorithm for checking the scan line below is same as that used for the scan line above.

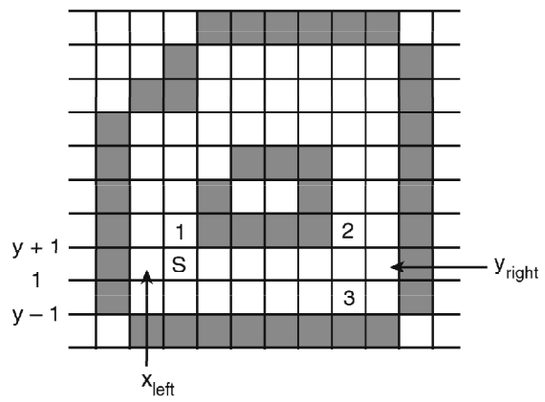


Fig. 3.23

Figure 3.23 Algorithm for checking scan line this boundary defined area with a hole inside to be filled using scan line seed fill algorithm let S be the seed pixel in scan line y; after the first cycle run of the algorithm, pixels marked 1 and 2 are stacked from scan line $y + 1$ and pixel marked 3 is stacked from scan line $y - 1$. The algorithm simply jumps the boundary of the hole in scan line $y + 1$.

NOTES

3.9 FUNDAMENTALS OF ANTIALIASING

'Alias' actually means substitution or alternative. Most common displays including our monitor, allow only two pixel states – ON or OFF. For these displays, lines may have jagged or staircase appearance because the sampling process digitizes coordinate points on an object to discrete integer pixel positions. **This process of distortion of information due to low frequency sampling or undersampling is called as aliasing.** Actually, there is a problem of discontinuity of lines and objects because our scanning is constrained to **integer values only.** This causes our lines and curves to look jagged. These gaps are called as **jaggies.**

For example, point (1, 2.6) is plotted as (1, 3) only. So, point (1, 3) is an alias location of (1, 2.6). **This process or phenomenon of shifting is called as aliasing.** Not only this, there are sudden jumps from one integer value to next. So, aliasing results in image quality problems. Aliasing is an inherent part of any discrete process and it cannot be eliminated but can be minimized. Basically, this effect occurs due to the fact that lines, polygon edges, color boundaries etc are continuous whereas raster display is discrete. Hence, to present lines, polygon edges, color boundaries etc. on the raster display devices, it must be sampled at discrete locations.

In signal analysis, in order to avoid the effects of aliasing (losing information) from such periodic objects, a signal must be sampled at a rate at least twice the highest frequency in the object and this frequency is called as **Nyquist sampling frequency or Nyquist sampling rate *f*s**–

$$f\text{s} = 2f_{\max}$$

In terms of the period of the signal, the sample interval must be less than or equal to half the period *i.e.*,

$$\text{Period} \propto \frac{1}{\text{frequency}}$$

or

$$\Delta x_s \propto \frac{1}{f_S}$$

This theorem can also be stated in terms of the sampling interval (Δx_s)

$$\Delta x_s = \Delta x_{cycle}/2$$

where Δx_s – Nyquist Sampling Rate

$$\Delta x_{cycle} = 1/f_{max}$$

There are three general manifestation of aliasing –

1. Staircase problem
2. Unequal intensity (Brightness).
3. Picket Fence Problem. Let us discuss these first.

1. Staircase problem : When any signal is sampled, this process digitizes coordinate points on an object to discrete integer pixel positions and jagged appearance can easily be seen on the resulting object.

A common example of aliasing effect is the staircase or jagged appearance. These jaggies are essentially caused by the problem of trying to map a continuous image onto a discrete grid of pixels. This **continuous-to-discrete transformation**

(known as scan conversion) is performed by sampling the continuous line, curve etc., **at discrete points (integer pixel positions) only** followed by generating image pixels at integer location that only approximates the true location of the sampled points. Pixels so generated are at alias locations of true objects and this problem is known as **staircase problem.**

2. Unequal Intensity (Brightness) : Another artifact that is less noticeable is the unequal brightness of lines of different orientation. A slanted line appears dimmer than a horizontal or vertical line, although all are presented at the same intensity level. The reason for this problem can be explained by the following example.

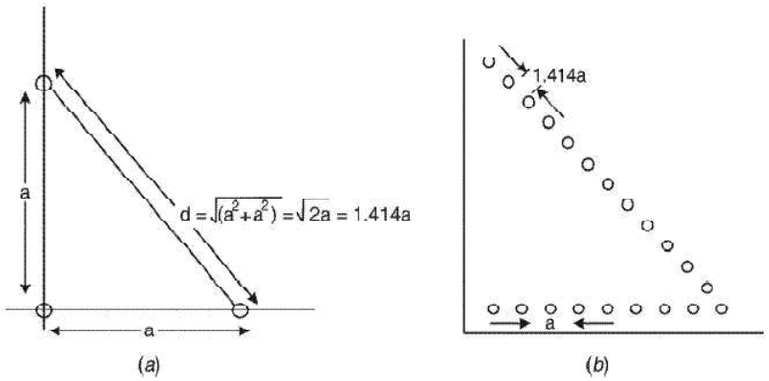


Fig. 3.24

Consider the fig. 3.24 where the pixels on the horizontal and vertical lines are placed at 'a' unit apart. Then the distance between two pixels on diagonal line will be as follows. Using Pythagoreon theorem,

$$d = \sqrt{a^2 + a^2} = \sqrt{2}\, a = 1.414\, a$$

As a result, density of pixels will be less on diagonal. Therefore, intensity of line will decrease.

3. Picket Fence Problem : When any object is displayed, sometimes the distance between two adjacent pickets is not an exact multiple of the unit distance between pixels then it does not fit into the **pixel grid** i.e., becomes unaligned with pixel grid. Such a problem is termed as **picket fence problem.**

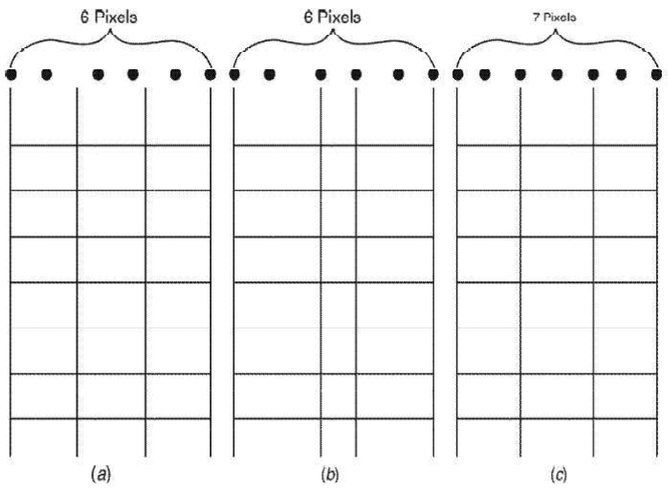


***Fig. 3.25** The Picket Fence Problem.*

NOTES

Say, an object is scan converted from an **object space** (i.e., the space where object is situated in its real dimensions) into the **image space** (i.e., the space where scaled object is stored in memory). This results in uneven distances between the pickets because the end points of the pickets will have to be aligned with pixel coordinates as shown in Fig. 3.25(b). Sometimes this is termed as **global aliasing** as the overall length of the picket fence is approximately correct. On the other hand, if we align the pixels with the picket fence at equal distance then the whole length of the fence distorts greatly. This is sometimes called **local aliasing**, as the distances between pickets are kept close to their time distances as shown in Fig. 3.25 (c).

Such problem also arises when scan conversion of any outline object is done. For example, if we scan convert an uppercase character "*P*" as shown in figure 3.26 (*a*) described by the outline into the pixel grid pattern. Some pixels are inside the region defined by the outlines and some outside. But resulting character appears as shown in figure 3.26 (b) because only insider pixels are intensified and resulting character does not look like "P" because some assymetry ocurs due to dropout of some part of image. This problem can be overcome by realigning the outlines as shown in figure 3.26 (c).

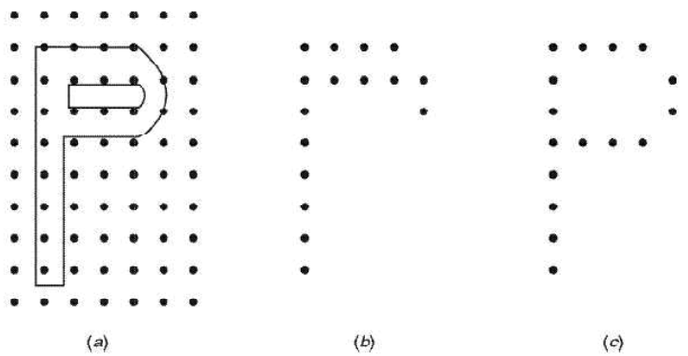


Fig. 3.26 *Some Converting an outline Font "P"*

What is antialiasing?

The process of minimization of aliasing is called as antialiasing. Antialising is one which compensates the consequences of undersampling process. The displays which allows setting pixel to gray levels between black and white, provides a means to reduce the effect of aliasing. It uses the gray levels to gradually turn off the pixels in a row as it gradually turns on the pixels in the next. Our vector generation algorithms can be modified to perform antialisaing. **Please recall that for gentle slope lines, we in effect examined the line position for each column index and decided which row was closest.** The line segment would lie between two pixels and we picked one. Suppose that instead of picking the closest, we turned them both on. We should choose the intensity values according to a function of the distance between the pixel index and the line segment so that the pixel closest to the line receives most of its intensity. The sum of the intensity values for the two pixels should match the total intensity values for the line. The function used can be a simple or a complex expression based on intensity patterns, pixel shapes and

how lines cover them. In general, we want the pixel's intensity to match the amount of the line which cover its area. Antialiasing with complicated functions can still be done efficiently by storing the function values in a table. The table is then used to look up the intensity for a distance between the pixel index and the line.

For antialisaing two approaches are possible–

1. To increase the resolution to such an extent that more pixels become available to coincide with the computed values *i.e.*, to make the steps so small and so many that the staircase begins to look more like a sloping ramp.
2. Display two or more pixels around the computed location at varying intensities or by **Dithering** (combining pixels of available colors) rather than a single pixel at the rounded off locations.

We can even shift the pixel by a fraction of the dimension, 1/4, 1/2 or 3/4, in a recently discovered technique called as **Pixel Phasing.**

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.

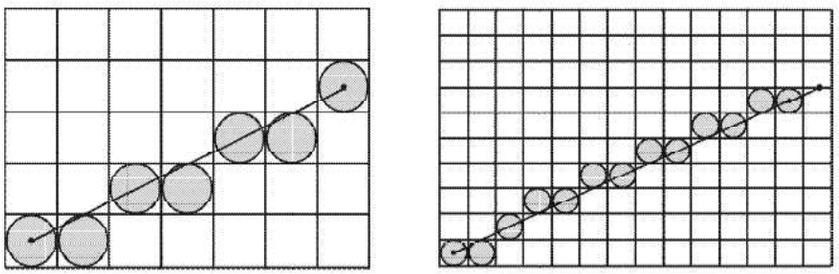


Fig. 3.27 *Effect on aliasing with increase in resolution*

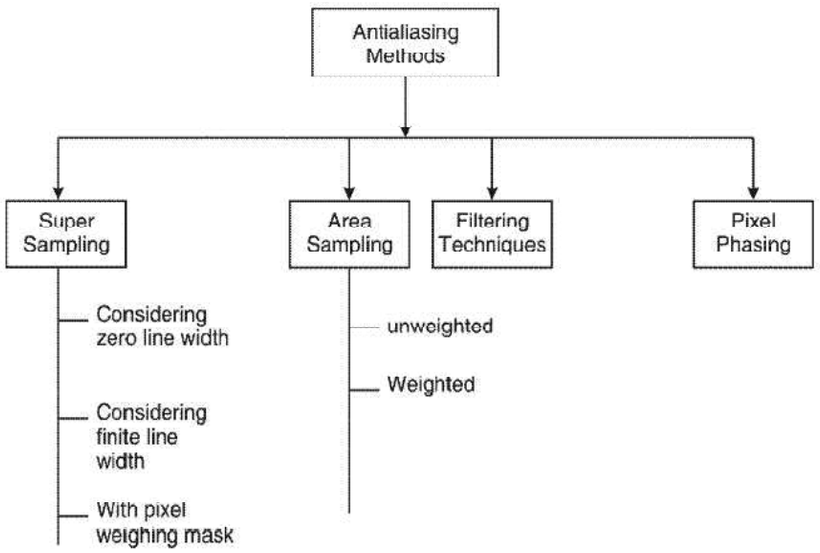


Fig. 3.28 *Types of antialising methods*

NOTES

With raster systems that are capable of displaying more than two intensity levels (color or gray scale), we can apply antialiasing methods to modify pixel intensities. By appropriately varying the intensities of pixels along the line or object boundaries, we can smooth the edges to lessen the stair-step or the jagged appearance. Anti-aliasing methods are basically classified as:

(a) Super sampling or Post filtering.

(b) Area sampling or Pre filtering.

Actually, antialiasing methods are classified as shown in Fig. 3.28.

I. Post Filtering/Super Sampling

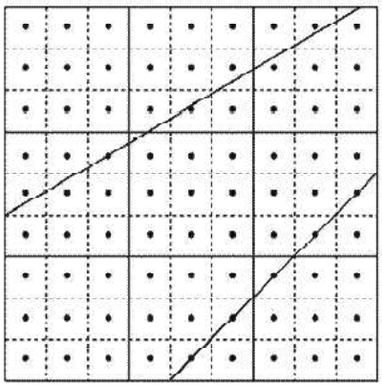
In this process sample rate is increased and this is accomplished by increasing the resolution of the raster. However, there is a limit to the ability of CRT raster scan devices to display very fine rasters. This limit suggests that **the raster be calculated at higher resolution and displayed at lower resolution, using some type of averaging to obtain the pixel attributes at the lower resolution. Actually the concept of filtering originates from the field of signal processing. This technique is called post filtering.**

There are two main post-filtering techniques discussed as under:

(i) Super Sampling

(ii) Lowpass Filtering

1. **Super Sampling.** In super sampling a pseudoraster of higher resolution than the physical raster is superimposed on the physical raster. The image is then rasterized at the higher resolution of the pseudosaster i.e. each pixel is subdivided into subpixels and the position of each subpixel is checked in relation to the object to be scan-converted. The object's contribution to a pixel's overall intensity value is proportional to the number of subpixels that are inside the area occupied by the object. Figure 3.29 (a) shows **an example** where we have a white object that is bounded by two slanted lines on a black background. We subdivided each pixel into nine (3×3) subpixels. The scene is mapped to the pixel values in figure 3.29 (b). The pixel at upper right corner is assigned 7/9 since seven of its nine sub-pixels are inside the object area. Had the object been red (1, 0, 0) and the background light yellow (0.5, 0.5, 0), the pixel would have been assigned (1 × 7/9 + 0.5 × 2/9, 0.5 × 2/9, 0) which is (8/9, 1/9, 0).



(a)

0	2/9	7/9
6/9	1	8/9
1	8/9	1/9

(b)

***Fig. 3.29** Super sampling*

This method is also known as post-filtering because filtering is carried out after sampling. **Please note that filtering means eliminating high frequencies *i.e*, combining supersamples to compute a pixel color.**

2. **Lowpass filtering.** In this method, we reassign each pixel a new value that is weighted average of its original value and the original values of its neighbours. A lowpass filter in the form of a $(2^n+1) * (2^n+1)$ grid where n ^{3}1, holds the weights for the computation. All weight values in a filter should sum to one.

   Assume a filter of (3×3) as shown below.

Here, we align the filter with the pixel grid and center it at the pixel in order to compute a new value for a pixel. The weighted average is simply the sum of products of each weight in the filter times the corresponding pixel's original value. The filter shown in figure 3.30 (a) means that half of each pixel's original value is retained in its new value, while each of the pixel's four immediate neighbours contributes one eighth of its original value to the pixel's new value. The result of applying this filter to the pixel values in figure 3.30. (*a*).

0	1/8	0
1/8	1/2	1/8
0	1/8	1/9

(*a*)

0	0	0	0	0	1/8	0
0	0	0	1/8	1/4	1/2	1/8
0	1/8	1/4	5/8	5/8	1/4	0
1/8	5/8	5/8	1/4	1/8	0	0
0	1/8	1/8	0	0	0	0

(*b*)

Fig. 3.30

A low pass filter with equal weights, sometimes referred to as a box filter, is said to be doing neighbourhood averaging. On the other hand, a filter with its weight values conforming to a two dimensional Gaussian distribution is called a Gaussian filter.

II. Pre-Filtering/Area sampling

In this method, a pixel is treated as a finite area rather than as a point and this technique, basically, works on the true signal in the continuous space to derive proper values for individual pixels. There is one most popular pre-filtering technique, is termed as area sampling, discussed below.

1. **Area Sampling.** In this approach, we superimpose a pixel grid pattern onto the continuous object definition. For each pixel area that intersects the object, we calculate the percentage of overlap by the object. This percentage determines the proportion of the overall intensity value of the corresponding pixel that is due to the object's contribution. **In other words, the higher the percentage of overlap, the greater influence the object has on the pixel's overall intensity value.**

Here, in figure 3.31 (a) a mathematical line shown in dotted form is represented by a rectangular region that is one pixel wide. The percentage of overlap between rectangle and each intersecting pixel is calculated analytically. Assuming that the background is black and the line is white, the percentage values can be used directly to set the intensity of the pixel as shown in figure 3.31 (b). On the otherhand, had the background been gray (0.5, 0.5, 0.5) and the green (0, 1, 0), each blank pixel in the grid would have had the background gray value and each pixel filled with a fractional number *f* would have been assigned a value of [0.5 (1–*f*) + 0.5 (1–*f*)+ *f*, 0.5 (1– *f*)] a proportional blending of the background and object color.

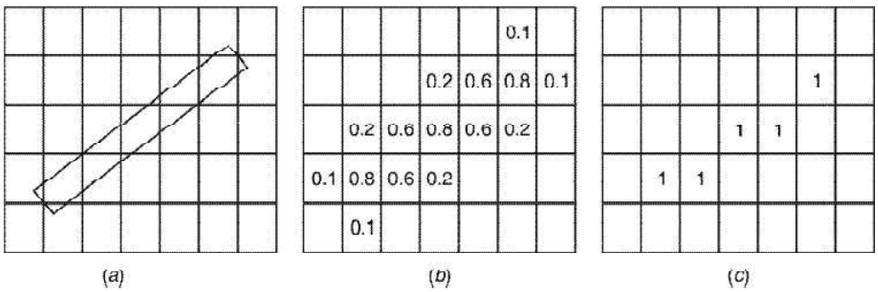


Fig. 3.31

Although the resultant discrete approximation of the line in figure 3.30 (b) takes on a blurry appearance, it no longer exhibits the sudden transition from an on pixel to an off pixel and vice- versa, which is what we would get an ordinary scan-conversion method as shown in figure 3.30 (c). This trade-off is characteristic of an anti-aliasing techniques based on filtering.

2. Another technique for antialiasing is ***pixel phasing***. This is a hardware based antialiasing technique. The graphics system in this case is capable of shifting individual pixels from their normal positions in the pixel grid by a fraction (typically 0.25 and 0.5) of the unit distance between pixels. By moving pixels closer to the true line or other contour, this technique is very effective in smoothing out the stair steps without reducing the sharpness of the edges.

Before we discuss further, please remember these points first for supersampling technique–

1. In super sampling, the intensity value of pixel is the average of the intensity values of all the sampled subpixels within the pixel.

2. Each pixel is subdivided into a (3×3) area. The **maximum number** of subpixel units covered by the line within each pixel will be 3 and the **minimum** will be 0 (zero). This allows upto 4 different levels of gradation in intensity to be defined rather than just 2.

3. Color of pixel $(x, y) = (x, y) = \dfrac{\sum \text{Intensities of sub pixels within pixel}(x, y)}{\sum \text{weights of sub pixels}}$

4. This type of filtering is also called as **unknown filtering** because each Super sample in a pixel has given an equal weight irrespective of the position. So, an unweighted filter gives an unweighted average.

5. **Non-uniform weighting** can be used when the values are combined, to give more weight to subpixel samples that are closest to the centre of the pixel. Herein, the color of pixel (x, y) is given by the formula–

$$\text{Color of pixel } (x, y) = \frac{\sum_{i=1}^{k} w_i \times I_i}{\sum_{i=1}^{k} w_i} \quad \text{[within pixel } (x, y)\text{]}$$

That is, each supersample is multiplied by its corresponding weight and products are summed to produce a weighted average, which is used as a pixel color. **Please note that weights to each super sample are given on the basis of its distance from the centre of the pixel. Also note that the centre sample within a pixel has a maximum weight.** Such an array of weights of subpixels is called as a pixel-weighted mask.

Let us now discuss some **points to remember** regarding **Area sampling or prefiltering** technique as follows–

1. This method treats pixel as an area.
2. For each pixel area that intersects the object, we compute the percentage of overlap by the object.
3. Area sampling works by defining the intensity of each pixel to be proportional to the area of the pixel.
4. The resultant intensity of pixel (x, y) is given by–

$$= \frac{\sum_{i=1}^{k} w_i \text{ of subpixels overlaping with enclosed area}}{\sum_{i=1}^{n} w_i} * \text{(maximum intensity of pixel)}$$

Note : Greater weight is given to area near the centre of the pixel.

This is shown in Fig. 3.31.

5. **Unweighted Area Sampling** is done as follows–

We have seen that for sloped lines, many a times the line passes between two pixels. In these cases, line drawing algorithm selects the pixel which is closer to the true line. This step in line drawing algorithms can be modified to perform antialiasing. instead of picking closest pixel, both pixels are highlighted. However their intensity values may differ.

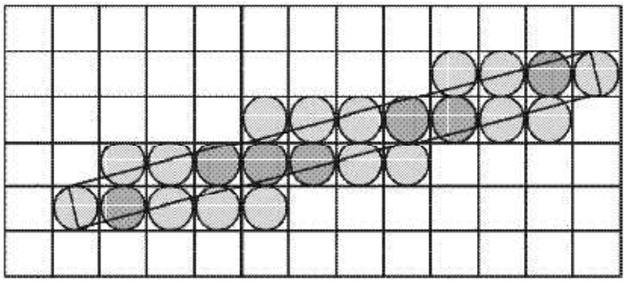


Fig. 3.32 *Unweighted area sampling*

In this method, the intensity of pixel is proportional to the amount of line area occupied by the pixel. This technique produces better results than does setting pixels either to full intensity or to zero intensity.

6. **Weighted Area Sampling** is done as follows–

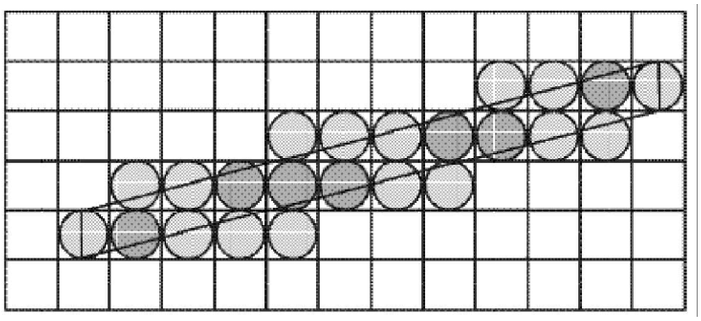


Fig. 3.33 *Weighted area sampling*

We have seen that in unweighted area sampling equal areas contribute equal intensity, regardless of the distance between the pixel's center and the area, only the total number of occupied area matters. Thus, a small area in the corner of the pixel contributes just as much as does an equal-sized area near the pixel's center. To avoid this problem even better strategy is used in the weighted area sampling.

In weighted area sampling equal areas contribute in weighted *i.e.*, a small area closer to the pixel center has greater intensity than does one at a greater distance. Thus, in weighted area sampling the intensity of the pixel is dependent on the line area occupied and the distance of area from the pixel's center. This is illustrated in Figure 3.32.

3.9.1 The Convolution Integral and Antialiasing

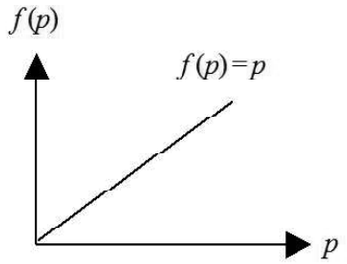
Image processing is actually performed by computing a new value for each pixel in an image. One simple way is to compute a pixels value (p') solely on the basis of its old value (p), without regard to any other pixel. Mathematically it can be represented by a transformation or mapping function $f(\ )$, such that,

$$p' = f(p)$$

One example of $f(p)$ is $W - p$, where W represents white color value; $f(p)$ being applied on an image produces an inverted (negative) image. Such individual pixel-based processing of image is known as ***pixel point processing***. By suit-

ably choosing the mapping function, color correction and alteration, brightness and contrast adjustments can be done on an image. Brightness adjusts the value of each pixel up or down uniformly – increasing brightness makes every pixel lighter while decreasing brightness makes every pixel darker. On the other hand enhancing contrast makes the lighter pixels even lighter while at the same time making the darker pixels even darker.

In terms of brightness (B) and contrast (C), p' and p are related linearly as, $p' = f(p) = C\,p + B$, which is analogous to $y = m\,x + c$; C is the gradient and B is the y-axis intercept of the p'-p line, p being plotted along x-axis and $p' = f(p)$ along y axis.



Another class of image processing, known as ***pixel group processing***, works by computing each pixel's new value as a function of not just it's old value, but also of the values of the neighbouring pixels. ***Convolution*** is one such operation, which combines a pixel's value with those of its neighbours by multiplying together two arrays of numbers, generally of different sizes, but of the same dimensionality. One of the input arrays is normally just a gray level image. The second array is a two-dimensional array of weights and is known as the ***convolution mask***. The convolution is performed by sliding the mask over the image, generally starting at the top left corner, so as to move the mask through all the positions where it fits entirely within the boundaries of the image. Each mask position corresponds to an array (same size as the mask) of pixel values known as ***convolution kernel*** and the resultant convolution output is a single pixel.

If a pixel has coordinates (x, y), so that it has neighbours at $(x-y, y+1)$, $(x, y+1)$,…. $(x+1, y-1)$ and we apply a convolution mask of the form:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

the value p' computed for the new pixel at (x, y) is,

$$p' = a\,p_{x-1,y+1} + b\,p_{x,y+1} + c\,p_{x+1,y+1} + d\,p_{x-1,y} + e\,p_{x,y} + f\,p_{x+1,y} + g\,p_{x-1,y-1} + h\,p_{x,y-1} + i\,p_{x+1,y-1}$$

where $p_{x,y}$ is the old value of the pixel at (x, y) and similarly the other 'p's.

If the image (I) has M rows and N columns, and the mask has m rows and n columns, then the size of the output image (O) will have $M-m+1$ rows, and $N-n+1$ columns.

NOTES

Mathematically we can generalize 2D convolution as:

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1)\, K(k,l)$$

where i runs from 1 to $M-m+1$ and j runs from 1 to $N-n+1$.

The transformation carried out by convolution in effect remove or attenuate certain spatial frequency in an image and hence is termed as *filtering* operation.

A classical ***blur* filter** (or in other terms ***low pass* filter**) for example is:

$$\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

Such filter smoothes out the sharpness at the edges (characterized by abrupt transition from high pixel value to relatively smaller values) by reducing the difference of the adjacent pixel values. In fact blur filter also known as ***mean filter*** replaces each pixel value in an image with the mean ('average') value of its neighbours, including itself.

On the contrary a ***sharpen* filter** (or ***high pass* filter**) is:

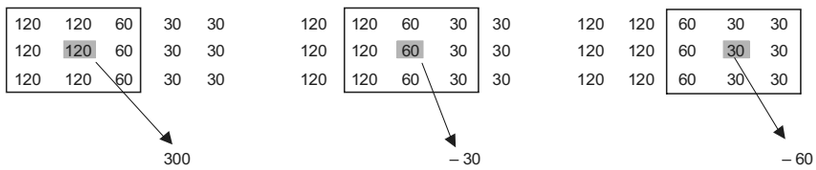
$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

This mask filters out the low frequency components. By subtracting values of the adjacent pixels (by multiplying -1), while magnifying the central value (by multiplying 9), it eliminates any value that is common to the central pixel and its surroundings.

If we apply the above mask or filter to a convolution kernel where there is a gradual discontinuity, such as

$$\begin{matrix} 120 & 60 & 30 \\ 120 & 60 & 30 \\ 120 & 60 & 30 \end{matrix}$$

assuming that all the pixels in the surrounding and to the left have the value 120 and those to the right 30, the new values computed on the central row will be 300, –30 and –60. The 3 x 3 masking is shown below.



Since negative pixel values are not allowed –30 and –60 pixels values will be set to 0. So, in effect the gradual transition $120 \rightarrow 60 \rightarrow 30$ will be replaced by a hard (sharp) line, while regions of constant value on either side will be left unchanged.

It should be noted that these blur and sharpen filter have no effect over regions where the pixels all have the same value.

Another common type of filter called *unsharp mask* is used for producing an edge image. Following are some examples of 3 × 3 unsharp mask that produces scaling followed by addition/subtraction.

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}$$

3.9.2 Half Tonning

Traditionally, ***Halftoning*** is the process of printing continuous tone images (with multiple levels of gray or color) in newspapers, magazines and books using a bi-level printer. In case of black & white printing one cannot directly print lots of shades of gray using only pure black ink on a printing press. What it does actually is, it creates a dot pattern by exposing a negative through a fine screen (a halftone screen) onto a photosensitive printing plate. The size of the dots varies depending on the fineness of the screen and length of exposure. In areas where the image was dark, the plate would print large dots of ink – so large and dense that the dots may even overlap. In areas where the image was light, only the smallest dispersed dots of ink are printed. The dots fuse together, both on the paper and in our eye, to give a convincing illusion of continuous tones. Such printed images made up of a series of varying size & varying density dots in a specific pattern to simulate the look of continuous tones are commonly known as ***halftones***. Similarly *color halftones* are actually made up of a series of dots in cyan, magenta, yellow, and black (**CMYK**) that fools the eye into seeing the millions of colors that make up the original image. In this case there are four separate halftone screens, which are then superimposed on each other in each of the four colors.

Fig. 3.34 *This is the original grayscale image*

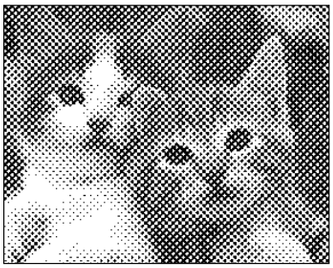


Fig. 3.35 *This is the B & W halftone of the original image*

See figures 3.34 and 3.35. This is how we view pictures in newspapers. As you hold your book farther from your eyes this image becomes clearer. Hold close and the individual halftone dots are visible. The shade of black is actually the same for the large and small dots but the smaller dots look like they are a lighter shade of gray than the larger ones. Note too that the dots are laid out in a diagonal pattern (at 45 degrees) rather than in horizontal rows and vertical columns. This is supposed to make the pattern a bit less noticeable to the human eye.

In computer graphics, ***Digital halftoning*** is adopted to simulate the traditional halftoning process. In this process an image is decomposed into a grid

NOTES

of halftone cells. Each such cell treated as a megapixel is a small two-dimensional array of actual pixels. The number and pattern of pixels turned on or off in them determine the gray value of the resulting picture element. The more number of off pixels in a halftone cell, the darker the cell appears.

3.10 TWO-DIMENSIONAL CLIPPING

The purpose of clipping procedure is to determine which part of a scene or specifically which points, lines (or curves) or portions of the lines (or curves) of a scene lie inside the clipping window. For viewing transformation, only these elements are retained for display and everything outside the window is discarded.

The algorithm for clipping is different for different shape of clipping window and for different type of objects (line, curve, etc.) Here we will discuss some of the standard clipping algorithms for rectangular clipping window.

3.10.1 Line Clipping

Figure 3.36 shows a 2-D scene and a rectangular clipping window. It is defined by left (L), right (R), top (T) and bottom (B) edges parallel to the edges or axes of the display surface.

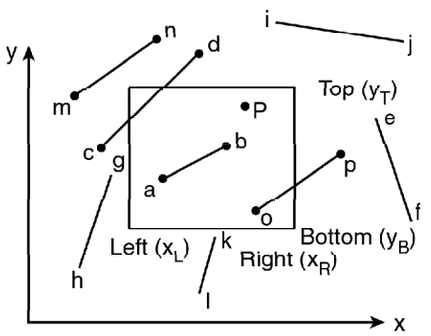


Fig. 3.36 *Two Dimensional Clipping Window*

Now any point (x, y) is interior to the clipping window provided that,

$$x_L \le x \le x_R \text{ and } y_B \le y \le y_T$$

where x_L, x_R, y_T, y_B are the x and y coordinates respectively of the left, right, top and bottom edges of the window.

The equal sign indicates that points on the window boundary are included within the window.

To determine visibility of lines with respect to the clipping window, we have to deal with three types of line.

(i) Lines totally visible

(ii) Lines totally invisible

(iii) Lines partially visible

Lines are interior to the clipping window and hence totally visible if both end points are interior to the window, e.g., line *ab* in Figure 3.36. However lines are not totally invisible if both end points are exterior to the window, e.g., line *cd*

in Figure 3.36. If both end points of a line are to the right (e.g., line *ef*) or to the left (e.g., line *gh*) or above (e.g., line *ij*) or below (e.g., line *kl*) the window the line is completely exterior to the window and hence totally invisible. Lines which fail these two tests are either partially visible (e.g., lines *cd* and *op*) or totally invisible (e.g., line *mn*).

Explicit Line Clipping Algorithm

This technique uses a 4-digit code to indicate which of the nine regions around a clipping window contain the end point of a line. The scheme which assigns this 4 bit code to the line end points is summarized as follows:

We consider x_L and x_R to be the x coordinates of the left and right edges of the window and y_T and y_B to be the y coordinates of the top and bottom edges of the window respectively. x,y are the coordinates of any end point of the test line. Here the rightmost bit is bit 1.

(i) if $x < x_L$ then set bit 1 to 1 else 0

(ii) if $x > x_R$ then set bit 2 to 1 else 0

(iii) if $y < y_B$ then set bit 3 to 1 else 0

(iv) if $y > y_T$ then set bit 4 to 1 else 0

Figure 3.37 shows the nine region codes with respect to a clipping window. You may note that the codes for the diagonal regions of the window are obtained by adding the codes for the two adjacent regions.

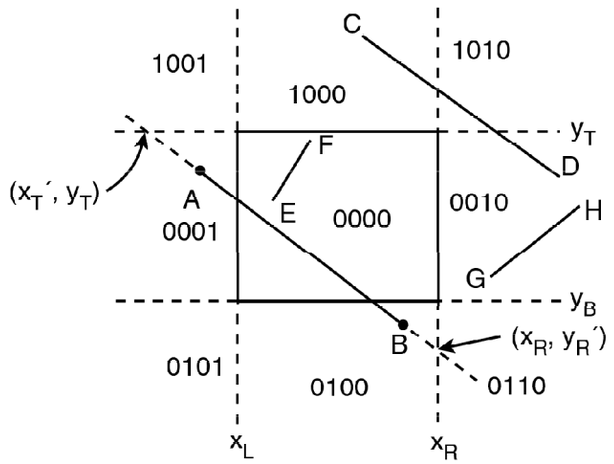


Fig. 3.37 *Nine Region Codes Relative to a Clipping Window*

Now the algorithm is as follows,

Input : $x_L, x_R, y_T, y_B, P_1(x_1, y_1), P_2(x_2, y_2)$: P_1, P_2 are the endpoints of the test line

Initialise $i = 1$

while $i <= 2$

if $x_i < x_L$ then bit 1 of code $- P_i = 1$ else 0
if $x_i > x_R$ then bit 2 of code $- P_i = 1$ else 0 : The endpoint codes of the line are

NOTES

set

if $y_i < y_B$ then bit 3 of code $-P_i = 1$ else 0

if $y_i > y_T$ then bit 4 of code $-P_i = 1$ else 0

$i = i + 1$

end while

if code $-P_1$ = code $-P_2 = 0$ then (the line is totally visible) draw $P_1 P_2$ (refer line *EF* in Figure 3.37).

if code $-P_1$ AND code $-P_2 <> 0$ then (the line is totally invisible*) ignore $P_1 P_2$ (refer line *GH* in Figure 3.36).

If both the end points codes are not equal to zero and their logical intersection is also not non-zero (i.e., zero) then the line is checked for partial visibility or total visibility (refer line *AB* and *CD* in Figure 3.37). The intersection of the line $P_1 P_2$ with window edges are found

For a line having end points at (x_1, y_1) and (x_2, y_2)

$$y_2 = m\ (x_2 - x_1) + y_1$$

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

The intersection with the window edges are given by:

Left : $x_L,\ y'_L = m(x_L - x_1) + y_1; m \neq \alpha$ ...(3.5)

Right : $x_R,\ y'_R = m(x_R - x_1) + y_1;\quad m \neq \alpha$ ...(3.6)

Top : $y_T,\ x'_T = x_1 + \left(\frac{1}{m}\right)(y_T - y_1);\quad m \neq 0$ ...(3.7)

Bottom : $y_B,\ x'_B = x_1 + \left(\frac{1}{m}\right)\ (y_B - y_1);\quad m \neq 0$ ...(3.8)

when $m = \alpha$ i.e. the line is vertical nd $x_2 - x_1 = 0$

assign very large numbers to x_L, x_R, y'_L, y'_R

when $m = 0$, i.e., the line is horizontal and $y_2 - y_1 = 0$

assign very large numbers to $x'_T,\ x'_B,\ y_T,\ y_B$

To get the two intersection points $(x'_1,\ y'_1)$ and $(x'_2,\ y'_2)$

Set $i = 1$

if $i <= 2$ then

if $y'_L < y_T$ and $y'_L > y_B$ then

$x'_i = x_L,\ y'_i = y'_L$

$i = i + 1$

end if

end if

if $i <= 2$ then
  if $y'_R < y_T$ and $y'_R > y_B$ then
  $x'_i = x_R$, $y'_i = y'_R$
  $i = i + 1$
  end if

end if
if $i <= 2$ then
  if $x'_B < x_R$ and $x'_B > x_T$ then
  $x'_i = x'_B$, $y'_i = y_B$
  $i = i + 1$
  end if
end if

if $i <= 2$ then
  if $x'_T < x_R$ and $x'_T > x_L$ then
  $x'_i = x'_T$, $y'_i = y_T$
  $i = i + 1$
  end if
end if

For partly visible lines intersecting two window edges in the visible region, any two of these 'if's work here and we get two (x'_i, y'_i) which are the end points of the visible portion of the line.

If $i < 2$ then (none of the intersections are found on the visible window edges) ignore P_1P_2 (totally invisible) (refer line *CD* in Figure 3.37).

else draw the line between (x'_1, y'_1) and (x'_2, y'_2) where codes of both the end points are non zero and their logical ANDing is zero.

end if

Now we consider the case where one of the end points of the line lies inside the window and the other outside it.

If code – $P_1 = 0$ then (P_1 is inside the window) $P = P_1$ and $P_{out} = P_2$.

else if code – $P_2 = 0$ (P_2 is inside the window) then $P = P_2$ and $P_{out} = P_1$ [refer Figure 3.38 (a)]

end if

NOTES

if $(P_{out}) \cdot x < x_L$ then

if $y'_L <= y_T$ and $y'_L >= y_B$ then $P_i = (x_L, \quad y'_L)$

else if $y'_L > y_T$ then $P_i = (x'_T, y_T)$

else (i.e., if $y'_L < y_B$) then $P_i = (x'_B, \quad y_B)$ [refer Figure 3.38(d)]

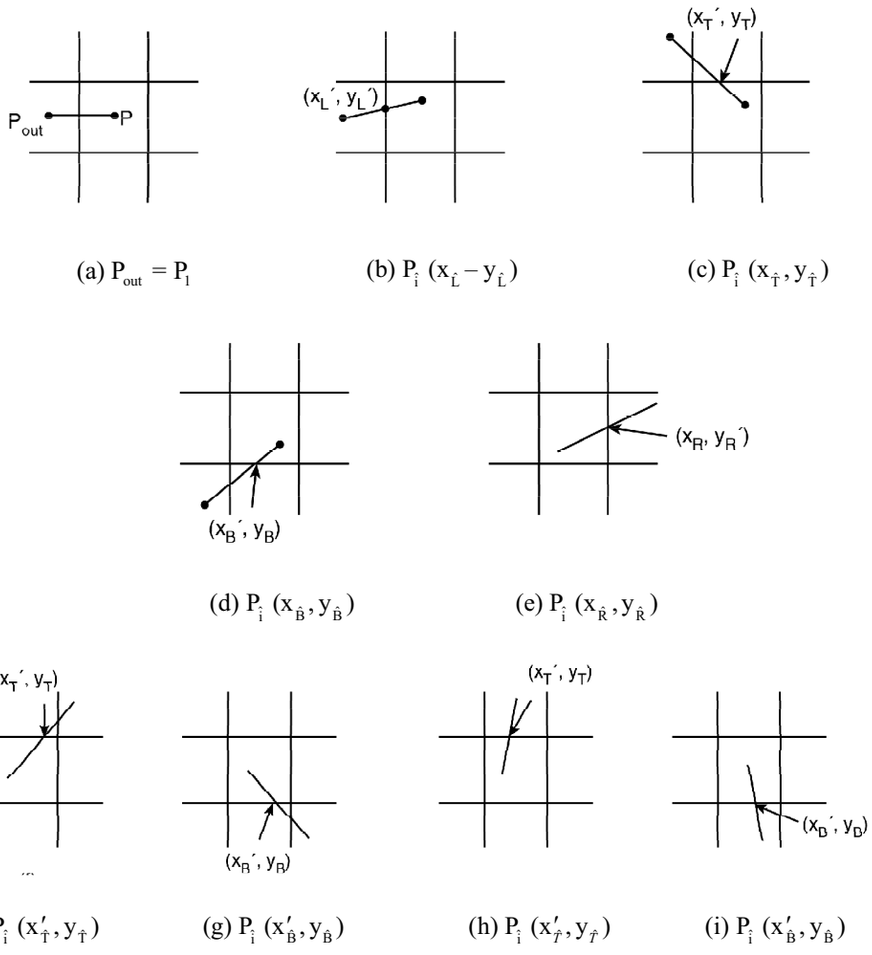
end if

else if $(P_{out}) \cdot x > x_R$ then

if $y'_R <= y_T$ and $y'_R >= y_B$ then $P_i = (x_R, \quad y'_R)$ [refer Figure 3.38 (e)]

else if $y'_R > y_T$ then $P_i = (x'_T, \quad y_T)$ [refer Figure 3.38(f)]

end if



(a) $P_{out} = P_1$ (b) $P_{\hat{i}}\ (x_{\hat{L}} - y_{\hat{L}})$ (c) $P_{\hat{i}}\ (x_{\hat{T}}, y_{\hat{T}})$

(d) $P_{\hat{i}}\ (x_{\hat{B}}, y_{\hat{B}})$ (e) $P_{\hat{i}}\ (x_{\hat{R}}, y_{\hat{R}})$

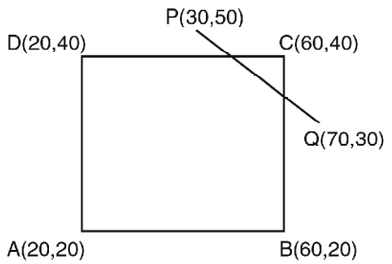
(f) $P_{\hat{i}}\ (x'_{\hat{T}}, y_{\hat{T}})$ (g) $P_{\hat{i}}\ (x'_{\hat{B}}, y_{\hat{B}})$ (h) $P_{\hat{i}}\ (x'_{\hat{T}}, y_{\hat{T}})$ (i) $P_{\hat{i}}\ (x'_{\hat{B}}, y_{\hat{B}})$

Fig. 3.38 *One End Point Lies Inside the Window and One Outside*

else (i.e., if $y'_R < y_B$) then $P_i = (x'_B, \quad y_B)$ [refer Figure 3.54(g)]
end if
else if $(P_{out}) \cdot y > y_T$ then $P_i = (x'_T, \quad y_T)$ [refer Figure 3.54(h)]
else (i.e.,if $(P_{out}) \cdot y < y_B$) then $P_i = (x'_B, \quad y_B)$ [refer Figure 3.54(i)]
end if

Draw visible portion PP_i

Example 3.11: Given a window A (20, 20), B (60, 20), C (60, 40), D (20, 40) use any clipping algorithm to find the visible portion of the line P (30, 50) to Q (70, 30) inside the window.



Solution: For the line PQ the slope is,

$$m = \left(\frac{y_2 - y_1}{x_2 - x_1}\right) = \left(\frac{50-30}{30-70}\right) = -\left(\frac{20}{40}\right) = -\frac{1}{2}$$

and the intersections with the window edges are

left : $x = 20$, $y = m\,(x_L - x_1) + y_1$

or, $y = -1/2\,(20 - 30) + 50 = -\,1/2\,(-10) + 50 = 5 + 50 = 55$

which is greater than y_T and is rejected.

right : $x = 60$, $y = m\,(x_R - x_1) + y_1$

or, $y = -\,1/2\,(60 - 30) + 50 = -15 + 50 = 35$

$\therefore$ The intersection with the right edge is at point (60, 35).

top : $y = 40$, $x = x_i + 1/m\,(y_T - y_1)$

or, $x = 30 - 2\,(40 - 50) = 30 + 20 = 50$

$\therefore$ The intersection with the top edge is at point (50, 40)

bottom : $y = 20$, $x = x_1 + 1/m\,(y_B - y_1)$

or, $x = 30 - 2\,(20 - 50)$

$= 30 + 60 = 90$

which is greater than x_R and thus rejected.

So the visible part of the line PQ is from P (50,40) to Q (60,35).

3.10.2 Sutherland–Cohen Algorithm

This is one of the most popular line clipping algorithm. The concept of assigning 4-bit region codes to the endpoints of a line and subsequent checking and AND

operation of the endpoint codes to determine totally visible lines and totally invisible lines (lying completely at one side of the clip window externally) was originally introduced by Dan Cohen and Ivan Sutherland in this algorithm. For clipping other totally invisible lines and partially visible lines, the algorithm breaks the line segments into smaller subsegments by finding intersection with appropriate window edges. For a pair of a non-zero endpoint and an intersection point, the corresponding subsegment is checked for two primary visibility state as done in the earlier steps. The process is repeated till two visible intersections are found or no intersection with any of the four visible window edge is found. Thus this algorithm cleverly reduces the number of intersection calculations unlike the previous algorithm. The steps of the algorithm are as follows (refer Figure 3.39).

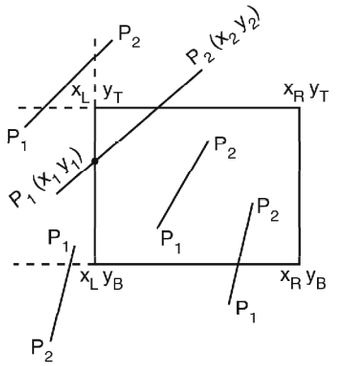


Fig. 3.39 *Sutherland– Cohen algorithm*

1. Input : $x_L, x_R, y_T, y_B, P_1(x_1, y_1), P_2(x_2, y_2)$

   Initialise $i = 1$

   while $i <= 2$

   if $x_i < x_L$ then bit 1 of code $- P_i = 1$ else 0
   if $x_i > x_R$ then bit 2 of code $- P_i = 1$ else 0
   if $y_i < y_B$ then bit 3 of code $- P_i = 1$ else 0
   if $y_i > y_T$ then bit 4 of code $- P_i = 1$ else 0
   are set : The endpoint codes of the line

   $i = i + 1$

   end while

   $i = 1$

2. Initialise $j = 1$

   while $j <= 2$

if $x_j < x_L$ then $C_{j\ \text{left}} = 1$ else $C_{j\ \text{left}} = 0$
if $x_j > x_R$ then $C_{j\ \text{right}} = 1$ else $C_{j\ \text{right}} = 0$
if $y_j < y_B$ then $C_{j\ \text{bottom}} = 1$ else $C_{j\ \text{bottom}} = 0$
if $y_j > y_T$ then $C_{j\ \text{top}} = 1$ else $C_{j\ \text{top}} = 1$: Set flags according to the position of the line endpoints with respect to window edges

   end while

NOTES

3. if codes of P_1 and P_2 are both equal to zero then draw P_1P_2 (totally visible)
4. if logical intersection or AND operation of code – P_1 and code – P_2 is not equal to zero then ignore P_1P_2 (totally invisible)
7. if code – P_1 = 0 then swap P_1 and P_2 along with their flags and set $i = 1$
6. if code – $P_1 <> 0$ then

   for $i = 1$,

   {if $C_{1\ \text{left}} = 1$ then

   find intersection (x_L, y'_L) with left edge vide Equation (3.5)

   assign code to (x_L, y'_L)

   $P_1 = (x_L,\ y'_L)$

   end if

   $i = i + 1$;

   go to 3

   }

   for $i = 2$,

   {if $C_{1\ \text{right}} = 1$ then

   find intersection (x_R, y'_R) with right edge vide Equation (3.6)

   assign code to (x_R, y'_R)

   $P_1 = (x_R, y'_R)$

   end if

   $i = i + 1$

   go to 3

   }

   for $i = 3$

   {if $C_{1\ \text{bottom}} = 1$ then

   find intersection (x'_B, y_B) with bottom edge vide Equation (3.7)

   assign code to (x'_B, y_B)

   $P_1 = (x'_B, y_B)$

   end if

   $i = i + 1$

   go to 3

   }

   for $i = 4$

   {if $C_{1\ \text{top}} = 1$ then

   find intersection (x'_T, y_T) vide Equation (3.8) with top edge

   assign code to (x'_T, y_T)

NOTES

$P_1 = (x'_T, y_T)$

end if

$i = i + 1$

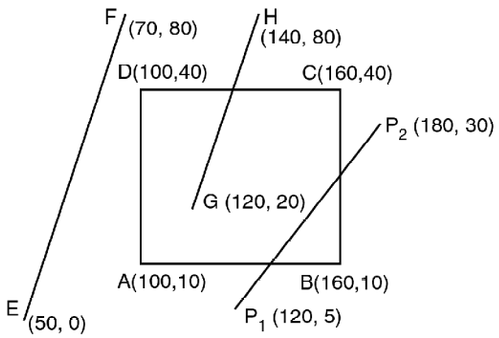
go to 3

}

end

Example 3.4: A Clipping window ABCD is located as follows:

A (100, 10)*, B* (160, 10), *C* (160, 40)*, D* (100, 40). *Using Sutherland-Cohen clipping algorithm find the visible portion of the line segments EF, GH and* $P_1 P_2$*. E* (50,0), *F* (70, 80), *G* (120, 20), *H* (140, 80), P_1 (120, 5)*,* P_2 (180, 30)



Solution: At first considering the line $P_1 P_2$

INPUT: $P_1(120, 5)$, $P_2(180, 30)$

$x_L = 100$, $x_R = 160$, $y_B = 10$, $y_T = 40$

$x_1 > x_L$ then bit 1 of code–$P_1 = 0$ $C_{1\ \text{left}} = 0$

$x_1 < x_R$ then bit 2 of code–$P_1 = 0$ $C_{1\ \text{right}} = 0$

$y_1 < y_B$ then bit 3 of code–$P_1 = 1$ $C_{1\ \text{bottom}} = 1$

$y_1 < y_T$ then bit 4 of code–$P_1 = 0$ $C_{1\ \text{top}} = 0$

code–$P_1 = 0100$,

$x_2 > x_L$ then bit 1 of code–$P_1 = 0$ $C_{2\ \text{left}} = 0$

$x_2 > x_R$ then bit 2 of code–$P_1 = 1$ $C_{2\ \text{right}} = 1$

$y_2 > y_B$ then bit 3 of code–$P_1 = 0$ $C_{2\ \text{bottom}} = 0$

$y_2 < y_T$ then bit 4 of code–$P_1 = 0$ $C_{2\ \text{top}} = 0$

code–$P_2 = 0010$.

Both code–$P_1 <> 0$ and code–$P_2 <> 0$

then $P_1 P_2$ not totally visible

code–P_1 AND code–$P_2 = 0000$

hence (code–P_1 AND code–$P_2 = 0$)

then line is not totally invisible.

As code–$P_1 <> 0$

NOTES

for $i = 1$

{

$C_{1\ \text{left}}\ (= 0) <> 1$ then nothing is done.

$i = i + 1 = 2$

}

code–$P_1 <> 0$ and code–$P_2 <> 0$

then P_1P_2 not totally visible.

code–P_1 AND code–$P_2 = 0000$

hence (code–P_1 AND code–$P_2 = 0$)

then line is not totally invisible.

for $i = 2$

{

$C_{1\ \text{right}}\ (=0) <> 1$ then nothing is done.

$i = i + 1 = 2 + 1 = 3$

}

code–$P_1 <> 0$ code–$P_2 <> 0$

then P_1P_2 not totally visible.

code–P_1 AND code–$P_2 = 0000$

hence (code–P_1 AND code–$P_2 = 0$)

then line is not totally invisible.

for $i = 3$

{

$C_{1\ \text{bottom}} = 1$ then find intersection of $P_1\ P_2$ with bottom edge

$y_B = 10$

$x_B = (180 – 120)(10–5)/(30 – 5) + 120$

$= 132$

then $P_1 = (132, 10)$

$x_1 > x_L$ then bit 1 of code–$P_1 = 0$ $C_{1\ \text{left}} = 0$

$x_1 < x_R$ then bit 2 of code–$P_1 = 0$ $C_{1\ \text{right}} = 0$

$y_1 = y_B$ then bit 3 of code–$P_1 = 0$ $C_{1\ \text{bottom}} = 0$

$y_1 < y_T$ then bit 4 of code–$P_1 = 0$ $C_{1\ \text{top}} = 0$

code–$P_1 = 0000$

$i = i + 1 = 3 + 1 = 4$

NOTES

}

code–P_1 = 0 but code–P_2 < > 0

then P_1P_2 not totally visible

code–P_1 AND code–P_2 = 0000

hence (code–P_1 AND code–P_2 = 0)

then line is not totally invisible

As code–P_1 =0

swap P_1 and P_2 along with the respective flags

P_1 = (180,30)

P_2 = (132,10)

code–P_1 = 0010

code–P_2 = 0000

$C_{1\ \text{left}}$	= 0	$C_{2\ \text{left}}$	= 0
$C_{1\ \text{right}}$	= 1	$C_{2\ \text{right}}$	= 0
$C_{1\ \text{bottom}}$	= 0	$C_{2\ \text{bottom}}$	= 0
$C_{1\ \text{top}}$	= 0	$C_{2\ \text{top}}$	= 0

Reset i = 1

for i = 1

{

$C_{1\ \text{left}}$ (= 0) < > 1 then nothing is done

$i = i + 1 = 1 + 1 = 2$

}

code–P_1 < > 0, and code–P_2 < > 0

then P_1P_2 not totally visible.

code–P_1 AND code–P_2 = 0000

hence (code–P_1 AND code–P_2 = 0)

then line is not totally invisible

for i = 2

{

$C_{1\ \text{right}}$ =1 then find intersection of P_1 P_2 with right edge

x_R = 160

$y_R = (30 - 5)(160 - 120)/(180 - 120) + 5$

NOTES

$= 21.6667$

$= 22$

then $P_1 = (160, 22)$

$x_1 > x_L$ then bit 1 of code–$P_1 = 0$ $\quad C_{1\ \text{left}} = 0$

$x_1 = x_R$ then bit 2 of code–$P_1 = 0$ $\quad C_{1\ \text{right}} = 0$

$y_1 > y_B$ then bit 3 of code–$P_1 = 0$ $\quad C_{1\ \text{bottom}} = 0$

$y_1 < y_T$ then bit 4 of code–$P_1 = 0$ $\quad C_{1\ \text{top}} = 0$

code–$P_1 = 0000$, $i = i + 1 = 2 + 1 = 3$

}

As both code–$P_1 = 0$ and code–$P_2 = 0$

then the line segment P_1P_2 is totally visible

So the visible portion of input line P_1P_2 is $P_1'P_2'$ where $P_1 = (160,22)$ & $P_2 = (132,10)$.

Considering the line EF

1. The endpoint codes are assigned

   $\text{code} - E \rightarrow 0101$

   $\text{code} - F \rightarrow 1001$

2. Flags are assigned for the two endpoints

   $E_{\text{left}} = 1$ (as x coordinate of E is less than x_L)

   $E_{\text{right}} = 0,\ E_{\text{top}} = 0,\ E_{\text{bottom}} = 1$

   Similarly,

   $F_{\text{left}} = 1,\ F_{\text{right}} = 0,\ F_{\text{top}} = 1,\ F_{\text{bottom}} = 0$

3. Since codes of E and F are both not equal to zero the line is not totally visible

4. Logical intersection of codes of E and F is not equal to zero. So we may ignore EF line and declare it as totally invisible

Considering the line GH

1. The endpoint codes are assigned

   $\text{code} - G \rightarrow 0000$

   $\text{code} - H \rightarrow 1000$

2. Flags are assigned for the two endpoints

   $G_{\text{left}} = 0, G_{\text{right}} = 0, G_{\text{top}} = 0, G_{\text{bottom}} = 0.$

   Similarly,

   $H_{\text{left}} = 0, H_{\text{right}} = 0, H_{\text{top}} = 1, H_{\text{bottom}} = 0.$

3. Since codes of G and H are both not equal to zero so the line is not totally visible

NOTES

4. Logical intersection of codes of G and H is equal to zero so we cannot declare it as totally invisible

5. Since code – $G = 0$, Swap G and H along with their flags and set $i = 1$

implying $G_{left} = 0,\ G_{right} = 0,\ G_{top} = 1,\ G_{bottom} = 0;$

$H_{left} = 0,\ H_{right} = 0,\ H_{top} = 0,\ H_{bottom} = 0;$

as $G \rightarrow 1000,\ H \rightarrow 0000$

6. Since code – $G <> 0$ then

for $i = 1$, {since $G_{left} = 0$

$i = i + 1 = 2$

go to 3

}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible

for $i = 2$, {since $G_{right} = 0$

$i = i + 1 = 3$

go to 3

}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible

for $i = 3$, {since $G_{bottom} = 0$

$i = i + 1 = 4$

go to 3

}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible

for $i = 4$, {since $G_{top} = 1$

Intersection with top edge, say $P(x, y)$ is found as follows

Any line passing through the points G, H and a point $P(x, y)$ is given by

$y - 20 = \{(80 - 20) / (140 - 120)\}\ (x - 120)$

or, $y - 20 = 3x - 360$

or, $y - 3x = -340$

Since the y coordinate of every point on line CD is 40, so we put $y = 40$ for the point of intersection $P(x, y)$ of line GH with edge CD

$40 - 3x = -340$

or, $-3x = -380$

or, $x = 380/3 = 126.66 \approx 127$

NOTES

So the point of intersection is P (127, 40)

We assign code to it.

Since the point lies on edge of the rectangle so the code assigned to it is 0000.

Now we assign G = (127, 40); $i = 4 + 1 = 5$.

conditions 3 and 4 are again checked.}

Since codes G and H are both equal to 0, so the line between H (120, 20) and G (127, 40) is totally visible.

3.10.3 Midpoint Subdivision Algorithm

Partially visible and totally invisible lines which cannot be identified by checking and operating (ANDing) endpoint codes are subdivided into two equal segments by finding the midpoint. Each half is then separately considered and tested with endpoint codes for immediate identification of totally visible and totally invisible state. Segments which cannot be identified even then are further subdivided at midpoint and each subdivision is subsequently tested. This bisection and testing procedure is continued until the intersection with a window edge is found with some specified accuracy. The sequential steps of the algorithm are given below.

Input : P_1 (x_1, y_1) P_2, (x_2, y_2), x_L, x_R, y_B, y_T
code assign (P_1), code assign (P_2)

Assume the function code assign ( ) assigns 4 bit code to a point with respect to the rectangular clipping window (refer Figure 3.56).

1. if code–P_1, code–P_2 both = 0, then (the line is totally visible) draw P_1P_2
2. if code–P_1 AND code–P_2, <> 0 then (line is totally invisible) ignore P_1P_2
3. $P_m = \frac{(P_1 + P_2)}{2}$; code assign (P_m)
4. if code–P_m <> 0

   then if code–P_1 AND code–P_m < > 0 then

   $P_1 = P_m$; go to 1

   else if code–P_2 AND code–P_m < > 0 then

   $P_2 = P_m$; go to 1

   end if

   end if
5. if code–P_m = 0 then

   if code–P_1, code–P_m both = 0 then consider P_2P_m

   else if code–P_2, code–P_m both = 0 then consider P_1P_m

   end if

   end if

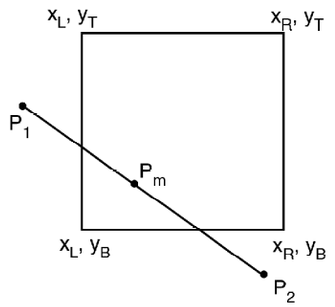


Fig. 3.40 *Rectangular Clipping Window*

6. Considering P_1P_m

$$\text{do } \{P_{m1} = \frac{(P_1 + P_m)}{2};\ \text{code assign } (P_{m1})$$

$$\text{if code–}P_{m1} <> 0 \text{ then } P_1 = P_{m1} \text{ else } P_m = P_{m1}$$

$$\}$$

while

$$(P_m \cdot x <> x_L \text{ and } P_m \cdot x <> x_R \text{ and } P_m \cdot y <> y_T \text{ and } P_m \cdot y <> y_B)$$

$$P_1 = P_m$$

7. Considering P_2P_m

$$\text{do } \{P_{m2} = \frac{(P_2 + P_m)}{2};\ \text{code assign } (P_{m2})$$

$$\text{if code–}P_{m2} <> 0 \text{ then } P_2 = P_{m2} \text{ else } P_m = P_{m2}$$

$$\}$$

while

$$(P_m \cdot x <> x_L \text{ and } P_m \cdot x <> x_R \text{ and } P_m \cdot y <> y_B \text{ and } P_m \cdot y <> y_T)$$

$$P_2 = P_m$$

8. Draw P_1P_2

In the above algorithm a calculated midpoint P_m is considered to be lying on any of the boundary lines of the window say boundary line $x = x_L$ if $P_m \cdot x = x_L \pm$ tolerance, where 'tolerance' is a very small number (say 0.1) prescribed depending on the precision of the display.

The midpoint subdivision algorithm, if implemented in hardware works very fast (even faster than Sutherland-Cohen algorithm) because hardware addition and division by 2 are very fast.

Example 3.13: Using midpoint subdivision algorithm find the visible portion of the line P_1P_2, P_1 (120, 5) P_2 (180, 30) with respect to a clipping window *ABCD* where *A* (100, 10), *B* (160, 10), *C* (160, 40), *D* (100, 40).

Solution: INPUT: P_1 (120, 5), P_2(180, 30)

$x_L = 100$, $x_R = 160$, $y_B = 10$, $y_T = 40$

$x_1 > x_L$ then bit 1 of code–$P_1 = 0$

$x_1 < x_R$ then bit 2 of code–$P_1 = 0$

$y_1 < y_B$ then bit 3 of code–$P_1 = 1$

$y_1 < y_T$ then bit 4 of code–$P_1 = 0$

code–P_1 = 0100,

$x_2 > x_L$ then bit 1 of code–$P_1 = 0$

$x_2 > x_R$ then bit 2 of code–$P_1 = 1$

$y_2 > y_B$ then bit 3 of code–$P_1 = 0$

$y_2 < y_T$ then bit 4 of code–$P_1 = 0$

code–P_2 = 0010.

code–P_1 < > 0 and code– P_2 < > 0

then P_1P_2 not totally visible.

code–P_1 AND code–P_2 = 0000

hence (code–P_1 AND code–P_2 = 0)

then line is not totally invisible

$P_m = (P_1 + P_2)/2 = ((120+180)/2, (5+30)/2) = (150 , 17.5)$

$x_m > x_L$ then bit 1 of code–P_m =0

$x_m < x_R$ then bit 2 of code–P_m =0

$y_m > y_B$ then bit 3 of code–P_m =0

$y_m < y_T$ then bit 4 of code–P_m =0

P_1	code–P_1	P_2	code–P_2	P_m	code–P_m	Comment
120,5	0100	180,30	0010	150,18	0000	Save P_2P_m, continue with P_1P_m
120,5	0100	150,18	0000	135,12	0000	Continue P_1P_m
120,5	0100	135,12	0000	128,9	0100	Continue P_mP_2
128,9	0100	135,11	0000	132,10	0000	Succeeds

code–P_m = 0000, i.e., code–P_m = 0

(Calculating the point by rounding it)

So one of the intersection parts has been found at (132,10)

For the second intersection point

P_1	code–P_1	P_2	code–P_2	P_m	code–P_m	Comment
120,5	0100	180,30	0010	150,18	0000	Continue with P_mP_2
150,18	0000	180,30	0010	165,24	0010	Continue with P_1P_m
150,18	0000	165,24	0010	158,21	0000	Continue with P_mP_2
158,21	0000	165,24	0010	162,23	0010	Continue with P_1P_m
158,21	0000	162,23	0010	160,22	0000	Succeeds

So the second point of intersection has been found at (160, 22)

NOTES

Polygon Clipping

So far you have learnt techniques to clip any line by a clipping window. Using any of these techniques we can attempt to clip a polygon because a polygon is simply a set of connected straight lines (edges). But the problem is, each edge clipped separately using a line clipping algorithm will certainly not produce a truncated polygon as one would expect. Rather it would produce a set of unconnected line segments as if the polygon is exploded. Herein lies the need to use a different clipping algorithm to output truncated yet bounded region(s) from input polygon. *Sutherland - Hodgman* algorithm is one such standard method for clipping arbitrary shaped polygons with a rectangular clipping window.

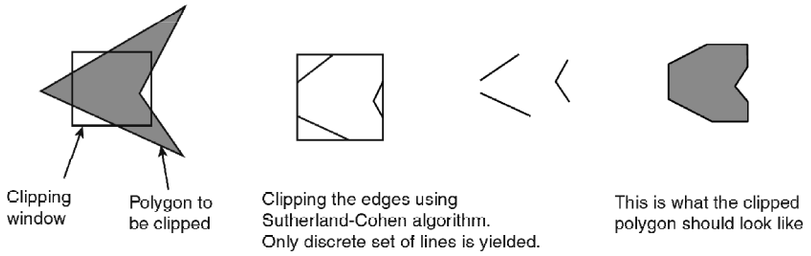


Fig. 3.41 *Clipping a Polygon*

To clip a polygon correctly as shown in Figure 3.41, the polygon must be tested against each edge of the clip rectangle; new edges must be added, and existing edges must be discarded, retained or divided. Even multiple polygons may result from clipping a single polygon. This is unlike the case of Sutherland-Cohen line clipping which tests the line endpoint outcode to see which clip-edge is crossed and clips only when necessary.

Sutherland–Hodgman clipper actually follows a *divide and conquer* strategy. It decomposes the problem of polygon-clipping against a clip window into identical subproblems. A subproblem is to clip all polygon edges (pair of vertices) in succession against a single infinite clip edge. The output is a set of clipped edges or pair of vertices that fall in the visible side with respect to that clip edge. These set of clipped edges or output vertices are considered as input to the next subproblem of clipping against the second window edge. Thus considering the output of the previous subproblem as the input, each of the subproblems are solved sequentially, finally yielding the vertices that fall on or within the window boundary. These vertices connected in order form the shape of the clipped polygon that may be optionally scan filled.

While clipping polygon edges against a window edge we move from one vertex (V_i) to the next vertex (V_{i+1}) and decide the output vertex according to the four simple rules stated below (refer Figure 3.42).

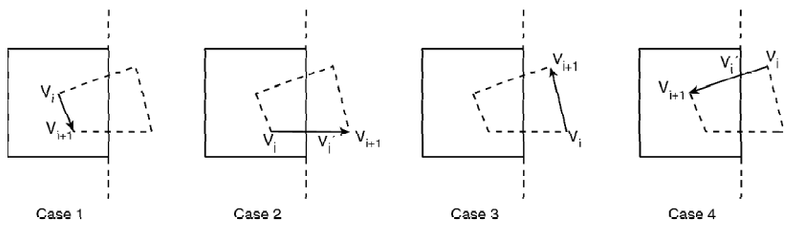


Fig. 3.42 *Clipping against a Window Edge*

	V_i	→ V_{i+1}	Output Vertex
Rule 1:	inside (window)	inside	V_{i+1}
Rule 2:	inside	outside	V_i', the intersection with the window edge
Rule 3:	outside	outside	none
Rule 4:	outside	inside	V_i', V_{i+1}

NOTES

For edge $V_i \rightarrow V_{i+1}$, the starting vertex V_i is assumed to have already been considered as a candidate for output while it was the terminating vertex of the edge $V_{i-1} \rightarrow V_i$.

The intersection point V_i' is easily found as the x or y coordinates of V_i, V_{i+1} and the concerned window edge are already known. By assigning a 4 bit outcode to every vertex we can determine whether the point falls on the visible side (when code = 0) of the window edge or on the other side (when code ≠ 0).

Now let us see in steps what exactly happens while clipping our initial arrow shaped quadrilateral against a rectangular clip window following the above rules. Note the vertex list at each step.

STEP 1 – Clip against Left edge [refer Figure 3.43(a)]

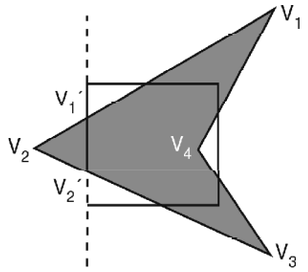


Fig. 3.43(a) *Clip Against Left Edge*

Input vertex list $[V_1, V_2, V_3, V_4]$

Edge $V_1 \rightarrow V_2$: output V_1'

Edge $V_2 \rightarrow V_3$: output V_2', V_3

Edge $V_3 \rightarrow V_4$: output V_4

Edge $V_4 \rightarrow V_1$: output V_1

Output vertex list $[V_1', V_2', V_3, V_4, V_1]$

STEP 2 – Clip against Bottom edge [refer Figure 3.43(b)]

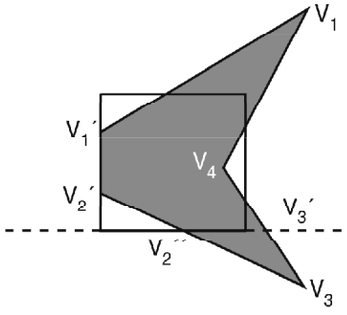


Fig. 3.43(b) *Clip Against Bottom Edge*

NOTES

Input vertex list $[V_1', V_2', V_3, V_4, V_1]$

Edge $V_1' \rightarrow V_2'$: output V_2'

Edge $V_2' \rightarrow V_3$: output V_2''

Edge $V_3 \rightarrow V_4$: output V_3', V_4

Edge $V_4 \rightarrow V_1$: output V_1

Edge $V_1 \rightarrow V_1'$: output V_1'

Output vertex list $[V_2', V_2'', V_3', V_4, V_1, V_1']$

STEP 3 – Clip against Right edge [refer Figure 3.43(c)]

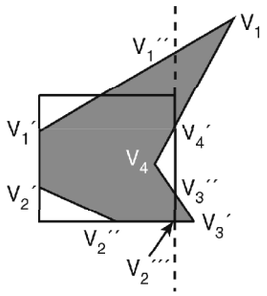


Fig. 3.43(c) *Clipping Against Right Edge*

Input vertex list $[V_2', V_2'', V_3', V_4, V_1, V_1']$

Edge $V_2' \rightarrow V_2''$: output V_2''

Edge $V_2'' \rightarrow V_3'$: output V_2'''

Edge $V_3' \rightarrow V_4$: output V_3'', V_4

Edge $V_4 \rightarrow V_1$: output V_4'

Edge $V_1 \rightarrow V_1'$: output V_1'', V_1'

Edge $V_1' \rightarrow V_2'$: output V_2'

Output vertex list $[V_2'', V_2''', V_3'', V_4, V_4', V_1'', V_1', V_2']$

STEP 4 – Clip against Top edge [refer Figure 3.43(d)]

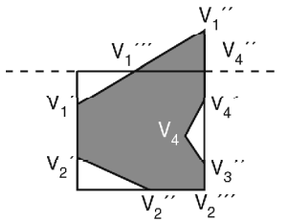


Fig. 3.43(d) *Clipping Against Top Edge*

Input vertex list $[V_2'', V_2''', V_3'', V_4, V_4', V_1'', V_1', V_2']$

Edge $V_2'' \rightarrow V_2'''$: output V_2'''

Edge $V_2''' \rightarrow V_3''$: output V_3''

Edge $V_3'' \rightarrow V_4$: output V_4

Edge $V_4 \rightarrow V_4'$: output V_4'

Edge $V_4' \rightarrow V_1''$: output V_4''

Edge $V_1'' \rightarrow V_1'$: output V_1''', V_1'

Edge $V_1' \rightarrow V_2'$: output V_2'

Edge $V_2' \rightarrow V_2''$: output V_2''

Output vertex list [V_2''', V_3'', V_4, V_4', V_4'', V_1''', V_1', V_2' V_2'']

The final clipped polygon, [refer Figure 3.43(e)]

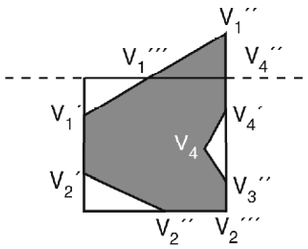


Fig. 3.43(d) *Clipping Against Top Edge*

Output vertex list [V_2''', V_3'', V_4, V_4', V_4'', V_1''', V_1', V_2', V_2'']

We started off with 4 vertices and ended up with 9 new vertices except V_4, which survived the 4-step revision of vertices.

Because clipping against one clip edge is independent of all others, it is possible to arrange the clipping stages in a pipeline. The input polygon is clipped against one edge and any points that are kept are passed on as input to the next stage of the pipeline. This way four polygons can be at different stages of the clipping process simultaneously for a rectangular clipping window. This is often implemented in hardware. The same algorithm, slightly modified, is found more suitable for hardware implementation. The modified approach recursively clips a single polygon edge (vertex) against all the window edges. The portion of the edge (if any) finally lying within the window boundary is saved. The process is repeated for the other polygon edges one after another. Thus this method generates the final clipped polygon without creating and storing any intermediate polygon definitions. If there are *n* polygon edges this method executes *n* cycles of window edge (4 per cycle) processing whereas the previous approach looped through 4 cycles of polygon-edge (^{3} *n* per cycle) processing.

Though the above discussion assumes only rectangular clip window, the Sutherland –Hodgman algorithm will clip any polygon, convex or concave, against any convex polygonal clipping window. So the usual code-testing method for determining the visibility of a polygon vertex with respect to an arbitrary clip edge will not be applicable. We may use the following simple rule to determine whether a point is inside, on or outside the clip-edge. Remember if the successive edges of the clipping polygon are considered anticlockwise the inside of the polygon is always to the left; if considered clockwise it is to the right.

NOTES

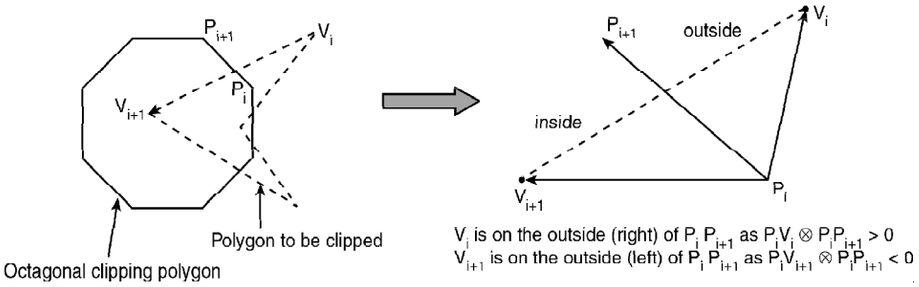


Fig. 3.44 *Octagonal Clipping Polygon*

If the position of any polygon vertex V_i is to be determined with respect to a clipping edge P_iP_{i+1} in general, then the sign of the cross product of the vectors P_iV_i and P_iP_{i+1} is tested (refer Figure 3.44).

If sign of $P_iV_i \otimes P_iP_{i+1}$ is +ve, then V_i is to the right of line P_iP_{i+1}

If sign of $P_iV_i \otimes P_iP_{i+1}$ is –ve, then V_i is to the left of line P_iP_{i+1}

If $P_iV_i \otimes P_iP_{i+1} = 0$, then V_i is on the line P_iP_{i+1}

From the basic right hand rule of cross product of vectors, the above interpretation is easily understandable.

Pseudocode for Sutherland–Hodgman Algorithm

Define variables

inVertexArray is the array of input polygon vertices

outVertexArray is the array of output polygon vertices

Nin is the number of entries in inVertexArray

Nout is the number of entries in outVertexArray

n is the number of edges of the clip polygon

ClipEdge [x] is the *x*th edge of clip polygon defined by a pair of vertices

s, *p* are the start and end point respectively of current polygon edge

i is the intersection point with a clip boundary

j is the vertex loop counter

Define Functions

AddNewVertex (newVertex, Nout, outVertexArray)

: Adds newVertex to outVertexArray and then updates Nout

InsideTest (testVertex, clipEdge[*x*])

: Checks whether the vertex lies inside the clip edge or not; returns TRUE if inside else returns FALSE

Intersect (first, second, clipEdge[*x*])

: Clips polygon edge (first, second) against clipEdge[x], outputs the intersection point

{ : begin main

$x = 1$

NOTES

```
while (x ≤ n)              : Loop through all the n clip edges
{
Nout = 0                   : Flush the outVertexArray
s = inVertexArray[Nin]     : Start with the last vertex in inVertexArray
for j = 1 to Nin do        : Loop through Nin number of polygon vertices (edges)
        {
         p = inVertexArray[j]
          if InsideTest (p, clipEdge[x]) = = TRUE then :    Cases 1 and 4
            if InsideTest(s, clipEdge[x]) = = TRUE then
              AddNewVertex (p, Nout, outVertexArray)  :    Case 1
              else
                i = Intersect(s, p, clipEdge[x])            :    Case 4
               AddNewVertex (i, Nout, outVertexArray)
               AddNewVertex (p, Nout, outVertexArray)
          end if
          else                          : i.e., if InsideTest (p, clipEdge[x]) = = FALSE
                                                  (Cases 2 and 3)
            if InsideTest(s, clipEdge[x]) = = TRUE then :    Case 2
              {
                Intersect(s, p, clipEdge[x])
               AddNewVertex (i, Nout, outVertexArray)
           end if                       :  No action for case 3
           s = p                        :  Advance to next pair of vertices
           j = j + 1
         end if                         :  end {for}
         }
      x = x + 1                         :  Proceed to the next ClipEdge[x + 1]
                                           Nin = Nout
       inVertexArray = outVertexArray   :  The ouput vertex array for the current
clip                                          edge becomes the input vertex
array for the                                 next clip edge
       }                                :  end while
       }                                :  end main
```

Text Clipping

Normalization transformation (*N*) maps world coordinates (or viewing coordinates) to normalized device coordinates and workstation transformation (*W*) maps normalized device coordinates to physical device coordinates. In general, the mapping of a part of a world coordinate scene to device coordinates is referred as

NOTES

viewing transformation (V), mathematically expressed as,

$$V = W.N$$

Sometimes the two dimensional viewing transformation is simply called window-to-viewport transformation.

By defining a closed boundary or *window* the enclosed portion of a world coordinate scene is clipped against the window boundary and the data of the clipped portion is extracted for mapping to a separately defined region known as *viewport* (refer Figure 3.45). While window selects a part of the scene, viewport displays the selected part at desired location on the display area. When the window is changed we see a different part of the scene at the same portion (viewport) on the display. If we change the viewport only, we see the same part of the scene drawn at different scale or at different place on the display. By successively increasing or decreasing the size of the window around a part of the scene the viewport remaining fixed, we can get the effect of zoom out or zoom in, respectively on the displayed part.

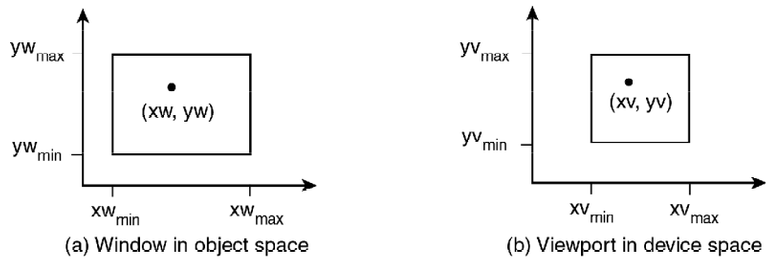


Fig. 3.46 *Window and Viewport Clipping*

Window or viewport can be general polygon shaped or circular. For simplicity here we will consider rectangular window and rectangular viewport, with edges of the rectangles being parallel to the coordinate axes.

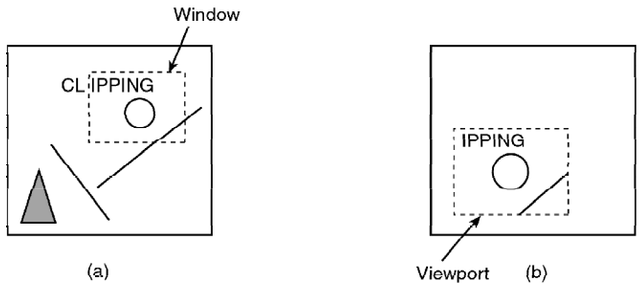


Fig. 3.47 *Text Clepping*

Figure 3.62 shows how out of several objects drawn in the object space only those clipped by the window are displayed in the viewport in image space; note that the viewport objects are larger than the window objects though the object shapes are not distorted because the viewport is a uniformly scaled version of the window.

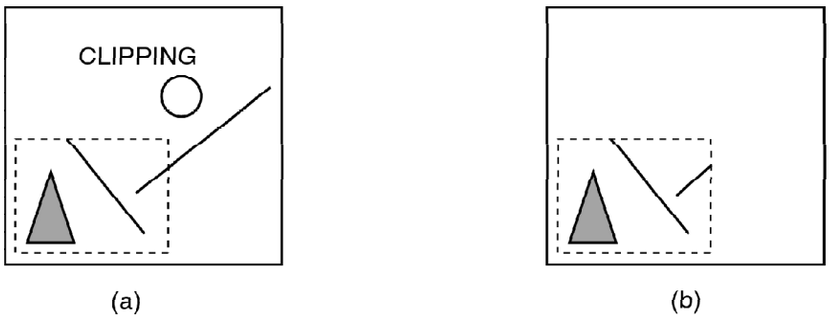


Fig. 3.48 *Window is Changed and Different Objects of the Scene Displayed in the Same Viewport*

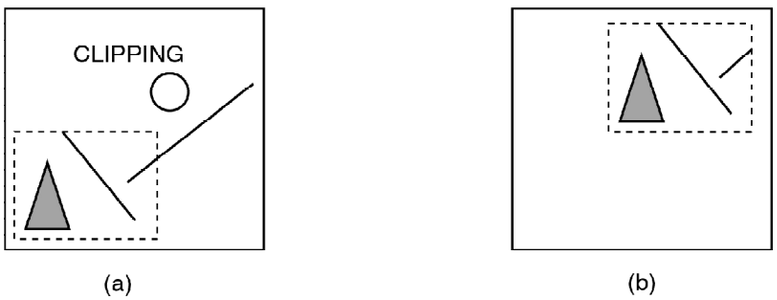


Fig. 3.49 *Only the Viewport is Moved and the Window Remains Same and Same Objects Displayed through the Viewport at a Different Position*

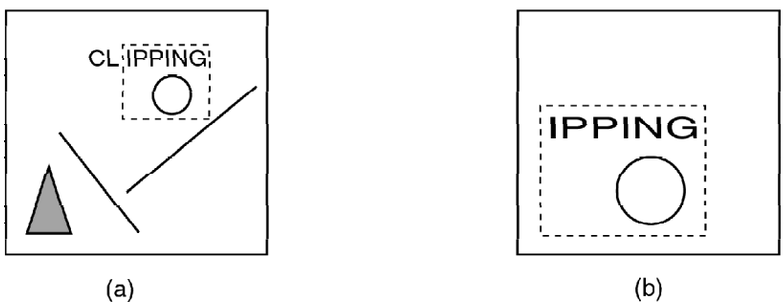


Fig. 3.50 *Zooming In*

Let us consider a point (*xw*, *yw*) within the window enclosure as shown in Figure 3.45. The window is mapped to the viewport such that (*xw*, *yw*) transform to (*xv*, *yv*) in the device space.

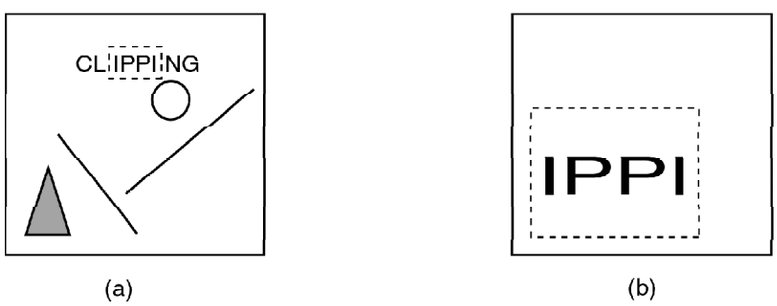


Fig. 3.50 *The Viewport Remaining Fixed the Window Size is Gradually Reduced, Maintaining the Scale Factors same the Zooming Effect is Obtained through the Viewports*

NOTES

To maintain the same relative placement of the point in the viewport as in the window,

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

And also,

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

This implies,

$$xv = xv_{\min} + (xw - xw_{\min}) \left(\frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}} \right)$$

$$yv = yv_{\min} + (yw - yw_{\min}) \left(\frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}} \right)$$

If $\left(\frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}} \right)$ be termed as s_x (the x scale factor for scaling the window to the size of the viewport) and $\left(\frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}} \right)$ be termed as s_y (the y scale factor for window-to-viewport scaling) then,

$$xv = xv_{\min} + (xw - xw_{\min}) s_x$$

and

$$yv = yv_{\min} + (yw - yw_{\min}) s_y$$

In terms of two-step geometric transformation the above relation can be interpreted as,

Step 1 Scaling the window area to the size of the viewport with scale factors s_x and s_y with respect to a fixed point $(xw_{\min}, yw_{\min})$.

$$\Rightarrow [T_1] = \begin{pmatrix} s_x & 0 & xw_{\min}(1 - s_x) \\ 0 & s_y & yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{pmatrix}$$

Step 2 Translating the scaled window to the position of the viewport so that,

$$\Delta x = xv_{\min} - xw_{\min} \quad \text{and}$$

$$\Delta y = yv_{\min} - yw_{\min}$$

$$\Rightarrow [T_2] = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

Concatenating $[T_1]$ and $[T_2]$ we get,

$$[T] = [T_1][T_2] = \begin{pmatrix} s_x & 0 & \Delta x + xw_{\min}(1 - s_x) \\ 0 & s_y & \Delta y + yw_{\min}(1 - s_y) \\ 0 & 0 & 1 \end{pmatrix}$$

Replacing the values of Δx and Δy in the above transformation matrix we finally get,

$$[T] = \begin{pmatrix} s_x & 0 & -s_x xw_{\min} + xv_{\min} \\ 0 & s_y & -s_y yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{pmatrix}$$

The aspect ratio of a rectangular window or viewport is defined by

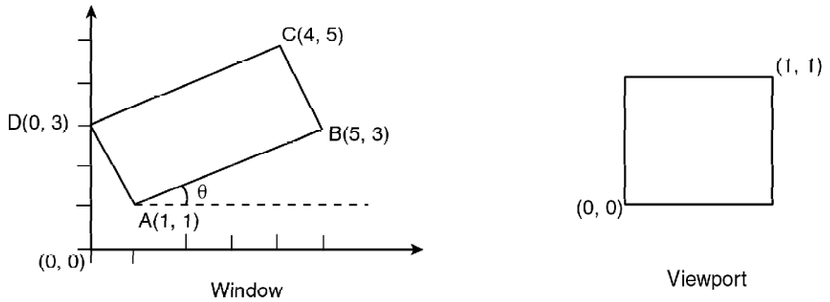
$$a = \frac{x_{\max} - x_{\min}}{y_{\max} - y_{\min}}$$

If $s_x = s_y$ then $\frac{xv_{\max} - xv_{\min}}{xw_{\max} - xw_{\min}} = \frac{yv_{\max} - yv_{\min}}{yw_{\max} - yw_{\min}}$

$$\Rightarrow \frac{xv_{\max} - xv_{\min}}{yv_{\max} - yv_{\min}} = \frac{xw_{\max} - xw_{\min}}{yw_{\max} - yw_{\min}} \Rightarrow a_v = a_{\hat{w}}$$

So it can be said that if the aspect ratio a_v of the viewport equals the aspect ratio a_w of the window, then $s_x = s_y$ and no distortion of displayed scene occurs other than uniform magnification or compression. If $a_v \neq a_w$ then the displayed scene in the viewport gets somewhat distorted with respect to the scene captured by the window.

Example 3.14: Find the normalization transformation *N* which uses the rectangle *A* (1, 1), *B* (5, 3), *C* (4, 5) and *D* (0, 3) as a window and the normalized device screen as the viewport.



Solution: Here we see that the window edges are not parallel to the coordinate axes. So we will first rotate the window about *A* so that it is aligned with the axes.

Now, $\tan\theta = \frac{3-1}{5-1} = \frac{1}{2}$

$\Rightarrow \sin\theta = \frac{1}{\sqrt{5}}, \quad \cos\theta = \frac{2}{\sqrt{5}}$

Here we are rotating the rectangle in clockwise direction. So θ is (–)ve, i.e., –θ.

The rotation matrix about *A*(1, 1) is,

NOTES

$$[T_{R,\theta}]_A = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \left(\frac{1-3}{\sqrt{5}}\right) \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & \left(\frac{1-1}{\sqrt{5}}\right) \\ 0 & 0 & 1 \end{pmatrix}$$

The x extent of the rotated window is the length of $\overline{AB}$ which is $\sqrt{(4^2+2^2)} = 2\sqrt{5}$

Similarly, the y extent is length of $\overline{AD}$ which is $\sqrt{(1^2+2^2)} = \sqrt{5}$

For scaling the rotated window to the normalized viewport we calculate s_x and s_y as,

$$s_x = \frac{\text{viewport } x \text{ extent}}{\text{window } x \text{ extent}} = \frac{1}{2\sqrt{5}}$$

$$s_y = \frac{\text{viewport } y \text{ extent}}{\text{window } y \text{ extent}} = \frac{1}{\sqrt{5}}$$

As in expression (1), the general form of transformation matrix representing mapping of a window to a viewport is,

$$[T] = \begin{pmatrix} s_x & 0 & -s_x xw_{\min} + xv_{\min} \\ 0 & s_y & -s_y yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{pmatrix}$$

In this problem $[T]$ may be termed as N as this is a case of normalization transformation with,

$$xw_{\min} = 1 \qquad xv_{\min} = 0$$

$$yw_{\min} = 1 \qquad yv_{\min} = 0$$

$$s_x = \frac{1}{2\sqrt{5}} \qquad s_y = \frac{1}{\sqrt{5}}$$

By substituting the above values in $[T]$, i.e., N,

$$N = \begin{pmatrix} \frac{1}{2\sqrt{5}} & 0 & \left(\frac{-1}{2}\right)\frac{1}{\sqrt{5}} + 0 \\ 0 & \frac{1}{\sqrt{5}} & \left(\frac{-1}{\sqrt{5}}\right)1 + 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now we compose the rotation and transformation N to find the required viewing transformation N_R

$$N_R = N\,[T_{R,\theta}]_A = \begin{pmatrix} \frac{1}{2\sqrt{5}} & 0 & \frac{-1}{2\sqrt{5}} \\ 0 & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{5} & 1-\frac{3}{\sqrt{5}} \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 1-\frac{1}{\sqrt{5}} \\ 0 & 0 & 1 \end{pmatrix}$$

Check Your Progress

11. Define the term ordered edge list algorithm.
12. What is an interior defined region?
13. What is the difference between a four connected and an eight connected method?
14. How does the stack become large in scan line seed fill algorithm?
15. Define the term antialiasing.
16. Write an advantage of Sutherland-Cohen algorithm over the explicit line clipping algorithm.

3.11 ANSWERS TO 'CHECK YOUR PROGRESS'

1. The derivative of a product of two functions = (the derivative of first function × second function) + (first function × derivative of second function).
2. The chain rule is the most important and widely used rule for differentiation. The rule states that if y is a differentiable function of z, and z is a differentiable function of x, then y is a differentiable function of x.
3. The Digital Differential Analyser (DDA) is used to rasterize lines, triangles and polygons.
4. In generalized Bresenham's algorithm, the line endpoints (xs, ys) and (xe, ye) which are assumed to be not equal are taken as input and the variables are assumed to be integers.
5. In Bresenham's circle generation algorithm, only integer arithmetic is used. The algorithm is based on the differential equation of a circle.
6. The process of converting the rasterized picture stored in a frame buffer to the rigid display pattern of video is called as scan conversion or rasterization.
7. The three basic methods to generate characters in computer screen are hardware based methods, vector based methods and bit map based methods.
8. Polygon filling is the process of colouring the area of a polygon.
9. The three types of seed fill algorithms are, flood fill algorithm, boundary fill algorithm and scan-line polygon fill algorithm.
10. The function of the scan-line polygon fill algorithm is to find the intersection point of the boundary of polygon and the scan lines.
11. Alternate techniques for efficiently scan-converting solid area polygons know as ordered edge list algorithm.

NOTES

12. An Interior-defined region is a collection or patch of same color contiguous pixels.
13. When we consider neighbours to the left-right, top and bottom only this method is called four connected method whereas when the four diagonal pixels are also included as neighbours then this method is called eight connected method.
14. The seed fill algorithm makes the stack become quite large because in every loop the algorithm pushes atmost 4 to 8 pixels into the stack. The stack frequently contains duplicate pixels.
15. The process of minimization of aliasing is called as antialiasing.
16. An advantage of Sutherland-Cohen algorithm is that it reduces the number of intersection calculations unlike the explicit line clipping algorithm.

3.12 SUMMARY

- The derivative of a product of two functions = (the derivative of first function × second function) + (first function × derivative of second function).
- The derivative of a constant function is equal to the constant multiplied by the derivative of the function.
- The chain rule is the most important and widely used rule for differentiation. The rule states that if y is a differentiable function of z, and z is a differentiable function of x, then y is a differentiable function of x.
- The Digital Differential Analyser (DDA) is used to rasterize lines, triangles and polygons.
- Bresenham's algorithm samples a line by incrementing by one unit either x or y, depending on the slope of the line and then selects the pixel lying at least distance from the true line path at each sampling position.
- In Bresenham's circle generation algorithm, only integer arithmetic is used. The algorithm is based on the differential equation of a circle.
- The process of converting the rasterized picture stored in a frame buffer to the rigid display pattern of video is called as scan conversion or rasterization.
- This encoding considers the pictures linearly or one-dimensionally. In many pictures, we observe that the large number of pixels in the picture have the same intensity or color. So, a picture can be defined by pixel intensity and the number of pixels on given scanline with the same intensity. Such a representation of picture is called as run-length encoding.
- We can encode the raster as a set of rectangular or square area called as cell encoding.
- The three basic methods to generate characters in computer screen are hardware based methods, vector based methods and bit map based methods.

- In bitmap based method a bold face character is obtained by writing the corresponding 'normal' character mask in consecutive frame buffer ? locations.
- Polygon filling is the process of colouring the area of a polygon. Area may be defined as total number of pixels outlining the polygon. The polygon defined by the total number of pixels is called as interior defined, and the algorithms used for filling the area are known as flood fill algorithms. The polygon defined by the bounding pixels that outline the polygon is called the boundary defined, and the corresponding algorithms are termed as boundary fill algorithms.
- Alternate techniques for efficiently scan-converting solid area polygons know as ordered edge list algorithm.
- Algorithms used for filling interior defined regions are generally known as flood fill algorithms.
- A scan line seed fill algorithm minimizes stack size by pushing to the stack only one pixel in any uninterrupted unfilled span of pixels in a single scan line or row of pixels in a boundary defined region.
- This process of distortion of information due to low frequency sampling or under sampling is called as aliasing.
- The process of minimization of aliasing is called as antialiasing.
- For clipping totally invisible lines and partially visible lines, the Sutherland Cohen algorithm breaks the line segments into smaller sub segments by finding intersection with appropriate window edges.
- In midpoint subdivision algorithm, partially visible and totally invisible lines cannot be identified by checking and operating endpoint codes are subdivided into two equal segments by finding the midpoint.
- Sutherland-Hodgman algorithm is one of the methods for clipping arbitrary shaped polygons with a rectangular clipping window.

3.13 KEY TERMS

- **Derivative:** It refers to the instantaneous rate of change of a function.
- **Scan Conversion:** The process of converting the rasterized picture stored in a frame buffer to the rigid display pattern of video is called as scan conversion.
- **Cell encoding:** We can encode the raster as a set of rectangular or square area called as cell encoding.
- **Polygon filling:** It is the process of colouring the area of a polygon.
- **Ordered edge list algorithm:** Alternate techniques for efficiently scan-converting solid area polygons know as ordered edge list algorithm.
- **Interior defined region:** It is a collection or patch of same color contiguous pixels. Pixels exterior to the region have different colors.

NOTES

- **Four connected method:** When we consider neighbours to the left, right, top and bottom only then this method is called four connected method.
- **Eight connected method:** When the four diagonal pixels are also included as neighbours then this method is called eight connected method.
- **Antialiasing:** The process of minimization of aliasing is called as antialiasing.

3.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Differentiate y=log [sun (3x2 + 5)] with respect to x.
2. Differentiate y = tan2.
3. When are DDA and Bresenham's algorithms used?
4. What is solid area scan conversion?
5. What is the real-time conversion?
6. How will you define frame buffers?
7. Define the term raster addressing.
8. Why is character mask used?
9. What is boundary fill algorithm?
10. What is edge flag algorithm?
11. Differentiate between interior defined and boundary defined area filling techniques.
12. What is edge fill algorithm?
13. Define the term half toning.
14. What is clipping?

Long-Answer Questions

1. Discuss the differentiable function with the help of appropriate example.
2. What is DDA algorithm? Explain how it functions.
3. Why is algorithm developed by Bresenham considered more accurate than DDA algorithm? Explain with the help of Bresenham's line algorithm.
4. Explain the Bresenham's concept of circle generation with an example.
5. Explain the Bresenham's algorithm or pseudocode with an example .
6. Describe the run-length encoding and cell organization with the help of diagram.
7. Discuss the scan line method with the help of appropriate example.
8. How are characters generated? Explain with the help of examples.
9. Explain the scan-line polygon fill algorithm with the help of diagram.

10. Discuss flood fill algorithm with the help of appropriate example.
11. Describe the simple ordered edge list algorithm.
12. Explain the more efficient ordered edge list algorithms giving appropriate example.
13. Explain the scan line seed fill algorithm with the help of example.
14. Discuss the various types of antialiasing with the help of diagram.
15. Describe the integral convolution and antialiasing.
16. Describe the various clipping techniques with the help of examples and algorithms.

NOTES

3.15 FURTHER READING

Hearn, Donald and M. Pauline Baker. *Computer Graphics*. New Jersey: Prentice Hall.

Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Higher Education.

Foley, James D., Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice*. London: Addison-Wesley Professional.

Mukhopadhyay, Anirban and Arup Chattopadhyay. *Introduction to Computer Graphics and Multimedia*. New Delhi: Vikas Publishing House Pvt Ltd.

UNIT 4 2D AND 3D TRANSFORMATION

NOTES

Structure

4.0 INTRODUCTION

In 2D transformations, Graphics packages, simple or specialized, have the facility of changing the shape or size or position or orientation of displayed objects. Animations are produced by resizing an object or moving object or camera along animation path. They require the application of various geometric transformations, such as translation, rotation, scaling, etc. Each of these transformations is represented by specific transformation matrices. This unit demonstrates the nature and origin of these matrices with 2D point object being considered as representative of any graphic object.

The concept of 3-Dimensional (3-D) graphics as well as geometric transformations. Geometric transformations are defined as changing of position for given images or graphics. Literal meaning of transformation is to alter the position of an object on a plane. The geometrical transformation refers to various processes, such as translation, scaling, rotation, reflection and shearing. To perform rotation about a line that is not parallel to one of the coordinate axes we first need to align the line with one of the coordinate axes through some intermediate transformation.

In this unit you will study about the 2D-transformation, transformations matrix, rotation, scaling, rotation, combined transformations and homogeneous coordinates, translation, 3D transformation, geometrical transformations, scaling, rotation, reflection and shearing.

4.1 OBJECTIVES

NOTES

After going through this unit, you will be able to:

- Explain the 2D-transformation
- Discuss the transformations and matrix
- Explain the process of rotation, scaling and rotation
- Describe the various combined transformations and homogeneous coordinates
- Describe the 3D transformation
- Explain the meaning of geometrical transformations
- Discuss the terms scaling, rotation, reflection and shearing

4.2 2D-TRANSFORMATION

So far we have discussed *geometric transformation* of 2-D objects which are well defined with respect to a *global coordinate system*, also called the *World Coordinate System(WCS)*–the principal frame of reference. But it is often found convenient to define quantities independent of the WCS with respect to a *local coordinate system*, also called the *model coordinate system* or the *User Coordinate System(UCS)*. While the UCS may vary from entity to entity and as per convenience, the WCS being the master reference system remains fixed for a given display system. Once you define an object 'locally' you can place it in the global system simply by specifying the location of the origin and orientation of the axes of the local system within the global system, then mathematically transforming the point coordinates defining the object from local to global system. Such transformations are known as the transformation between the coordinate systems. Here we will briefly discuss only the transformations between two cartesian frames of reference.

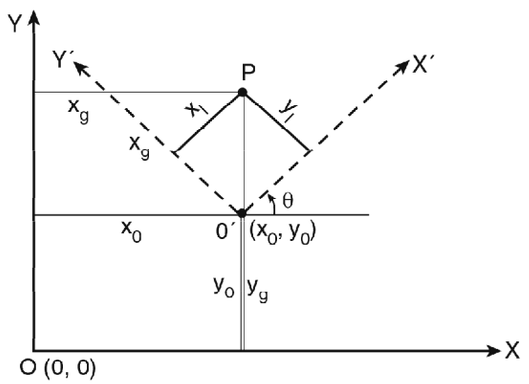


Fig. 4.1 *The Coordinates of Point P with Respect to Local Coordinate Sytem X´O´Y´ is, (x_l, y_l) while its Coordinates with respect to WCS XOY is $(x´_g, y´_g)$*

Local to Global: Figure 4.1 shows two Cartesian systems global *XOY* and local *X′OY′* with coordinate origins at (0, 0) and (x_0, y_0) respectively and with an of angle θ between the *X* and *X′* axes. To transform object descriptions (x_l, y_l) with respect to local system to that x_g, y_g with respect to global system we need to follow the following two steps that superimpose the *XOY* frame to the *X′O′Y′* frame.

Step 1

Translation: So that origin '*O*' of the *XY* system moves to origin '*O*' of the *X′Y′* system.

Transformation matrix:

$$[T_T]_{x_0,\, y_0} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 2

Rotation: So that *X* aligns with *X′* and *Y* aligns with *Y′*

Transformation matrix:

$$[T_T]_{\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So the global coordinates (x_g, y_g) of point *P* are expressed in terms of its known local coordinate (x_l, y_l) as,

$$\begin{pmatrix} x_g \\ y_g \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_l \\ y_l \\ 1 \end{pmatrix}$$

Global to Local: Unlike the previous case, if the object is originally defined in the global *XY* system then to transform its description in local *X′Y′* system. we have to superimpose local *X′O′Y′* frame to the global *XOY* frame. So the transformations required this time are,

1. $[T_T]_{-x_0,\, -y_0} = \begin{pmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{pmatrix}$ and

2. $[T_R]_{-\theta} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

The description of point *P* in the local system is given by

$$\begin{pmatrix} x_l \\ y_l \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_g \\ y_g \\ 1 \end{pmatrix}$$

NOTES

Affine Transformation: All the two dimensional transformations where each of the transformed coordinates x' and y' is a linear function of the original coordinates x and y as

$$\left.\begin{aligned} x' &= A_1 x + B_1 y + C_1 \\ y' &= A_2 x + B_2 y + C_2 \end{aligned}\right\}$$ where A_i, B_i, C_i are parameters fixed for a given transformation type.

2-D transformation of coordinate systems, translation, rotation, scaling, reflection, shear-all are examples of 2-D affine transformations and exhibit the general property, that parallel lines transform to parallel lines and finite points map to finite points. An affine transformation involving only translation, rotation and reflection preserves the length and angle between two lines.

4.2.1 Representation of Points

A point's coordinates can be expressed as elements of a matrix. Two matrix formats are used: one is row matrix and other is column matrix format.

For a 2-D point (x, y) the row matrix format is a 1-row, 2-column matrix $[x\ y]$.

For a 3-D point (x, y, z) it's a 1-row, 3-column matrix $[x\ y\ z]$. The same points in the column matrix format can be expressed as 1-column, 2-row matrix, $\begin{pmatrix} x \\ y \end{pmatrix}$ and 1-column 3-row matrix $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$, respectively.

In this Unit we will follow column matrix convention. As all graphic objects are built with finite number of points each such object can be uniquely represented by a optimum number of definition points lying on the object. Though there may be thousands of different adjacent points on a line between its two end points, this particular line segment can be uniquely expressed by the end points only. For example, in column matrix format, $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ may represent one and only one line between points (1, 2) and (3, 4) [refer Figure 4.2].

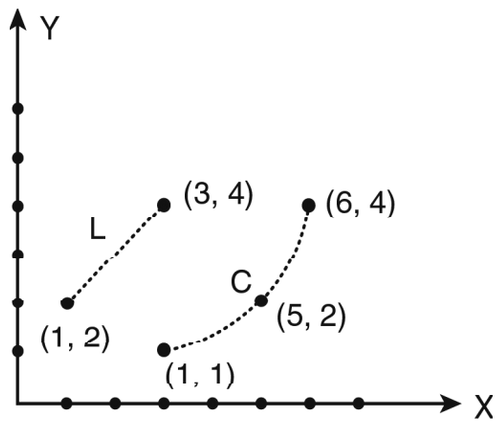


Fig. 4.2 *A closer look at the line will reveal that it is basically a series of*

adjacent points, maintaining the straight line connectivity between the definition points

(1, 2) and (3, 4). Similarly the arc segment 'C' is uniquely represented by

3 points (1, 1) – (5, 2) – (6, 4) in column-matrix format $C = \begin{pmatrix} 1 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix}$

NOTES

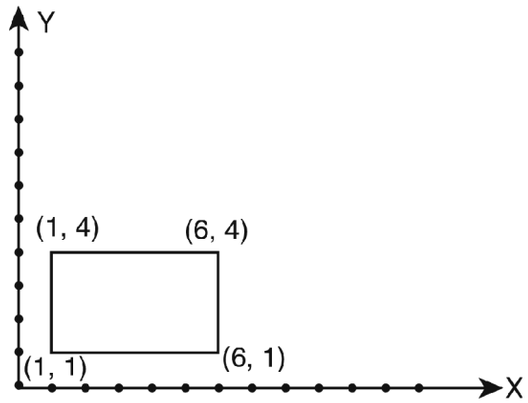


Fig. 4.3 *Representation of Rectangle in 2-D*

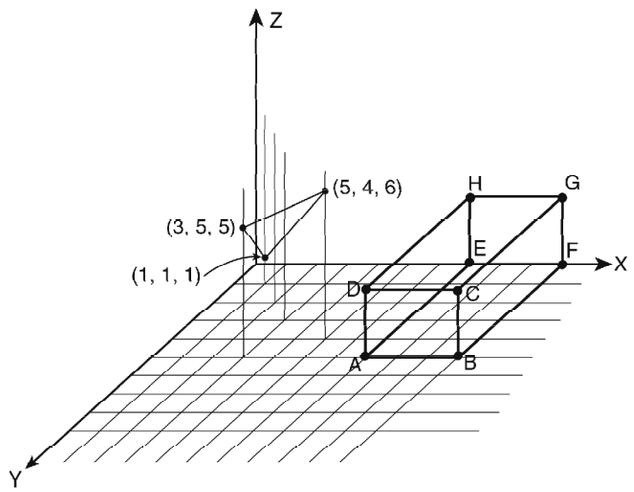


Fig. 4.4 *Representation of a Triangle and Rectangular Parallelopiped*

A triangle having vertices at (1, 1, 1), (3, 5, 5) and (5, 4, 6) in 3-D space (refer Figure 4.4) can be represented uniquely by

$$\begin{pmatrix} 1 & 3 & 5 \\ 1 & 5 & 4 \\ 1 & 5 & 6 \end{pmatrix}$$

A 5 × 3 2-D rectangle having its lower left vertex at (1, 1) in Figure 4.3 can be represented by

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

The rectangular parallelopiped *ABCDEFGH* in 3-D space (refer Figure 4.4) may be defined by the matrix

NOTES

$$\begin{pmatrix} 7 & 10 & 10 & 7 & 7 & 10 & 10 & 7 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \end{pmatrix}$$
$$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$$
$$A \quad B \quad C \quad D \quad E \quad F \quad G \quad H$$

Now in the following sections, we will see how the coordinates of these object definition points changes to represent the transformed object.

4.3 TRANSFORMATIONS AND MATRIX

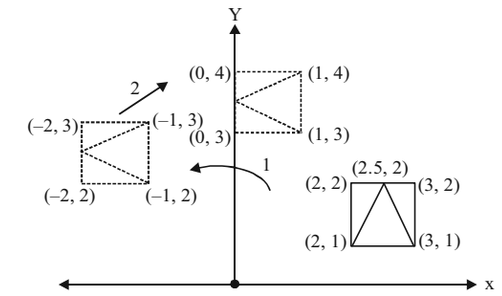
Since we are now aware of representing all kinds of transformations in a homogeneous manner, we can reduce a long sequence of transformations to a series of matrix multiplications, with little difficulty. No matrix addition is required. If $[T_1]$, $[T_2]$, $[T_3]$ be any three-transformation matrices, then the matrix product, $[T_1]\,[T_2]\,[T_3] = ([T_1]\,[T_2])\,[T_3] = [T_1]\,([T_2]\,[T_3])$, which implies that we can evaluate matrix product using either a left to right or a right to left associative grouping. But we must be careful about the order in which transformations actually takes place and the order we follow while multiplying the corresponding transformation matrices. *For column-matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left***.** Different orders of multiplication of matrices will give different results, because we all know that matrix product may not be commutative always, i.e., $[T_1]\,[T_2] \neq [T_2]\,[T_1]$.

For example, if we first rotate an object (counterclockwise 90° about origin) and then translate the rotated object (by 2,1) the representative matrix expression is:

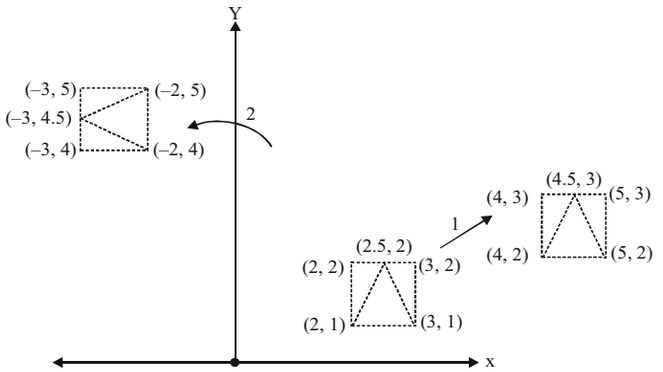
$$[X'] = [T_T]_{2,1} \quad [T_R]_{90^\circ} \quad [X]$$
$$= \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 3 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

(Here $[X]$ represents the rectangle shown in Figure 3.21(a).

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



(a) First Rotation and then Translation



(b) First Translation and then Rotation

Fig. 4.5 *Sequence of Rotation and Transition*

Now, if we reverse the order of multiplication of $[T_T]_{2,1}$ and $[T_R]_{90^\circ}$ implying first translation and then rotation, then:

$$[X'] = [T_R]_{90^\circ} \quad [T_T]_{2,1} \quad [X]$$

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

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

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

The transformed object matrix $[X']$ is different in the above two cases.

However, *multiplication of transformation matrices is commutative for a sequence of transformations, which are of the same kind.*

As an example, two successive rotations could be performed in either order and the final result would be the same. If the successive rotations are θ_1 and θ_2 about the origin to finally transform the point P to P', then:

NOTES

$$P' = [T_R]_{\theta_2} \quad [T_R]_{\theta 1} \quad P$$

$$= \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= \begin{pmatrix} (\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2) & -(\sin\theta_1\cos\theta_2 + \cos\theta_1\sin\theta_2) & 0 \\ (\cos\theta_1\sin\theta_2 + \sin\theta_1\cos\theta_2) & (-\sin\theta_1\sin\theta_2 + \cos\theta_1\cos\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= \begin{pmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= [T_R]_{\theta_1} \quad [T_R]_{\theta_2} \quad P = [T_R]_{\theta_1+\theta_2} \quad P$$

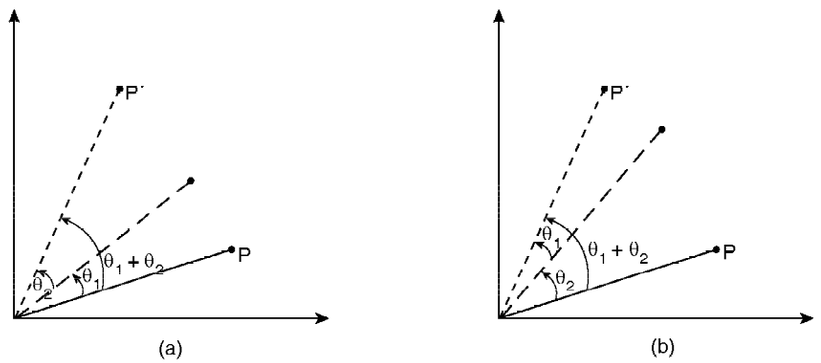


Fig. 4.6 *Successive Rotation: First by θ_1 and then by θ_2First θ_2, Rotation followed by θ_1 Rotation; Result Same*

The commutative property also holds true for successive translations or scalings or successive reflections about coordinate axes (excluding the diagonals).

For two successive translations of $(\Delta x_1, \Delta y_1)$ and $(\Delta x_2, \Delta y_2)$

$$[T_T]_{\Delta x2,\ \Delta y2}\ [T_T]_{\Delta x1,\Delta y1} = [T_T]_{\Delta x1,\ \Delta y1}\,[T_T]_{\Delta x2,\Delta y2}$$

$$= \begin{pmatrix} 1 & 0 & \Delta x_1 \\ 0 & 1 & \Delta y_1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x_2 \\ 0 & 1 & \Delta y_2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & \Delta x_1 + \Delta x_1 \\ 0 & 1 & \Delta y_1 + \Delta y_2 \\ 0 & 0 & 1 \end{pmatrix} = [T_T]_{\Delta x1+\Delta x2,\ \Delta y1+\Delta y2}$$

For two successive scalings by (S_{x_1}, S_{y_1}) and (S_{x_2}, S_{y_2}),

$$[T_s]_{sx2,\ sy2} \quad [T_s]_{sx1,\ sy1} = \begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

$$= [T_s]_{s_{x1},\ s_{y1}} \quad [T_s]_{sx2,\ sy2}$$

$$= [T_s]_{(sx1,\ sx2),\ (sy1,\ sy2)}$$

NOTES

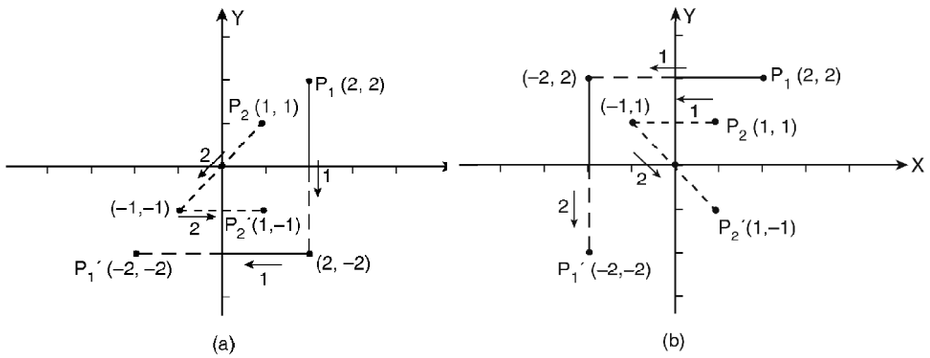


Fig. 4.7 *Two Commutative Pair of Reflection Transformation*
$P_1 \rightarrow P'_1$*: about X axis and Y axis,* $P_2 \rightarrow P'_2$*: about Origin and Y axis.*

Operation sequences are shown by arrows and numbers.

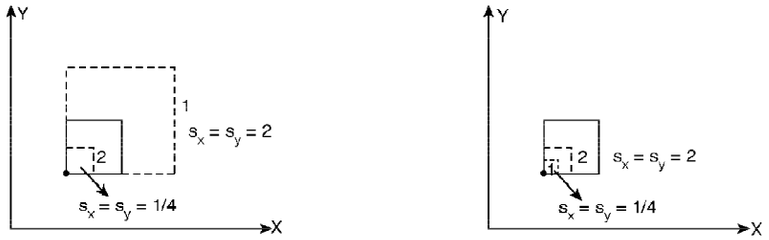
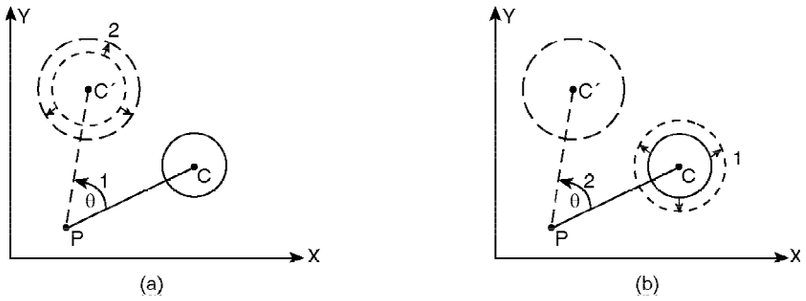


Fig. 4.8 *Commutative Pair of Fixed Point Scaling; Irrespective of the Order of Application of Scale Factors 2 and ¼ the Final Size of the Square is 2 × ¼, i.e., Half the Size of the Original Square.*

One more example of commutative pair of operations is rotation and uniform scaling.



First rotation of circle about P, by scaling of circle w.r.t. its centre

First scaling about centre, C followed then rotation about P; the result is same

Fig. 4.9 *Rotation and Scaling of a Circle*

Inverse Transformation

For each geometric transformation, there exists an inverse transformation which describes just the opposite operation of that performed by the original transformation. Any transformation followed by its inverse transformation keeps an object unchanged in position, orientation, size and shape. For example if an object is translated 3 units in the (+)ve X direction and then 3 units in the (–)ve X direction, the object comes back to its initial position implying no resultant

NOTES

transformation. We will use inverse transformation to nullify the effects of already applied transformation.

We will designate inverse of any transformation $[T]$ as $[T]^{-1}$. Listed below are the notations of inverse transformation matrices of some standard transformations.

Translation : $[T_T]^{-1}{}_{\Delta x, \Delta y} = [T_T]_{-\Delta x, -\Delta y}$

Rotation : $[T_R]^{-1}{}_{\theta} = [T_R]_{-\theta}$

Scaling : $[T_S]^{-1}{}_{Sx,\, Sy} = [T_S]_{1/Sx,\, 1/Sy}$

Reflection : $[T_M]^{-1}{}_{y=0} = [T_M]_{y=0}$ and $[T]^{-1}{}_{x=0} = [T_M]_{x=0}$

In the following sections, we will derive the transformation matrices for general pivot point rotation, general fixed point scaling and general reflection (about any line), using the basic matrices for pure transformation (such as rotation and scaling with respect to coordinate origin and reflection about coordinate axes) and translation. The technique here is based on the concept of composite transformation and inverse transformation.

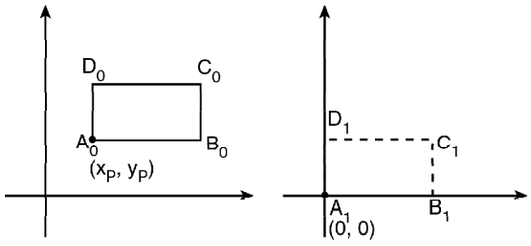
General Pivot Point Rotation

Apart from solving a problem from the most basic theory another, alternative approach is to derive the solution with reference to the familiar solution of a simpler problem of its kind. The standard solution (transformation matrix) of rotation about origin is much simpler and easy to remember compared to that of rotation about an arbitrary point other than origin. Hence, in an alternative approach, we will convert our present problem to a simple problem of rotation about the origin with the help of some additional translations. The sequential steps to be performed in this regard are as follows:

(*a*) Translate the pivot point (x_P, y_P) and the object by the same amount such that the pivot point moves to the coordinate origin, while the relative distances between the pivot point and the object points remain unchanged.

(*b*) Rotate the object about the origin (by θ).

(*c*) Translate the rotated object by an amount such that the pivot point comes back to its original position.



(a) Rectangle $A_0\ B_0\ C_0\ D_0$ to be rotated about pivot-pt. $A_0\ (x_P, y_P)$

(b) The rectangle & P.Pt. translated by $-x_P$, $-y_P$ so that A_0 becomes A_1 (0, 0) i.e. the origin

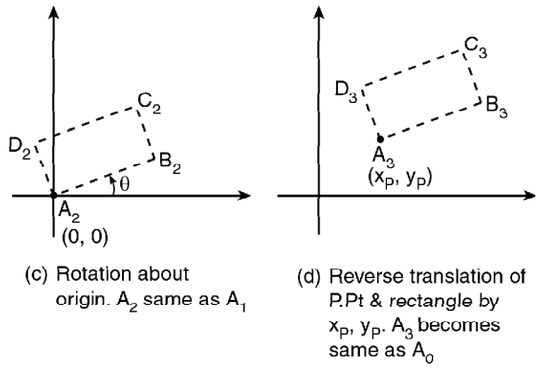


Fig. 4.10 *General Pivot Point Rotation*

Now combining the matrices corresponding to above transformations sequence-wise the resultant transformation matrix is: $[T_{COMB}] = [T_T]^{-1}_{-x_P,\,-y_P}[T_R]_\theta[T_T]_{-x_P,-y_P}$

$$= \begin{pmatrix} 1 & 0 & x_P \\ 0 & 1 & y_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_P \\ 0 & 1 & -y_P \\ 0 & 0 & 1 \end{pmatrix}$$

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

This is the same transformation matrix we obtained for general pivot point rotation in equation (4.15).

The same principle '*reducing a complex transformation problem to its simplest basic form at the cost of some additional to and fro translational or rotational movement*' is followed while deriving the transformation matrix for general fixed-point scaling or general reflection.

General Fixed Point Scaling

Here the problem is to scale an object with respect to a fixed point other than the origin. We will reduce this to the basic and simplest form of scaling with respect to origin through the following steps. (Refer Figure 4.11):

(*a*) Translate the object and the fixed point (x_f, y_f) equally so that the fixed point coincides with the coordinate origin.

(*b*) Scale the translated object with respect to the coordinate origin (with scale factors s_x, s_y).

(*c*) Use the inverse translation of step(a) to return the object and the fixed point to their original positions.

NOTES

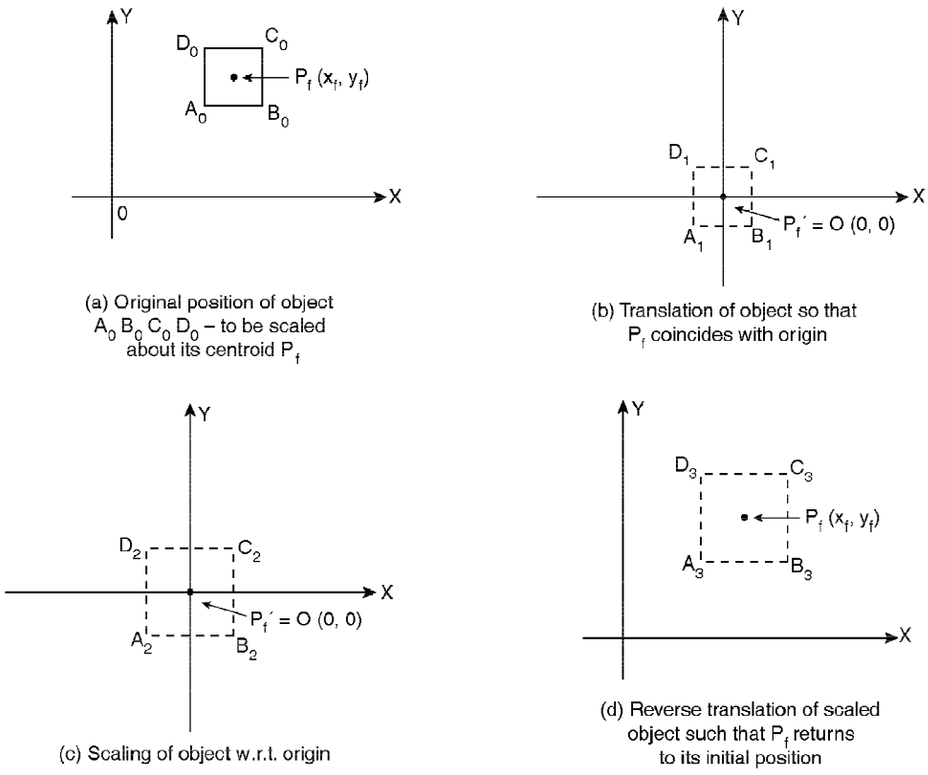


Fig. 4.11 *General Fixed Point Scaling*

The composite transformation matrix resulting from combination of transformations as per above sequence is given by

$$[T_{\text{Comb}}] = [T_T]^{-1}_{-x_f, -y_f} [T_s]_{s_x, s_y} \quad [T_T]_{-x_f, -y_f}$$

$$= \begin{pmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x & 0 & (1-s_x)x_f \\ 0 & sy & (1-s_y)y_f \\ 0 & 0 & 1 \end{pmatrix}$$

Compare this with equation 4.16.

Reflection through an Arbitrary Line

The problem of reflection of an object through a line that neither passes through the origin nor is parallel to the coordinate axes can be solved using the following steps. (Figure 4.12.)

(*a*) Translate the line and the object so that the line passes through the origin.

(*b*) Rotate the line and the object about the origin until the line is coincident with one of the coordinate axes about which we are familiar to perform reflection.

(*c*) Reflect the object about that coordinate axis.

(*d*) To the objects, apply the inverse rotation about the origin.

(*e*) Translate the object back to the original location.

In matrix notation $[T_{COMB}] = [T_T]\,[T_R]\,[T_M]\,[T_R]^{-1}\,[T_T]^{-1}$

NOTES

If the arbitrary mirror line is given by $y = mx + c$ where m = slope of the line = tan θ, $\theta = \text{zan}^{-1}(m)$ being the angle the line makes with the X axis, and c = intercept on the Y axis made by the line [implying $(0, c)$ is a point on the line].

Then the amount of translation in step (*a*) should be $(0, -c)$, whereas the amount of rotation in step (*b*) should be $-\theta$, i.e., $-\tan^{-1}(m)$ about the origin if we prefer to merge the line with the X axis.

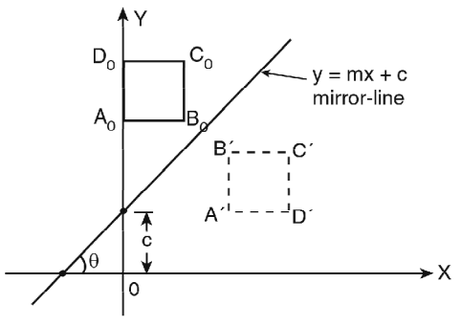
So we can say:

$$[T_{\text{COMB}}] = [T_T]^{-1}_{0,-c}\,[T_R]^{-1}_{-\theta}\,[T_M]_{y=0}\,[T_R]_{\theta}\,[T_T]_{0,-c}$$

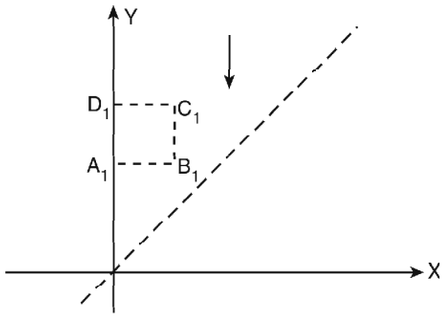
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -c \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} (\cos^2\theta - \sin^2\theta) & 2\sin\theta\cos\theta & -2c\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & (\sin^2\theta - \cos^2\theta) & c(\cos^2\theta - \sin^2\theta) + c \\ 0 & 0 & 1 \end{pmatrix} \qquad (4.1)$$

Put $\theta = \tan^{-1}(m)$



$A_0\,B_0\,C_0\,D_0$: original object
$A'\,B'\,C'\,D'$: object after reflection about y = mx + c



(a) After translation by $\Delta y = c$

NOTES

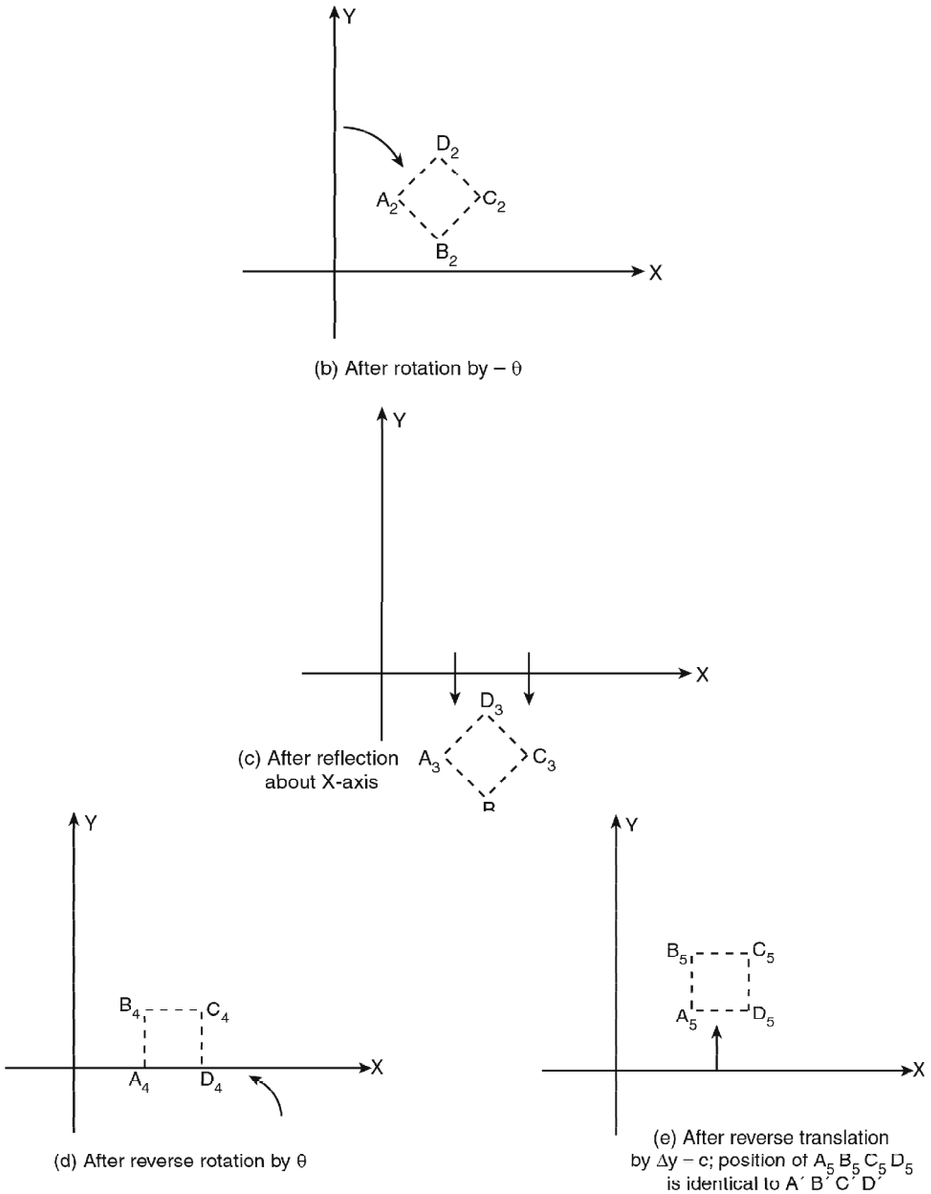


Fig. 4.12 *Reflection through an Artillery Line*

Note that equation (4.1) does not hold good for a mirror line parallel to the Y axis, say, $x = k$ straight line where $K \neq 0$. The reason is simple; we assumed that the mirror line $y = mx + c$ makes a finite intercept 'c' on the Y axis, but the line $x = k$ does not intersect the Y axis at all. For such cases, we will first translate the object and mirror line $-k$ units in X direction to merge the mirror line with Y axis, then reflect the object about Y axis and finally translate the reflected object k units in X direction.

Accordingly, $[T_M]_{x=k} = [T_T]^{-1}_{-k,0}\,[T_M]_{x=0}\,[T_T]_{-k,0}$

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

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

4.3.1 2-D Rotation

NOTES

Rotation is used to rotate objects about any point in a reference frame. Unlike translation, rotation brings about changes in position as well as orientation. The point about which the object is rotated is called the *pivot point* or *rotation point*. Conventionally, anticlockwise rotation about the pivot point is represented by positive

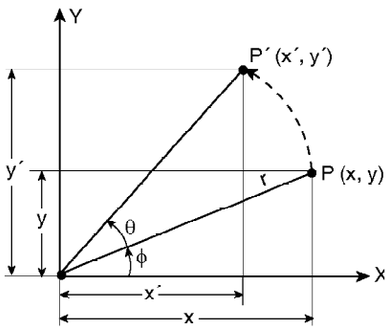


Fig. 4.13 Rotation of a Point about the Origin

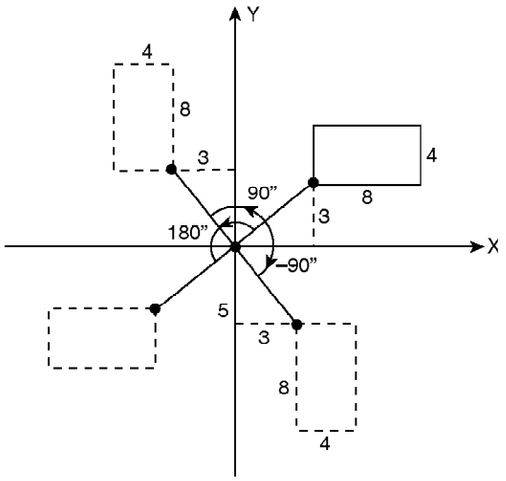


Fig. 4.14 Rotation of a Rectangle about the Origin

angular value. Such transformation can also be visualized as rotation about an axis that is perpendicular to the reference plane and passes through the pivot point.

Rotation about Origin

Consider a trial case where the pivot point is the origin as shown in Figure 4.13. Then the point to be rotated $P(x, y)$ can be represented as:

$$x = r\cos\phi \quad y = r\sin\phi$$

where (r, θ) is the polar coordinate of P. When this point P is rotated through an angle θ in anticlockwise direction, the new point $P'(x', y')$ becomes:

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

Rewriting the above equations using laws of sines and cosines from trigonometry,

$$\left.\begin{aligned} x' &= r\ \cos\theta\ \cos\phi - r\sin\theta\sin\phi \\ y' &= r\ \sin\theta\ \cos\phi + r\cos\theta\sin\phi \end{aligned}\right\} \tag{4.2}$$

Replacing $r\cos\phi$ and $r\ \sin\phi$ with x and y respectively in (4.2) we get the simplified form:

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

In matrix rotation the above relation can be more compactly expressed as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.3}$$

Symbolically, $[X] = [T_R]\ [X]$, where $[T_R]$ is the transformation matrix for rotation.

NOTES

Rotation about an Arbitrary Pivot Point

Equation 4.3 is applicable only for rotation about the origin. But in many applications, the pivot point will not be the origin. It may be any point lying on the object(s) to be rotated or any point outside the object simply anywhere in the same 2D plane. For example, consider the cases when we want to rotate any straight-line about one of its end points or any rectangle about one of the corner points or any object lying on a circle about its centre.

In Figure 4.15, the pivot point is an arbitrary point P_P having coordinates (x_P, y_P). After rotating $P(x, y)$ through a positive θ angle, its new location is $x'y'$ (P').

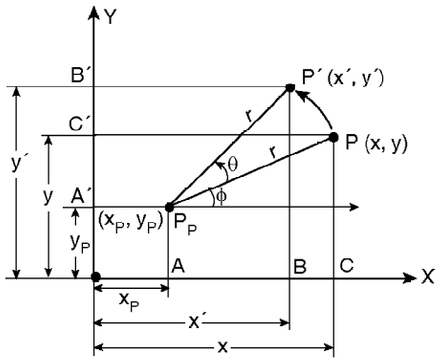


Fig. 4.15 *CCW Rotation of Point P about* P_P

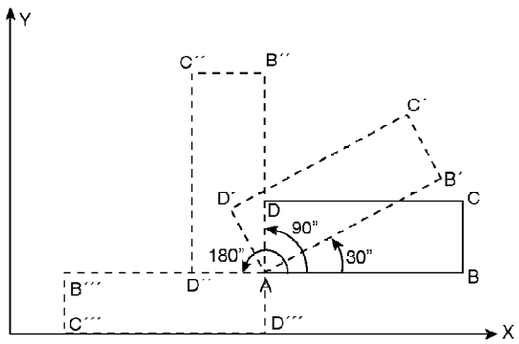


Fig. 4.16 *CCW Rotation of Rectangle ABCD about one of its corner Point A*

Here $x' = OB$

$= OA + AB$

$= x_P + r\cos(\theta + \phi)$

$= x_P + r\cos\phi\cos\theta - r\sin\phi\sin\theta$

$= x_P + (x - x_P)\cos\theta - (y - y_P)\sin\theta$

$\because \quad r\cos\varphi = AC$

$= OC - OA$

$= x - x_P$

and $\quad r\sin\varphi = A'C'$

$= OC' - OA'$

$= y - y_P$

Also, $y' = OB'$

$= OA' + A'B'$

$= y_P + r\sin(\theta + \varphi)$

$= y_P + r\cos\varphi\sin\theta + r\sin\varphi\cos\theta$

$= y_P + (x - x_P)\sin\theta + (y - y_P)\cos\theta$

Now,
$$\left.\begin{aligned} x' &= x_P + (x - x_P)\cos\theta - (y - y_P)\sin\theta \\ y' &= y_P + (x - x_P)\sin\theta - (y - y_P)\cos\theta \end{aligned}\right\} \quad (4.4)$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x - x_P \\ y - y_P \end{pmatrix} + \begin{pmatrix} x_P \\ y_P \end{pmatrix}$$

If we rearrange equation (4.4) by grouping the (x_P, y_P) and (x, y) related terms, we get:

$$x' = (x_P - x_P\cos\theta + y_P\sin\theta) + (x\cos\theta - y\sin\theta)$$
$$= \{x_P(1 - \cos\theta) + y_P\sin\theta\} + (x\cos\theta - y\sin\theta)$$

Similarly,

$$y' = \{(-x_P\sin\theta) + y_P(1 - \cos\theta)\} + (x\sin\theta + y\cos\theta)$$

This grouping allows us to express $x'\,y'$ matrix, in terms of x_P, y_P matrix and x, y matrix as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 - \cos\theta & \sin\theta \\ -\sin\theta & 1 - \cos\theta \end{pmatrix} \begin{pmatrix} x_P \\ y_P \end{pmatrix}$$

Symbolically $[X'] = [T_R]\ [X] + [T_P]$ (4.5)

Thus, we see that the general equation for rotation, i.e., equation (3.5) differs from equation (4.3) by an additive term $[T_P]$ involving pivot point coordinates. In a later section we will convert such an expression into a more convenient format $[X'] = [T]\ [X]$ involving no additive terms.

Evaluation of the determinant of the general rotation matrix:

$$[T_R] = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \text{ yields } \det\ [T_R] = \cos^2\theta + \sin^2\theta = 1$$

In general, transformations with a determinant identically equal to + 1 yield pure rotation.

In Figure 4.13, P is transformed to P' through a (+) ve θ rotation. If we wish to bring back P' to P, we have to apply an inverse transformation, i.e., a (–) ve θ rotation. According to equation (4.3) the required transformation matrix to obtain P from P' is:

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

If we symbolize the above matrix as $[T_R]^{inv}$, then we find

NOTES

$$[T_R]\ [T_R]^{inv} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$= \begin{pmatrix} \cos^2\theta + \sin^2\theta & \cos\theta\sin\theta - \sin\theta\cos\theta \\ \sin\theta\cos\theta - \cos\theta\sin\theta & \sin^2\theta + \cos^2\theta \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = [\mathrm{I}], \text{ where [I] is the identity matrix.}$$

This implies that the inverse rotation matrix $[T_R]^{inv}$ for pure rotation is identical to the inverse of the rotation matrix, i.e., $[T_R]^{-1}$

The above is true because from matrix properties we know that only $[T_R]$ $[T_R]^{-1} = [I]$.

The transpose of the rotation matrix $[T_R]$

$$\text{i.e., } [\mathrm{T_R}]^{\mathrm{T}} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = [\mathrm{T_R}]^{-1}$$

Thus, we can infer that *the inverse of any pure rotation matrix, i.e., one with a determinant equal to +1, is its transpose.*

4.3.2 Reflection

A reflection is a transformation that produces a mirror image of an object. In 2D reflection we consider any line in 2D plane as the mirror; the original object and the reflected object are both in the same plane of the mirror line. However we can visualize a 2D reflection as equivalent to a 3D rotation of 180º about the mirror line chosen. The rotation path being in the plane perpendicular to the plane of mirror line. Here we will study some standard 2D reflection cases characterized by the mirror line.

Reflection about X axis

The basic principles of reflection transformation

(*i*) The image of an object is formed on the side opposite to where the object is lying, with respect to the mirror line.

(*ii*) The perpendicular distance of the object (i.e., the object points) from the mirror line is identical to the distance of the reflected image (i.e., the corresponding image points) from the same mirror line. Once again, the two perpendiculars must be along the same straight line.

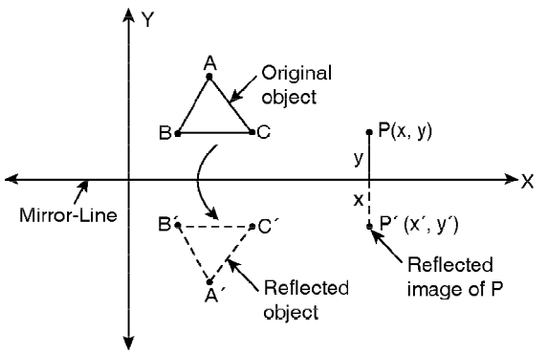


Fig. 4.17 Reflection about X axis

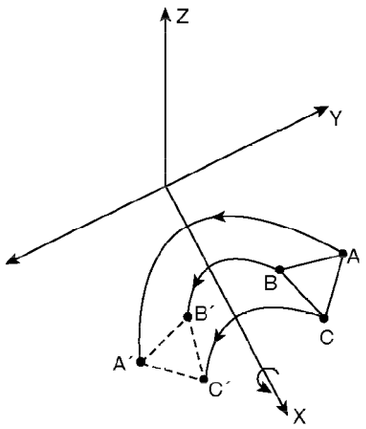


Fig. 4.18 *2D Reflection about X axis as 3D rotation(180^0) about X axis*

Therefore, the relation between the point $P(x, y)$ and its image $P'(x', y')$ about X axis is simply,

$$x' = x$$
$$y' = -y \quad \text{(Refer Figure 4.17)}$$

So, the transformation matrix for reflection about X axis or $y = 0$ axis is:

$[T_M]_{y=0} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ and the transformation is represented as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.6}$$

i.e. $[X'] = [T_M]_{y=0} [X]$

Consider the 2D reflection of the triangle *ABC* about *X* axis (Figure 4.17) forming the image $\Delta\ A'B'C'$. In the other way, we can interpret this event as a 3D rotation–we can think of the *ABC* triangle moving out of the *xy* plane and rotating 180º in 3D space about the X axis and back into the *xy* plane on the other side of the *X* axis.

Reflection about Y axis

A reflection about Y axis flips *x* coordinates while *y* coordinates remains the same. For reflection of $P(x, y)$ to $P'(x', y')$ in Figure 4.19,

$$x' = -x$$
$$y' = y$$

This transformation is identified by the reflection –transformation matrix.

$$[T_M]_{x=0} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.7}$$

The transformed new vertices $A'B'C'$ of triangle *ABC* are given by:

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 7 & 5 & 10 \\ 9 & 2 & 4 \end{pmatrix}=\begin{pmatrix} -7 & -5 & -10 \\ 9 & 2 & 4 \end{pmatrix}$$

NOTES

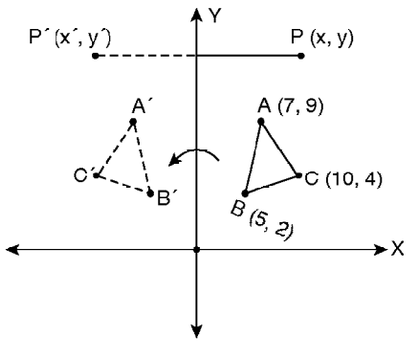


Fig. 4.19 *Reflection about Y axis*

Reflection about the Straight Line $y = x$

To find the relation between a point's coordinates (x, y) and those of its image (x', y'), reflected through $y = x$ straight line, refer to Figure 4.20.

Here $PK = P'K$ and both PK and $P'K$ are perpendicular to $y = x$ or OK.

From simple geometry, you will find $\Delta\ POK \cong \Delta\ P'OK$

$\therefore\ OP = OP'$ and $\angle\ POK = \angle\ P'OK$

Implying also

$\angle\ POM = \angle\ P'ON$ because $y = x$ straight line makes 45° angle with both the axes.

That, in turn, implies $\Delta POM \cong \Delta P'ON$

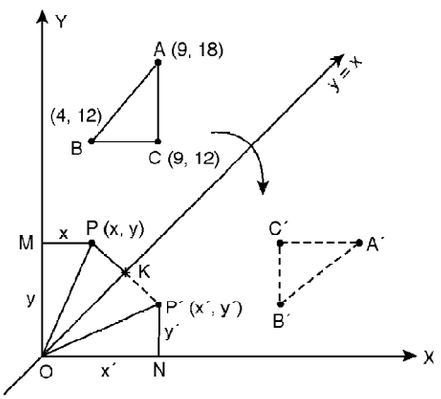


Fig. 4.20 *Reflection about the Straight Line y = x*

So, $PM = P'N$ and $OM = ON$

but $PM = x$, $P'N = y'$, $OM = y$, $ON = x'$

Hence $x' = y$

$y' = x$

This can be represented by:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (4.8)$$

where $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=x}$

Thus, the transformed vertices $A'B'C'$ of triangle ABC in Figure Similar to Figure 4.20. are given by:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 9 & 4 & 9 \\ 18 & 12 & 12 \end{pmatrix} = \begin{pmatrix} 18 & 12 & 12 \\ 9 & 4 & 9 \end{pmatrix}$$

Reflection about the Straight Line $y = -x$

Considering the experimental point $P(x, y)$, we draw a figure similar to Figure 4.20. We can prove:

$$\left.\begin{aligned} PM &= P'N \\ OM &= ON \end{aligned}\right\}$$

But this time we cannot simply state $x' = y$ and $y' = x$ from the above relation, because that is only the equation of magnitude. Looking at the figure, we find that P and its image P' both being in the 4th quadrant, x and x' are (+)ve, whereas y and y' are –ve. So, considering this fact, we should write:

$$x' = -y$$
$$y' = -x$$

This can be represented by,

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

where $\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=-x}$

Thus, we can infer that unlike in the case of reflection about diagonal axis $y = x$, in reflection about the other diagonal $y = -x$, the coordinate values are interchanged with their signs reversed.

Notice the changes of the vertices of triangle ABC to $A'B'C'$ in obtained the figure given by:

$$\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 4 & -2 & 4 \\ 6 & 4 & -2 \end{pmatrix} = \begin{pmatrix} -6 & -4 & 2 \\ -4 & 2 & -4 \end{pmatrix}$$

Reflection Relative to the Origin

In this case, we actually choose the mirror-line as the axis perpendicular to the xy plane and passing through the origin. After reflection both the x and y coordinates of the object point are flipped, i.e., x' becomes $-x$ and y' becomes $-y$**.** This can be easily proved from geometry as shown in Figure 4.21(b). The coordinate signs

NOTES

are just opposite because the image is always formed in the quadrant opposite to that of the object.

NOTES

Thus $x' = -x$

$y' = -y$, which can be represented in matrix form as:

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

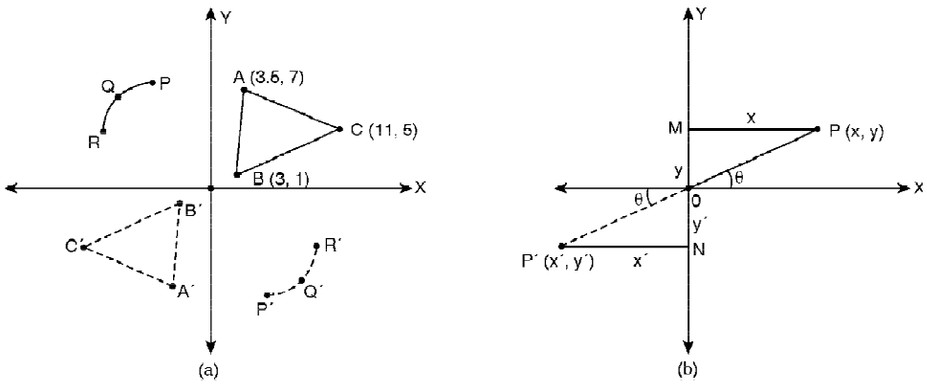


Fig. 4.21 *Reflection Relative to the Origin*

where

$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=x=0}$

The change of the triangle ABC to $A'B'C'$ in Figure 3.16(a) is given by,

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 3.5 & 3 & 11 \\ 7 & 1 & 5 \end{pmatrix} = \begin{pmatrix} -3.5 & -3 & -11 \\ -7 & -1 & -5 \end{pmatrix}$$

Similarly the curve PQR is changed to $P'Q'R'$ represented by,

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} -5 & -8 & -9 \\ 8 & 7 & 4 \end{pmatrix} = \begin{pmatrix} 5 & 8 & 9 \\ -8 & -7 & -4 \end{pmatrix}$$

Notice that the radial distances of all the object points remain unaltered before and after reflection.

In the previous sections we have discussed reflection transformation about axes or lines passing through origin. The questions, which now comes to mind, is how to find the transformation equations for reflections about any arbitrary line $y = mx + c$ in 2D xy plane. We cannot derive those equations as easily as we did for rotation about any arbitrary pivot point or scaling about any arbitrary fixed point. However, we can accomplish such reflections with a combination of basic translate–rotate–reflect transformations, which may be easier for you to understand after we establish a convenient method of combining consecutive basic transformations on an object.

Before that, it is important to note the fact that all the standard reflection transformations discussed so far could have been achieved, alternatively, by scaling with appropriate choice of scale factors (+ve, or –ve). In that sense, reflection is not a basic transformation. But, interestingly, all these reflection matrices have a determinant equal to -1.

In general, if the determinant of a transformation matrix is identically –1, then the transformation produces a pure reflection.

4.3.3 Scaling

Scaling is a transformation that changes the size or shape of an object. Scaling with respect to origin can be carried out by multiplying the coordinate values (x, y) of each vertex of a polygon, or each endpoint of a line or arc or the center point and peripheral definition points of closed curves like a circle by scaling factors s_x and s_y respectively to produce the coordinates $(x' y')$.

The mathematical expression for pure scaling is:

$$\left.\begin{aligned} x' &= s_x \cdot x \\ y' &= s_y \cdot y \end{aligned}\right\} \Rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{4.9}$$

or symbolically, $[X'] = [T_S]\,[X]$

s_x expands or shrinks object dimensions along X direction, whereas s_y affects dimensions along Y direction.

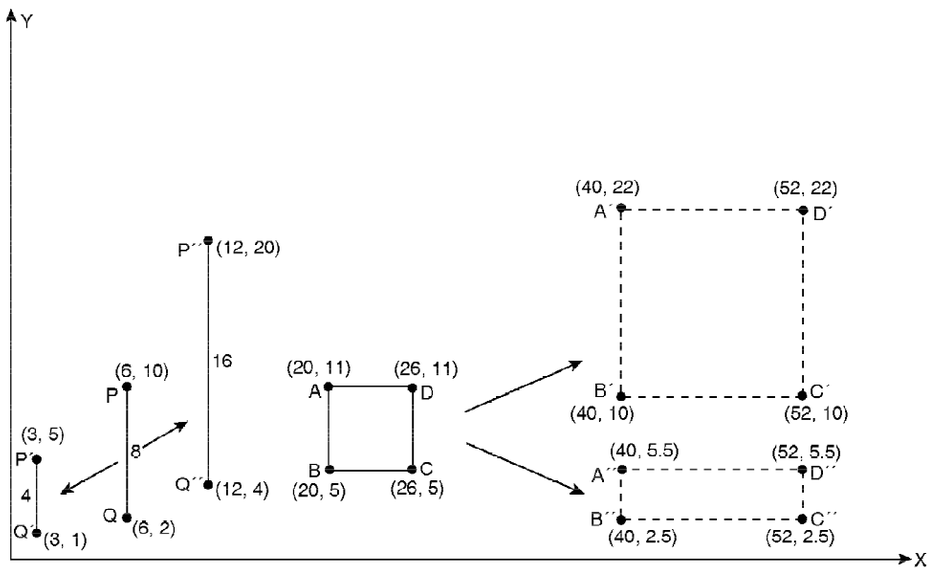


Fig. 4.22 *Line PQ scaled to P′Q′ ($s_x = s_y = 0.5$) and to P′′Q′′ ($s_x = s_y = 2$). Square ABCD (each side 6 units) uniformly scaled ($s_x = s_y = 2$) producing a bigger square A′B′C′D′ with each side 12 units. But with non-uniform scaling ($s_x = 2$, $s_y = 0.5$) the square changes to a 12 by 3 rectangle A′′B′′C′′D′′.*

We can represent the scaling transformation carried out on square *ABCD* in Figure 4.22 as:

NOTES

$$[A'\,B'\,C'\,D'] = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} [A\,B \quad CD]$$

$$\text{and } [A''\,B''\,C''\,D''] = \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix} [A\,B\;C\,D]$$

Notice that in the second case where $s_x \neq s_y$, a distortion in shape has occurred and the square has transformed into a rectangle.

In general, for *uniform scaling*, if $s_x = s_y > 1$, a uniform expansion occurs; i.e., the object becomes larger. If $s_x = s_y < 1$, then a uniform compression occurs, i.e., the object gets smaller. Non-uniform expansions or compressions occur, depending on whether s_x and s_y are individually > 1 or < 1 but unequal. Such scaling is also known as *differential scaling*. While the basic object shape remains unaltered in uniform scaling, both the shape and the size change in differential scaling. Another interesting point to be noted is that there is always a translation associated with scaling. Look at the figures, irrespective of the nature of scaling and the scale factors, the scaled objects have substantially moved from their respective original positions. This is easily understood if we recall that during scaling transformation, the position vectors of the definition points are actually scaled with respect to the origin. For example, consider the position vector of point $Q(6, 2)$ of line PQ. Its magnitude with respect to origin is $\sqrt{6^2 + 2^2} = 2\sqrt{10}$. The magnitudes of the position vectors of the scaled points $Q'(3, 1)$, $Q''(12, 4)$ are $\sqrt{10}$ and $4\sqrt{10}$ respectively implying uniform scaling with scale factors 0.5 and 2 respectively. Also note that the direction of the position vectors of the original point $\left(\tan^{-1}\frac{2}{6}\right)$ and the scaled points $\left(\tan^{-1}\frac{1}{3} \text{ and } \tan^{-1}\frac{4}{12}\right)$ remains same. Thus, quite obviously, *pure uniform scaling with factors < 1 moves objects closer to the origin, while factors > 1 moves objects farther from origin, at the same time decreasing or increasing the object size.*

Scaling with Respect to any Arbitrary Point

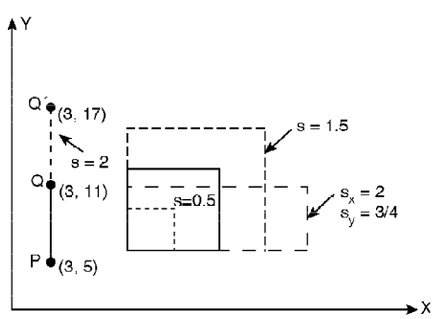


Fig. 4.23 Scaling Transformation of PQ to PR

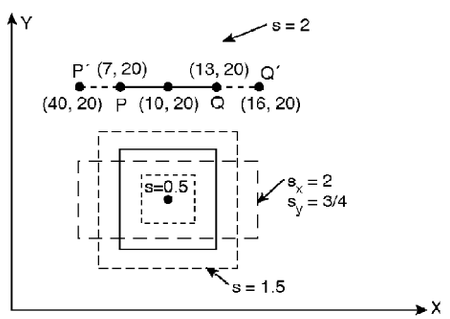


Fig. 4.24 Scaling of aRrectangle

By scaling an object, we generally expect it to grow or shrink in size or shape based on its original position (as shown in Figure 4.23 and Figure 4.24). Apart from scaling, the movement of the object (which is intrinsically associated in scaling) with respect to origin is mostly unwanted and can be eliminated by scaling the

object with respect to a point conveniently chosen on the object itself. The point so chosen, called the *fixed point*, remains unaltered in position after the scaling transformation when the scale factors are applied on the objects dimensions relative to that fixed point. As a result, the object seems to expand or shrink in size or change in shape without any displacement of the object as a whole.

For example, consider the scaling transformation of PQ to PQ' (not $P'Q'$) in Figure 4.23. Here P is considered as the fixed point. So, instead of multiplying the scale factor ($s = 2$) to both P and Q position vectors, it is multiplied with $Q - P$ and then added with P to obtain shifted Q' and then the line PQ' is reconstructed.

$$\begin{aligned} Q' = P + s(Q - P) &= (3,5) + 2\{(3-3), (11-5)\} \\ &= (3,5) + (0,12) \\ &= (3, 17) \end{aligned}$$

Note that $Q - P$ is the dimension of the line PQ relative to P. Now compare the result with the scaling of line PQ as shown in Figure 4.22. Instead of endpoint P, if we consider the midpoint of PQ as the fixed point, it will expand symmetrically in both directions forming $P'Q'$ (Figure 4.25), the midpoint remaining unchanged in position.

For scaling of a rectangle shown in Figure 4.24, the fixed point is the centroid of the rectangle. Thus, the fixed point need not necessary lie on the object—it can be any point either on, within or outside the object. Here we derive the most generalized expression for scaling any object (P) coordinates say (x, y), with respect to any arbitrary point, say $P_f(x_f, y_f)$.

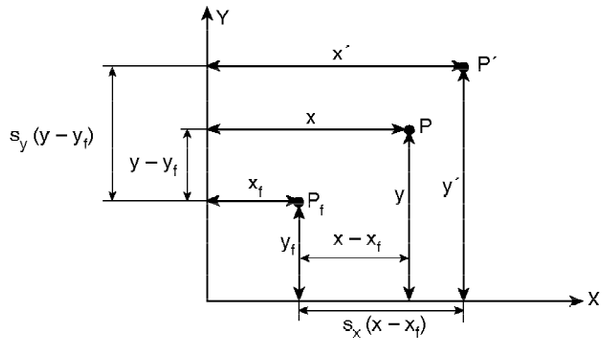


Fig. 4.25 *Scaling of a Point with Respect to a Fixed Point s_x, $s_y > 1$*

As in this case, the distance between the point in question $P(x, y)$ and the fixed point $P_f(x_f, y_f)$ is scaled, we can write:

$$x' = x_f + (x - x_f)s_x$$
$$y' = y_f + (y - y_f)s_y$$

where (x' y') are the scaled coordinates and s_x, s_y are the scale factors.

We can rewrite these transformation equations to separate the multiplicative and additive terms:

$$x' = s_x\, x + (1 - s_x)\, x_f$$
$$y' = s_y\, y + (1 - s_y)\, y_f$$

In matrix notation,

NOTES

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

Symbolically,

$$[X'] = [T_s]\,[X] + [T_f]^* \tag{4.10}$$

s_x and s_y can be equal or unequal and can be >1, <1, equal to 1 or even negative integer or fraction but never equal to zero.

Comparing Equation (4.10) with Equation (4.5), we find that coordinates for a *fixed point* feature in the scaling equations similar to the coordinates for a *pivot point* in the rotation equations.

4.3.4 Combined Transformations and Homogeneous Corrdinates

The shape, size, position and orientation of 2D objects can be controlled by performing matrix operation on the position vectors of the object definition points. In some cases, however, a desired orientation of an object may require more than one transformation to be applied successively. For example, let us consider a case which requires 90° rotation of a point $\begin{pmatrix} x \\ y \end{pmatrix}$ about origin followed by reflection through the line $y = x$.

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

This $\begin{pmatrix} x' \\ y' \end{pmatrix}$ then undergoes reflection to produce $\begin{pmatrix} x'' \\ y'' \end{pmatrix}$

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

These successive matrix operations can be symbolically expressed as:

$$[X''] = [T_M]_{y=x}\,[X']$$

i.e., $[X''] = [T_M]_{y=x}\ \{[T_R]_{90^\circ}\ [X]\}$

As we know matrix multiplication is associative, we can first perform $[T_M]_{y=x}\,[T_R]_{90^\circ}$ instead of performing $[T_R]_{90^\circ}\,[X]$ first (but not $[T_R]_{90^\circ}\,[T_M]_{y=x}$, thereby maintaining the right to left order of succession) to form a *resultant transformation matrix*. This matrix when multiplied with the original point coordinates will yield the ultimate transformed coordinates.

$$\text{Thus, } [T_M]_{y=x}\,[T_R]_{90^\circ} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$= [T_{COMB}]\ (say)$$

$$\text{Now, } [T_{COMB}][X] = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$$

Which shows the same result as before.

This process of calculating the product of matrices of a number of different transformations in a sequence is known as *concatenation* or, *combination* of transformations and the resultant product matrix is referred to as *composite or concatenated transformation matrix.* The application of concatenated transformation matrix on the object coordinates eliminates the calculation of intermediate coordinate values after each successive transformation.

But the real problem arises when there is a translation or rotation or scaling about an arbitrary point other than the origin involved among several successive transformations. The reason is that the general form of expression of such transformations is not simply, $[X'] = [T]\,[X]$ involving the 2 by 2 array $[T]$ containing multiplicative factors, rather it is in the form, $[X'] = [T_1]\,[X] + [T_2]$, where $[T_2]$ is the additional two-element column matrix containing the translational terms. Such transformations cannot be combined to form a single resultant representative matrix. This problem can be eliminated if we can combine $[T_1]$ and $[T_2]$ into a single transformation matrix. This can be done by expanding the usual 2×2 transformation matrix format into 3×3 form. The general 3×3 form will be something like:

$\begin{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} & \begin{matrix} m \\ n \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix}$ where the elements a, b, c, d of the upper left 2×2 sub matrix are the multiplicative factors of $[T_1]$ and m, n are the respective x, y translational factors of $[T_2]$.

But such 3×3 matrices are not conformable for multiplication with 2×1 2D position vector matrices. Herein lies the need to include a *dummy coordinate* to make 2×1 position vector matrix $\begin{pmatrix} x \\ y \end{pmatrix}$ to a 3×1 matrix $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ where the third coordinate is dummy.

Now if we multiply, $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ with a non-zero scalar 'h', then the matrix it forms is:

$\begin{pmatrix} x\,h \\ y\,h \\ h \end{pmatrix}$ or symbolically, say, $\begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$ which is known as the *homogeneous coordinates* or homogeneous position vector of the same point $\begin{pmatrix} x \\ y \end{pmatrix}$ in 2D plane.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix} = h \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The extra coordinate h is known as a weight, which is homogeneously applied to the cartesian components.

Thus, a general homogeneous coordinate representation of any point $P(x, y)$ is (x_h, y_h, h) or (xh, yh, h) that implies, $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$

NOTES

As 'h' can have any non-zero value, there can be infinite number of equivalent homogeneous representations of any given point in space. For example, (6, 4, 2), (12, 8, 4), (3, 2, 1), (1/2, 1/3, 1/6), (–3, –2, –1) all represent the physical point (3, 2).

But so far as geometric transformation is concerned, our choice is simply h = 1 and the corresponding homogeneous coordinate triple $(x, y, 1)$ for representation of point positions (x, y) in xy-plane. Other values of parameter 'h' are needed frequently in matrix formulation of three-dimensional viewing transformation. Though we have introduced homogeneous coordinates just as a tool for making our transformation operations easier, they have their own significance as the coordinates of points in projective space.

Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations uniformly as matrix multiplication. Coordinates are represented with three element column vectors and transformation operations are written in form of 3 by 3 matrices. Thus, for translation, we now have:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.11}$$

or,

Symbolically, $[X'] = [T_T][X]$

Compare equation with (4.17) both yield the same result, $x' = x + \Delta x$ and $y' = y + \Delta y$

Equations of rotation and scaling with respect to coordinate origin (derived earlier as (4.3) and (4.9) respectively) may be modified as,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.12}$$

and

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.13}$$

respectively.

Similarly, the modified general expression for reflection may be:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.14}$$

where values of a, b, c, d depend upon choice of coordinate axes or diagonal axes as mirror line.

The simplest $[X'] = [T]\,[X]$ form representing rotation about any arbitrary pivot point (x_P, y_P) as modified from equation (4.5) is:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_P + \sin\theta \cdot y_P \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta \cdot x_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.15}$$

and for scaling with respect to any arbitrary fixed point (x_f, y_f) the modified form of equation (4.10) is:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & (1-s_x)\cdot x_f \\ 0 & s_y & (1-s_y)\cdot y_f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{4.16}$$

It is now for you to do some matrix multiplication to compare the results of equation (4.15) and (4.16) with those of equations (4.5) and (4.10) respectively for complete satisfaction.

But before running to the next section, just pause for a few minutes. Have you noticed that in all the above 3 × 3 transformation matrices the bottom corner element on the diagonal line is always 1 as the bottom row is always [0 0 1]? Now what happens if we force this corner element to be anything other than 1, say 's'? The effect is interesting. If the diagonal becomes [1 1 s] and all the other elements are zero, i.e., if

$$[T] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix}$$

$$\text{then } [T][X] = [T] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ s \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ h \end{pmatrix} = [X']$$

Here $x' = x$, $y' = y$, $h = s$. Normalizing this yields the transformed coordinates as x/s and y/s which implies that overall scaling has occurred, s being the scale factor.

If $s < 1$, then an uniform expansion occurs whereas, if $s > 1$ an uniform compression occurs.

4.3.5 Translation

Let us think of a point P in a 2D plane. Assume also that the current position or location of P is depicted by its coordinate (x, y) with respect to a reference frame.

Now if we force P to move Δx distance horizontally and at the same time Δy distance vertically, then the changed location of P becomes $(x + \Delta x, y + \Delta y)$.

NOTES

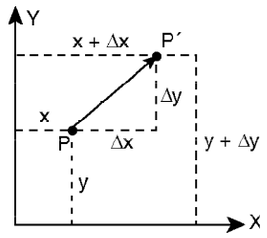


Fig. 4.26 *Translation of a Point*

In the terms of object transformation, we can say that the original point object $P(x, y)$ has been translated to become P' $(x' y')$ and amount of translation applied is the vector $\overrightarrow{PP'}$, where $|\overrightarrow{PP'}| = \sqrt{(\Delta x)^2 + (\Delta y)^2}$

Vectorially, we can express this transformation as: $P' = P + \overrightarrow{PP'}$

Algebraically, $x' = x + \Delta x$

$y' = y + \Delta y$

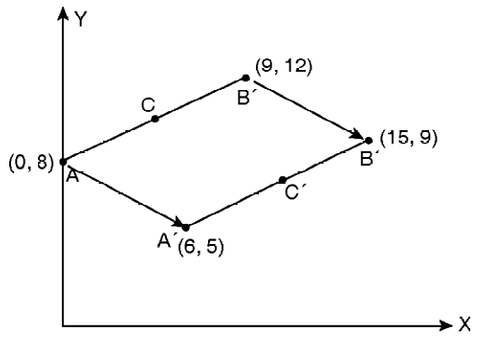


Fig. 4.27 *Translation of a Line*

In matrix formulation the above relation can be more compactly expressed as:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{4.17}$$

$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$ is called *translation vector* or *shift vector*.

In general, the above Equation (4.17) may be expressed as $[X'] = [X] + [T_T]$, where $[X']$ is the transformed object matrix, $[X]$ is the original object matrix and $[T_T]$ is the transformation (translation) matrix.

Consider the line (shown in Figure 4.27) with endpoints A (0,8) and B (9,12). If we have to move this line AB to $A'B'$, we have to apply equal translation to each of the end points A and B and then redraw the line between the new end points deleting the old line AB. The actual operation of drawing a line between two end points depends on the display device used and the draw–algorithm followed. Here, we consider only the mathematical operations on the position vectors of the end points.

Here, A (0, 8) becomes A' (6, 5), implying $\Delta x = 6,\ \Delta y = -3$

B (9,12) becomes B' (15, 9), implying $\Delta x = 6,\ \Delta y = -3$

So we can say, $\begin{pmatrix}6\\5\end{pmatrix}=\begin{pmatrix}0\\8\end{pmatrix}+\begin{pmatrix}6\\-3\end{pmatrix}$ and $\begin{pmatrix}15\\9\end{pmatrix}=\begin{pmatrix}9\\12\end{pmatrix}+\begin{pmatrix}6\\-3\end{pmatrix}$

Combining these two $\Rightarrow \begin{pmatrix}6 & 15\\5 & 9\end{pmatrix}=\begin{pmatrix}0 & 9\\8 & 12\end{pmatrix}+\begin{pmatrix}6 & 6\\-3 & -3\end{pmatrix}$

It is to be noted that *whatever amount of translation we apply to a straight line, the length and orientation (slope) of the translated line remain the same as those of the original line.*

It is implied from the Figure 4.27 that this $\begin{pmatrix}6\\-3\end{pmatrix}$ translation matrix is theoretically applied to (i.e., added to) all the points forming the line *AB*. This can be tested with any intermediate point *AB* between *A* & *B*. Think of the midpoint. Before transformation, it is:

$$C=\left(\frac{9+0}{2},\frac{12+8}{2}\right)=(4.5,\ 10)$$

After transformation, it is:

$$C'=\left(\frac{6+15}{2},\frac{5+9}{2}\right)=(10.5,\ 7)$$

So, $\Delta x = 10.5 - 4.5 = 6$

$\Delta y = 7 - 10 = -3$

And this is the reason why we can express transformations in terms of any constituent point of the object concerned.

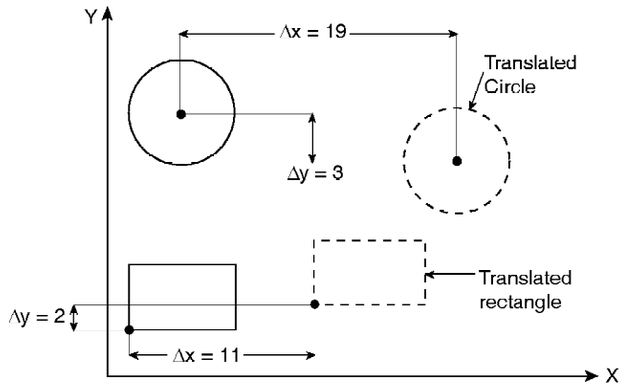


Fig. 4.28 *A Translated Rectangle's Point*

In Figure 4.28, coordinates are changed by $\begin{pmatrix}11\\2\end{pmatrix}$ with respect to the corresponding point-coordinates of the original rectangle. Similarly, the circle is displaced by $\begin{pmatrix}19\\-3\end{pmatrix}$.

For changing the position of a circle or ellipse, we translate the centre coordinates and redraw the figure in the new location.

Note from Figures 4.27 and 4.28 that we are transforming the objects without distorting the original shape, size and orientation.

NOTES

Thus, we can define *translation as a rigid body transformation that moves objects without deformation. Every point on the object is translated by the same amount and there exists a one-to-one correspondence between the transformed points and original points.*

Check Your Progress

1. What is transformation between the coordinate systems?
2. What is the point?
3. How will you define inverse transformation?
4. Define the pivot or rotation point.
5. Write the basic principles of reflection.
6. Define the term scaling.

4.4 3-D TRANSFPORMATION

Geometric transformations are defined as changing of position for given image or graphics. Literal meaning of transformation is to alter the position of an object on a plane. The transformation is explained in triangle. The Euclidean transformations are the most commonly used transformations. A Euclidean transformation is a translation, a rotation, or a reflection. Rotations in space are more complex, because we can either rotate about the *X*-axis, the *Y*-axis or the *Z*-axis. When rotating about the *Z*-axis only coordinates of *X* and *Y* will change and the *Z*-coordinate will be the same. In effect, it is exactly the rotation about the origin in the *XY*-plane. Scaling transformations stretch or shrink a given coordinate system and as a result change lengths and angles. So, scaling is not a Euclidean transformation. The effect of a shear transformation refers to pushing a geometric object in a direction that is parallel to a coordinate plane (3-D) or a coordinate axis (2-D). Transformations do not alter the degree of a polynomial, parallel lines or planes are transformed to parallel lines/planes, and intersecting lines or plane are transformed to intersecting lines and planes. However, affine transformations do not preserve lengths and angle measures and as a result they will change the shape of a geometric object. Projective transformation can bring finite points to infinity and points at infinity to finite range. The geometrical transformation refers to various processes, such as translation, scaling, rotation, reflection and shearing. Translation means the object can be moved from one plane to another plane. Translation is different from rotation and reflection. Rotation means object will turn around. Basically every rotation has centre and the angle of rotation. In the rotation there is no change in shape and size but there is change in the position. Reflection means object produce the mirror image of the object. Generally every reflection has the mirror line. The concept of translation, scaling, rotation, reflection and shearing is explained below.

4.4.1 Translation

If any point $P(x, y, z)$ in 3-D space is moved to position $P'(x', y', z')$ such that (refer Figure 4.28),

$$\begin{aligned} x' &= x + \Delta x \\ y' &= y + \Delta y \\ z' &= z + \Delta z \end{aligned} \qquad \text{...(4.18)}$$

NOTES

Δx, Δy, Δz being the displacement of P in three principal directions x, y, z respectively.

Then we can express this 3-D translation in homogeneous matrix form as

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

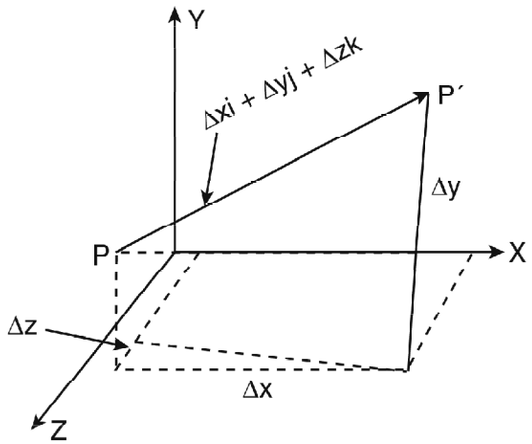


Fig. 4.29 *Point P in 3-D Space*

4.4.2 Scaling

The matrix expression for scaling transformation, relative to the coordinate origin, will be,

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

$$\Rightarrow \left.\begin{aligned} x' &= s_x \; x \\ y' &= s_y \; y \\ z' &= s_z \; z \end{aligned}\right\} \quad \begin{aligned} &\text{where } s_x, s_y, s_z \text{ are scale factors in} \\ &x, y \text{ and } z \text{ directions respectively.} \end{aligned} \qquad \text{...(4.19)}$$

But as we already know this type of scaling repositions an object relative to origin and also changes the shape (i.e., relative dimensions of the object) if the scaling parameters (s_x, s_y, s_z) are not equal.

To induce scaling without allowing any shape change or position change, we have to perform scaling with respect to a point lying on the object itself, and that too, using uniform scale factors ($s_x = s_y = s_z$).

NOTES

Applying methods similar to that used in 2-D the transformation matrix for scaling with respect to a selected fixed point (x_f, y_f, z_f) can be obtained as,

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

4.4.3 Rotation

We have seen that any 2-D rotation transformation is uniquely defined by specifying a centre of rotation and amount of angular rotation. But these two parameters do not uniquely define a rotation in 3-D space because an object can rotate along different circular paths centering a given rotation centre and thus forming different planes of rotation. We need to fix the plane of rotation and that is done by specifying an axis of rotation instead of a centre of rotation (refer Figure 4.30). The radius of rotation-path is always perpendicular to the axis of rotation. Before considering three-dimensional rotation about an arbitrary axis we examine rotation about each of the coordinate axes. By convention positive rotation angles produce counter-clockwise rotations about a coordinate axis when looking from positive axis towards the coordinate origin.

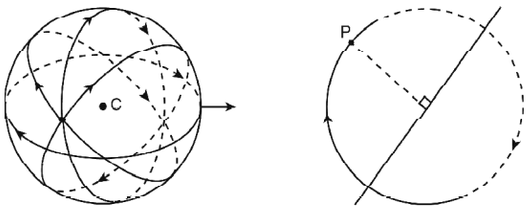


Fig. 4.30 *Rotation*

Rotation about *Z*-Axis

If the rotation is carried out about the *Z*-axis, the *z* coordinates remains unchanged (because rotation occurs in planes perpendicular to the *Z*-axis) while *x* and *y* coordinates behave exactly the same way as in two-dimensions (refer Figure 4.31). Rotation of any point $P(x, y, z)$ about *Z*-axis by an amount θ is represented by,

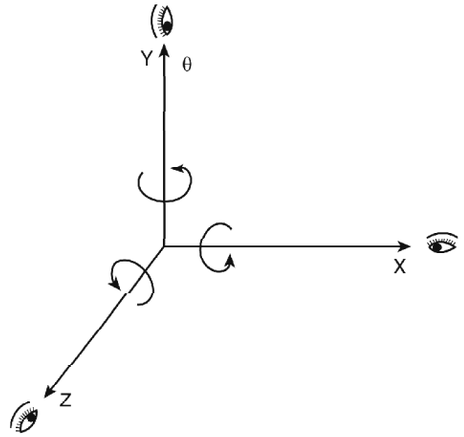


Fig. 4.31 *Rotation about Z-Axis*

$$\left.\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned}\right\} \quad ...(4.20)$$

NOTES

The corresponding transformation matrix in 4 × 4 form is,

$$[T_R]_{Z,\,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about *X*-Axis

Here rotation takes place in planes perpendicular to *X*-axis hence *x* coordinate does not change after rotation while *y* and *z* coordinates are transformed. The expression can be derived similarly as before if we replace the *Z*-axis in Figure 4.31 with *X*-axis and the other two axes accordingly, maintaining right-handed coordinate system.

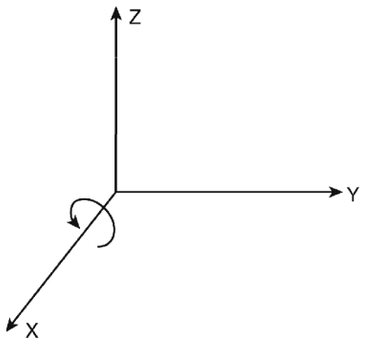


Fig. 4.32 *Rotation about X-Axis*

Looking at the Figure 4.4 replace *z* with *x*, *y* with *z* and *x* with *y* in Equation (4.20) to obtain the required expressions as,

$$\left.\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned}\right\} \quad ...(4.21)$$

The transformation matrix is thus,

$$[T_R]_{X,\,\theta} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about *Y*-Axis

This time we replace *Z*-axis with *Y*-axis and *X* and *Y*-axis with *Z* and *X*-axis respectively, in Figure 4.31 to yield Figure 4.33. Accordingly modifying Equation (4.20), we get

NOTES

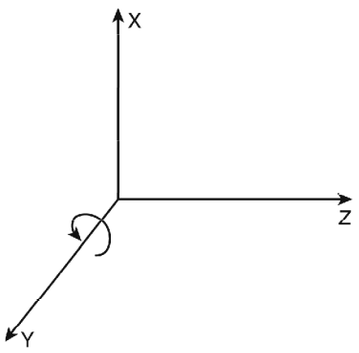


Fig. 4.33 *Ratation about Y-Axis*

$$\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \\ y' &= y \end{aligned} \qquad \text{...(4.22)}$$

The transformation matrix is now,

$$[T_R]_{Y,\,\theta} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation about any Arbitrary Axis in Space

To perform rotation about a line that is not parallel to one of the coordinate axes we first need to align the line with one of the coordinate axes through some intermediate transformation. Then the actual specified rotation is performed about that coordinate axis, the expressions for which are known standards.

Assuming an arbitrary axis in space passing through the point (x_0, y_0, z_0) and having direction cosines (C_x, C_y, C_z), rotation about this axis by some angle θ is accomplished through the following steps:

Step 1 Translate so that the point (x_0, y_0, z_0) is at the origin of the coordinate system. The required matrix is,

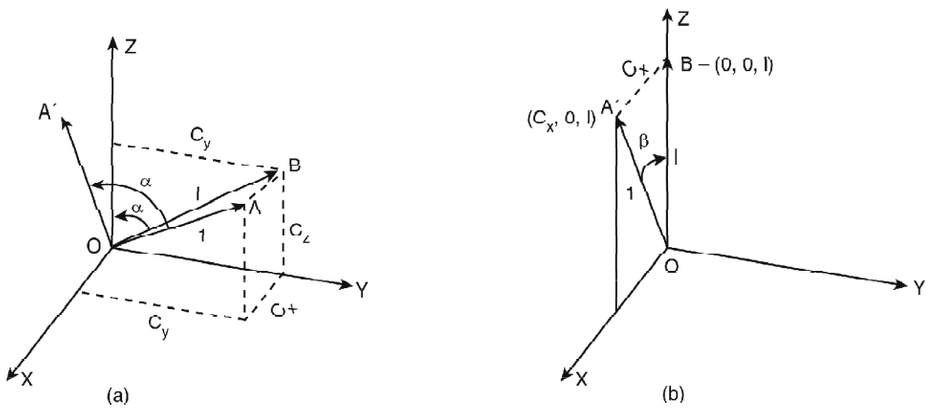
$$[T_T] = \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 2 Perform appropriate rotations to make the axis of rotation coincident with the *Z*-axis. In fact we can align the rotation axis with any of the three coordinate axes. Here our choice of *Z*-axis is arbitrary. In general, making an arbitrary axis passing through the origin coincide with one of the coordinate axes requires two successive rotations about the other two coordinate axes.

(i) Here first perform rotation about *X*-axis until the rotation axis is in the *ZX*-plane.

(ii) Then perform rotation about *Y*-axis until the rotation axis coincides with the *Z*-axis.

NOTES



(a) Unit Vector along the Axis of Rotation *(b) Rotation Axis in the ZX-Plane*

Fig. 4.34 *Rotation Axis*

To obtain the transformation matrix for Step 2A and 2B let us consider Figure 4.34(a) where $\overrightarrow{OA}$ represents the unit vector along the given axis of rotation and $\overrightarrow{OB}$ is the projection of $\overrightarrow{OA}$ on the *YZ*-plane.

The amount of rotation required in Step 2A to bring $\overrightarrow{OA}$ in the *ZX*-plane is equal to α, the angle between $\overrightarrow{OB}$ and *Z*-axis.

If $|\overrightarrow{OB}| = l$, then

$$\cos\alpha = \frac{C_z}{l}, \sin\alpha = \frac{C_y}{l}$$

Again $l = \sqrt{(C_z^2 + C_y^2)}$

So for Step 2A, i.e., for rotation about *X*-axis by angle α, the matrix is,

$$[T_R]_{X,\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{l} & \frac{-C_y}{l} & 0 \\ 0 & \frac{C_y}{l} & \frac{C_z}{l} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

NOTES

After Step 2A, $\overrightarrow{OA}$ becomes $\overrightarrow{OA'}$ and $\overrightarrow{OB}$ becomes $\overrightarrow{OB'}$ both being in the *ZX*-plane. Also the angle between *OA'* and *Z*-axis is say β in the *ZX*-plane and this is the angle through which the Step 2A – transformed rotation axis ($\overrightarrow{OA'}$) should be rotated about *Y*-axis in clockwise direction in order to coincide with *Z*-axis.

From Figure 4.34(b) $\cos\beta = \frac{l}{1}$; $\sin\beta = \frac{C_x}{1} = C_x$

Hence, rotation matrix for Step 2B is,

$$[T_R]_{y,\beta} = \begin{pmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Step 3 Rotate about *Z*-axis by the angle θ, the transformation matrix being,

$$[T_R]_{Z,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 4 (i) Reverse the rotation about *Y*-axis , the matrix being $[T_R]^{-1}_{Y,\theta}$

(ii) Reverse the rotation about *X*-axis; matrix $[T_R]^{-1}_{X,\theta}$

Step 5 Reverse the translation carried out in Step 1 with the matrix $[T_T]^{-1}$

Putting together all these steps the complete transformation is then,

$$[T_{COMB}] = [T_T]^{-1}\,[T_R]^{-1}_{X,\theta}\,[T_R]^{-1}_{Y,\theta}\,[T_R]_{Z,\theta}\,[T_R]_{Y,\theta}\,[T_R]_{X,\theta}\,[T_T]$$

4.4.4 Reflection

Unlike in 2-D, reflection of objects in 3-D occurs through a plane. In real life all the mirrors are basically planes in 3-D space. Just like 3-D rotation there are some standard reflections and arbitrary reflections guided by the orientation of the reflecting plane (refer Figure 4.35). Reflection about coordinate planes *XY, YZ, ZX* are standard ones and can be easily derived as only 1-dimension or coordinate changes in each case.

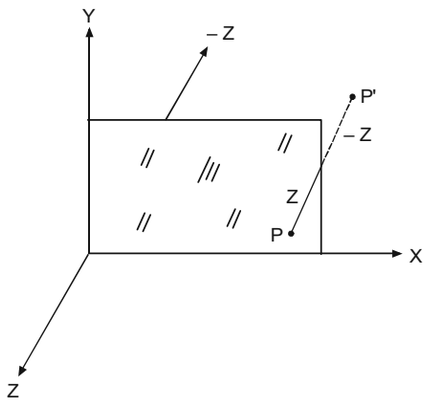


Fig. 4.35 *Reflection*

Reflection about *XY*-Plane

After reflection the object goes on the opposite side of the reflection plane along the *Z* direction. So in terms of coordinates, *z*-coordinate gets reversed in sign, $x' = x, y' = y, z' = -z$. The resulting transformation matrix is thus,

$$[T_M]_{XY} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection about *YZ*-Plane

In this case after reflection *y* and *z* coordinates remain unchanged while *x* coordinate gets reversed in sign; $x' = -x,\ y' = y, z' = z$ (refer Figure 4.36).

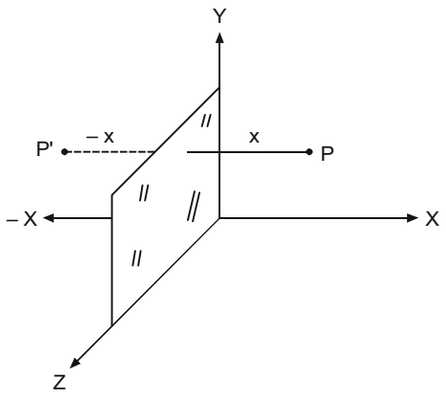


Fig. 4.36 *Reflection about YZ-Plane*

$$[T_M]_{YZ} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection about *ZX*-Plane

Here *z* and *x* coordinates remain unchanged while *y* coordinate gets reversed in

sign only; $x' = x, y' = -y, z' = z$ (refer Figure 4.37).

NOTES

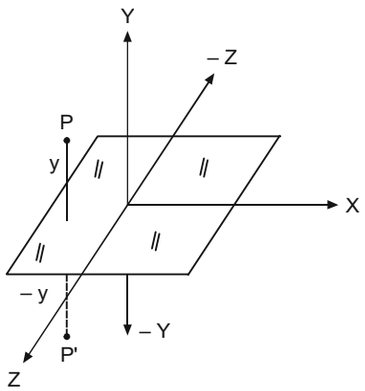


Fig. 4.37 *Reflection about ZX-Plane*

$$[T_M]_{ZX} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Reflection about any Arbitrary Plane

If the reflection plane is neither any of the coordinate planes nor parallel to any of them then for our convenience we may first align the plane with any of the coordinate planes (say, *XY*-plane) with some additional transformations. Then we can carry out the necessary reflection in terms of standard reflection about *XY*-plane. This approach is similar to that adopted for 3-D rotation about an arbitrary axis.

Assuming that the arbitrary reflection plane is defined by an in-plane point $P_0 (x_0, y_0, z_0)$ and a vector $\mathbf{N} = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$ normal to the plane, reflection about the plane is accomplished through following steps (refer Figure 4.38).

Step 1

Translate the system so that $P_0 (x_0, y_0, z_0)$ coincides with the origin of the coordinate reference frame. The required transformation matrix is,

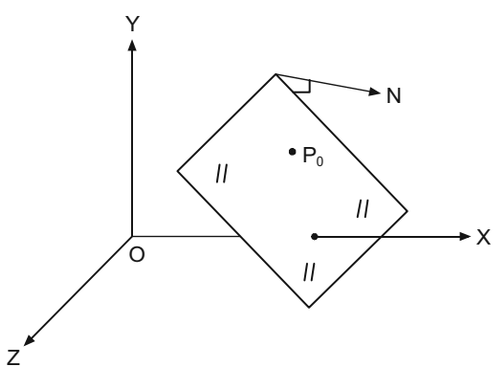


Fig 4.38 *Point P_0 in Arbitrary Plane*

$$[T_T]=\begin{pmatrix}1 & 0 & 0 & -x_0\\ 0 & 1 & 0 & -y_0\\ 0 & 0 & 1 & -z_0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Step 2

Rotate the system such that the normal vector (**N**) to the plane at origin coincides with the *Z*-axis. This will make the reflection plane the $z = 0$ coordinate plane. If the matrix representing this step is $[T_R]$ then,

$$[T_R] = [T_R]_Y [T_R]_X$$

Here $[T_R]_X$ represents rotation about *X*-axis and is given by,

$$[T_R]_X = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \frac{C_z}{l} & -\frac{C_y}{l} & 0\\ 0 & \frac{C_y}{l} & \frac{C_z}{l} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

$[T_R]_Y$ represents rotation about *Y*-axis and is given by,

$$[T_R]_Y = \begin{pmatrix}1 & 0 & -C_x & 0\\ 0 & 1 & 0 & 0\\ C_x & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

If you recall the same steps performed in case of rotation about an arbitrary axis, then you get C_x, C_y, C_z are the direction cosines of the normal vector **N** where,

$C_x = n_1/\sqrt{(n_1^2 + n_2^2 + n_3^2)}$, $C_y = n_2/\sqrt{(n_1^2 + n_2^2 + n_3^2)}$, $C_z = n_3/\sqrt{(n_1^2 + n_2^2 + n_3^2)}$ and $l = \sqrt{(C_y^2 + C_z^2)}$

Step 3

Reflect the object in question about the *XY* or $z = 0$ plane. The corresponding matrix is,

$$[T_M]_{XY} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Step 4

Reverse Step 2 by first reversing rotation about *Y*-axis and then reversing rotation about *X*-axis

The matrix is $[T_R]^{-1} = [T_R]_X^{-1} [T_R]_Y^{-1}$.

Step 5

Reverse Step 1, the inverse translation matrix being $[T_T]^{-1}$.

Combining the above transformations the resultant transformation matrix can be evaluated as,

$$[T_{COMB}] = [T_T]^{-1}\,[T_R]_X^{-1}\,[T_R]_Y^{-1}\,[T_M]_{XY}\,[T_R]_Y\,[T_R]_X\,[T_T]$$

4.4.5 Shearing

NOTES

In 2-D transformations, we discussed shear transformations relative to the *X*- or *Y*- axes to produce distortions in the shapes of planar objects. In 3-D, we can also generate shears relative to the *Z*-axis and the result is change of volume and 3-D shape of any object. This is useful in three-dimensional viewing for obtaining general projection transformations. However, it is a bit difficult to visualize the changed object after any shear transformation.

For example, a *Z*-axis 3-D shear can be expressed as,

$x' = x + az$

$y' = y + bz$

$z' = z$

The following is the corresponding transformation matrix.

$$[T_{SZ}] = \begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The effect of this transformation matrix is to alter *x* and *y* coordinate values by an amount that is proportional to the *z* value, while leaving the *z* coordinate unchanged.

3-D shearing transformations can be carried out about the other two principal axes as well.

An *X*-axis 3-D shear can be expressed as,

$x' = x$

$y' = y + a\,x$

$z' = z + b\,x$

The corresponding transformation matrix is,

$$[T_{SX}] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Similarly a *Y*-axis 3-D shear can be expressed as,

$x' = x + a\,y$

$y' = y$

$z' = z + b\,y$

The corresponding transformation matrix is,

$$[T_{SY}] = \begin{pmatrix} 1 & a & 0 & 0 \\ a & 1 & 0 & 0 \\ b & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

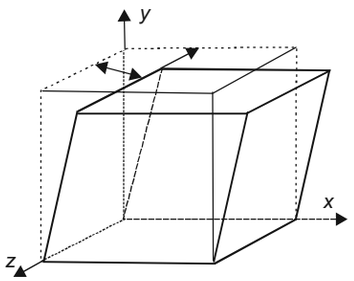


Fig.4.39 *3-D Y-Axis Shear of a Unit Cube*

In *Y*-axis shear, boundaries of planes that are perpendicular to the *Y*-axis are shifted by an amount proportional to *y*. An example of the effect of this shearing matrix on a unit cube is shown in Figure 4.39, for shearing factors *a* and *b* respectively in *X* and *Z* direction.

4.5 MULTIPLE TRANSFORMATION

As already discussed in 2D transformations, successive transformation can be combined or concatenated into a single (4×4) transformation that yields the same result. Since matrix multiplication is non-commutative i.e, the order of multiplication is important. In general,

[A][B] g [B] [A]

4.5.1 Rotation about an Axis Parallel to a Coordinate Axis

Say, we want to rotate the object about the axis other than *x*, *y* and *z* axis.

Now, we want to rotate the object about axis other than *x*, *y* and *z* axis. Here, the special case of an axis that is parallel to one of the *x*, *y* or *z* co-ordinate axis is considered.

Suppose a body with a local axis system *x¢ y¢ z¢* parallel to the fixed co-ordinate axis, *x*, *y*, *z*. Rotation of the body or object about any of the individual *x¢*, *y¢* or *z¢* axis involves following three steps:

1. Translate the body until the local axis is coincident with co-ordinate axis
2. Rotate about the specified axis.
3. Translate the body back to its original position.

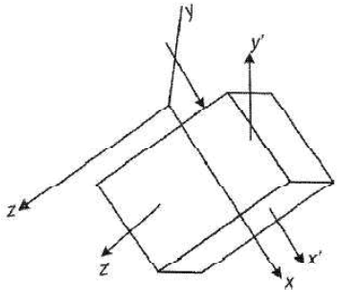


Fig. 4.40

Mathematically,

$$[X^*] = [X]\,[Ttrans]\,[Trot]\,[Ttrans]\text{-}1$$

where X* represents a transformed body,

[Ttrans] = Translation matrix,

[Trot] = Rotation matrix.

NOTES

4.5.2 Rotation About an Arbitrary Axis in Space

To perform rotation about a line that is not parallel to one of the coordinate axis we first need to align the line with one of the coordinate axes through some intermediate transformation. Then the actual specified rotation is performed about that coordinate axis, the expressions for which are known standards.

Assuming an arbitrary axis in space passing through the point (x_0, y_0, z_0) and having the direction cosines(C_x, C_y, C_z), rotation about the axis by some angle is accomplished through the following steps:

Step 1. Translate so that the point (x_0, y_0, z_0) is at the origin of the coordinate system. The required matrix is

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

Step 2. Perform appropriate rotations to make the axis of rotation coincident with the *Z* axis. In fact we can align the rotation axis with any of the three coordinate axes. Here our choice of *Z* axis is arbitrary. In general, making an arbitrary axis passing through the origin coincide with one of the coordinate axes requires two successive rotations about the other two coordinate axes.

(A) Here first perform rotation about X axis until the rotation axis in the *ZX* plane.

(B) Then perform rotation about *Y* axis until the rotation axis coincides with the Z axis.

Let us consider Fig. 4.41 (*a*) where $|\overrightarrow{OA}|$ represents the unit vector along the given axis of rotation and $\overrightarrow{OB}$ is the projection of $\overrightarrow{OA}$ on the *YZ* plane. The amount of rotation required in step (2A) to bring $\overrightarrow{OA}$ in the *ZX* plane is equal to a, the angle between $\overrightarrow{OB}$ and *z*-axis.

If $\quad |\overrightarrow{OB}| = l$, then $\cos\alpha = \frac{Cz}{l}$ and $\sin\alpha = \frac{Cy}{l}$

where $\quad l = \sqrt{C_z^2 + C_y^2}$

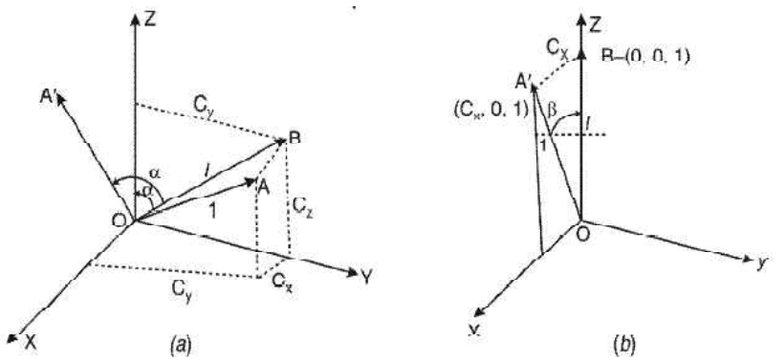


Fig. 4.41

So for step 2A i.e., for rotation about *x*-axis by angle α, the matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{l} & \frac{-C_y}{l} & 0 \\ 0 & \frac{C_y}{l} & \frac{C_z}{l} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

After step 2A, $\overline{OA}$ becomes $\overline{OA}'$ and $\overline{OB}'$ both being in the *ZX'* plane. Also the angle between *OA'* and Z axis is say β in the *ZX* plane and this is the angle through which the step 2A- transformed rotation axis (uuur) should be rotated about *Y* axis in clockwise direction in order to coincide with Z-axis.

From Fig. 4.41 (b)

$$\cos\beta = \frac{l}{1} = l$$

$$\sin\beta = C_x$$

Hence rotation matrix for step 2B is

$$= \begin{bmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} l & 0 & -C_x & 0 \\ 0 & 1 & 0 & 0 \\ C_x & 0 & l & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3. Rotate about *z*-axis by an angle θ.

∴ Transformation matrix is–

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

Step 4. Reverse the rotation about y-axis by an angle . Reverse the rotation about x-axis by an angle .

Step 5. Reverse the translation carried out in step-1 above.

NOTES

Scaling w.r.t. fixed Point

As in 2D transformations, in 3D also we get the following transformation matrix for scaling w.r.t. a selected fixed point (x_C, y_C, z_C)-

$$= \begin{bmatrix} S_x & 0 & 0 & (1-S_x)x_C \\ 0 & S_y & 0 & (1-S_y)y_C \\ 0 & 0 & S_z & (1-S_z)z_C \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Check Your Progress

7. How will you define concatenation?
8. What are the most commonly used geometric transformations?
9. How do affine transformations change the shape of a geometric object?
10. How does the reflection of objects occur in 3-D?

4.6 ANSWERS TO 'CHECK YOUR PROGRESS'

1. Once you define an object 'locally' you can place it in the global system simply by specifying the location of the origin and orientation of the axes of the local system within the global system, then mathematically transforming the point coordinates defining the object from local to global system. Such transformations are known as the transformation between the coordinate systems.
2. A point's coordinates can be expressed as elements of a matrix.
3. For each geometric transformation, there exists an inverse transformation which describes just the opposite operation of that performed by the original transformation. Any transformation followed by its inverse transformation keeps an object unchanged in position, orientation, size and shape.
4. The point about which the object is rotated is called the pivot or rotation point.
5. The basic principles of reflection transformation are:
   - The image of an object is formed on the side opposite to where the object is lying, with respect to the mirror line.
   - The perpendicular distance of the object (i.e., the object points) from the mirror line is identical to the distance of the reflected image (i.e., the corresponding image points) from the same mirror line.
6. Scaling is a transformation that changes the size or shape of an object.
7. The process of calculating the product of matrices of a number of different transformations in a sequence is known as concatenation.

8. The Euclidean transformations are the most commonly used transformations.
9. The affine transformations do not preserve lengths and angle measures and as a result they will change the shape of a geometric object.
10. The reflection of objects in 3-D occurs through a plane.

NOTES

4.7 SUMMARY

- Once you define an object 'locally' you can place it in the global system simply by specifying the location of the origin and orientation of the axes of the local system within the global system, then mathematically transforming the point coordinates defining the object from local to global system.
- A point's coordinates can be expressed as elements of a matrix. Two matrix formats are used: one is row matrix and other is column matrix format.
- For each geometric transformation, there exists an inverse transformation which describes just the opposite operation of that performed by the original transformation. Any transformation followed by its inverse transformation keeps an object unchanged in position, orientation, size and shape.
- Rotation is used to rotate objects about any point in a reference frame. Unlike translation, rotation brings about changes in position as well as orientation.
- The point about which the object is rotated is called the pivot point or rotation point.
- Scaling is a transformation that changes the size or shape of an object. Scaling with respect to origin can be carried out by multiplying the coordinate values (x, y) of each vertex of a polygon, or each endpoint of a line or arc or the center point and peripheral definition points of closed curves like a circle by scaling factors sx and sy respectively to produce the coordinates (x′ y′).
- A reflection is a transformation that produces a mirror image of an object. In 2D reflection we consider any line in 2D plane as the mirror; the original object and the reflected object are both in the same plane of the mirror line.
- The shape, size, position and orientation of 2D objects can be controlled by performing matrix operation on the position vectors of the object definition points.
- Geometric transformations are defined as changing of position for given image or graphics. Literal meaning of transformation is to alter the position of an object on a plane. The transformation is explained in triangle.
- The Euclidean transformations are the most commonly used transformations.
- In 3-D, we can also generate shears relative to the Z-axis and the result is change of volume and 3-D shape of any object.

NOTES

4.8 KEY TERMS

- 2-D transformation: It means a change in position, orientation, size or shape of graphics objects.
- **Pivot or rotation point:** The point about which the object is rotated is called the pivot or rotation point.
- **Reflection:** A transformation that produces a mirror image of an object.
- **Scaling:** A transformation that changes the size or shape of an object.
- **Concatenation:** The process of calculating the product of matrices of a number of different transformations in a sequence.
- **Geometric transformations:** These are the changing of position for given image or graphics.
- **Reflection:** It means that object produce the mirror image of the object generally every reflection has the mirror line.

4.9 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What does 2-D transformation mean?
2. How to represented points?
3. State about the general fixed point scaling.
4. What is transformation matrix?
5. Write the basic principles of reflection transformation.
6. Find the transformation that converts a square with diagonal vertices (0, 3) and (-3, 6) into a unit square at the origin.
7. Prove that 2D rotation and scaling commute if Sx = Sy or if q = np for integer n.
8. Why are homogeneous coordinates used for transformation computations in Computer Graphics?
9. What are the various processes involved in geometric transformation?
10. How does convention effect rotation?
11. What is shearing in 3-D transformation?

Long-Answer Questions

1. Explain the 2-D transformation with the help of appropriate examples.
2. Describe the reflection through an arbitrary line for transformation matrix with the help of diagram.
3. Discuss the rotation about an arbitrary pivot point with appropriate example.

4. Show how reflections in the line y = x and in the line y = -x can be performed by a scaling operation followed by a rotation.
5. Discuss about the homogeneous corrdinates with appropriate example.
6. A triangle is located at P (10, 40), Q(40, 40), R(40, 30). Work out the transformation matrix which would rotate the triangle by 90 degrees (ccw) about the point Q. Find the coordinates of the rotated triangle.
7. Explain the process of translation and scaling in geometric transformation.
8. Discuss the various types of rotations with the help of examples.
9. Explain the rotation about an axis parallel to coordinate axis with the help of diagram
10. Describe the multiple transformation and rotation about an arbitrary axis in space with appropriate example.

NOTES

4.10 FURTHER READING

Hearn, Donald and M. Pauline Baker. *Computer Graphics*. New Jersey: Prentice Hall.

Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Higher Education.

Foley, James D., Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice*. London: Addison-Wesley Professional.

Mukhopadhyay, Anirban and Arup Chattopadhyay. *Introduction to Computer Graphics and Multimedia*. New Delhi: Vikas Publishing House Pvt Ltd.

UNIT 5 GEOMETRY PERSPECTIVE, HIDDEN-SURFACE, LINES AND BEZIER CURVE, MULTIMEDIA AND ANIMATION

Structure

5.0 INTRODUCTION

Three-dimensional projection can be defined as a method of mapping three-Dimensional (3D) points to a two-Dimensional (2D) plane. The most recent methods for displaying graphical data are based on planar two-dimensional media; therefore, the use of this type of projection is widespread, especially in engineering design, computer graphics and drafting. There are basically two methods of projection-one method shows the object as it appears, called perspective projections and another method called parallel projection the object in its true size and shape.

Surface functions, such as the quadrics, can be used to describe spheres, ellipsoid and similar other smooth surfaces. For design applications, you can use super-quadrics, splines or spherical objects that represent smooth surface shapes. Additionally, construction techniques, such as sweep representation, are useful for designing compound object shapes that are built up from a set of simpler shapes, as well as surface information that can be stored in octree representations.

A multimedia system implies a combination of specified hardware components with certain minimum capabilities and compatible software that has an interactive interface studded with different media elements. The multimedia applications are offered by new media in which various interfaces use graphical interaction using GUI. The applications are used to create, modify, view and interact with multimedia data. Multimedia applications are required to manage a complicated range of run time tasks specified by the application developer. The hardware and software requirements for creating multimedia. When text lives in a computer, instead of on printed pages, the computer's powerful processing capabilities can be applied to make the text more lively or interactive. HTML presumes a Document Type Definition (DTD), which specifies valid tag names, attributes and their syntax. But once you understand their properties and uses, coding or marking up a document is quite simple.

In this unit you will study about the geometric projection, orthographic and oblique projections, perspective transformation, hidden surface and lines, back face detection and removal, binary space partition, cubic Bezier curve, multimedia, applications of multimedia, multimedia hardware, basic tools of multimedia, multimedia building blocks, animation and graphic file format.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Discuss the geometric projection, orthographic and oblique projections
- Describe the perspective transformation
- Describe the hidden surface and lines
- Discuss the back face detection and removal
- Explain the binary space partition and the concept of cubic Bezier curve
- Discuss the concept of multimedia
- Explain the various fields of applications of multimedia
- Discuss the multimedia hardware and explain the basic tools of multimedia
- Explain the multimedia building blocks

5.2 GEOMETRIC PROJECTION

In perspective projection the viewpoint or center of projection is at a finite distance from the viewplane or plane of projection as shown in Fig. 5.1. The size of a projected object is inversely proportional to its distance from the viewplane. The perspective projections of any set of parallel lines that are not parallel to the plane of projection converge to a *vanishing point*. In 3D, the parallel lines meet only at infinity, so the vanishing point can be thought of as the projection of a point at infinity. There could therefore be an infinite number of vanishing points, one for each of the infinite directions in which a line could be oriented.

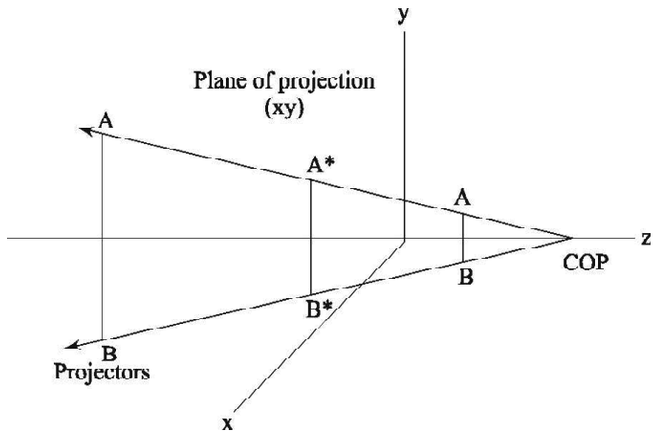


Fig. 5.1 *Perspective projection with center of projection on the z axis at COP and plane of projection is xy plane. AB is the object to be projected and A*B* is the projected object.*

If a set of lines is parallel to one of the three principal axes, the vanishing point is called an axis vanishing point. There are at most three such points, corresponding to the number of principal axes cut by the projection plane. For example, if the projection plane cuts only the *z*-axis (and is therefore normal to it), only *z*-axis has a principal vanishing point, because the lines parallel to either the *y*- or *x*-axes are parallel to the projection plane and have no vanishing point.

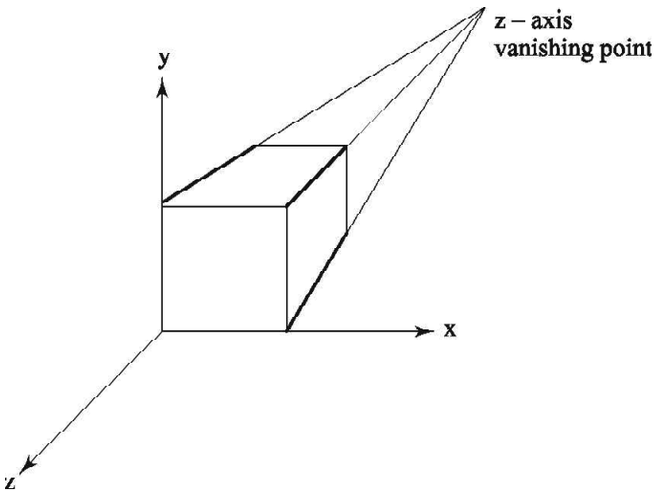


Fig. 5.2 *One-point perspective projection.*

Perspective projections are categorized by their number of principal vanishing points and therefore by the number of axes the projection plane cuts. Hence, the names, one-point, two-point and three-point perspective for one principal vanishing point, two principal vanishing points and three principal vanishing points respectively as shown in Fig. 5.3. Two-point perspective is commonly used in architectural, engineering, industrial designs, and in advertising drawings. Three-point perspectives are used less frequently, since they add little realism beyond that afforded by the two-point perspective. The complete classification of projections is shown in Fig. 5.3.

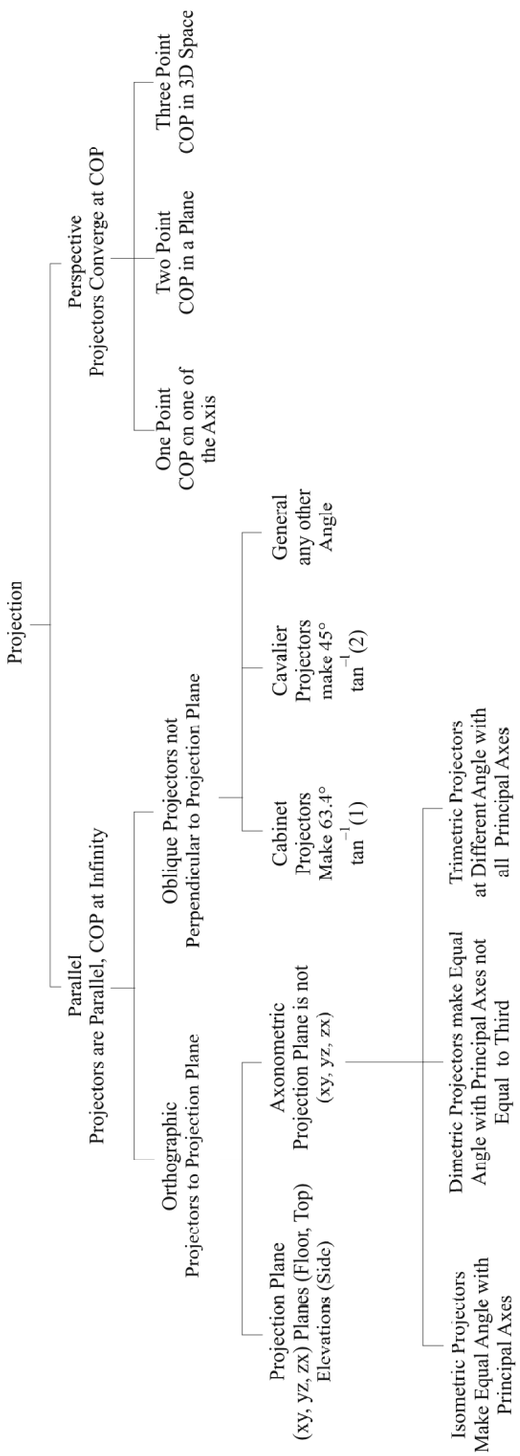


Fig. 5.3 *Planar geometric projection classification.*

5.2.1 Orthographic Projections

Parallel projection shows the true image, size and shape of the object. The angle made by the direction of projecting lines with the projection plane or view plane. Projection rays (projectors) emanate from a COP (Centre of Projection) and intersect Projection Plane. The centre of projection for parallel projectors is at

infinity. The length of a line on the projection plane is the same as the 'true length'.

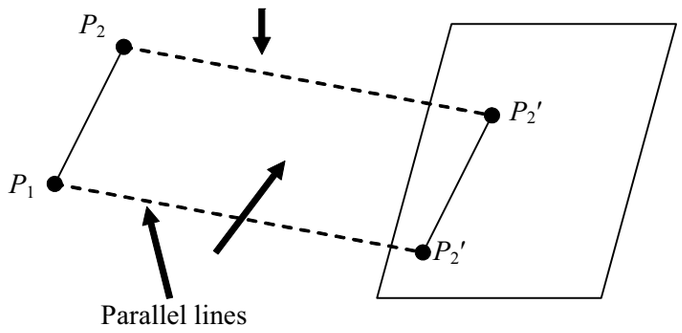


Fig. 5.4 *Illustration of Parallel Projection*

There exist two different types of parallel projections in practice. In **orthographic** projection, the direction of projection is perpendicular to the projection plane. In case of **oblique** projection, the direction of projection is not perpendicular to the projection plane. Let us consider Figure 5.5 that illustrates the parallel projection of a point (*x, y, z*). The projection plane is across the line *z = 0.* The values *x, y* represent the orthographic projection values and the values *xp, yp* are the oblique projection values.

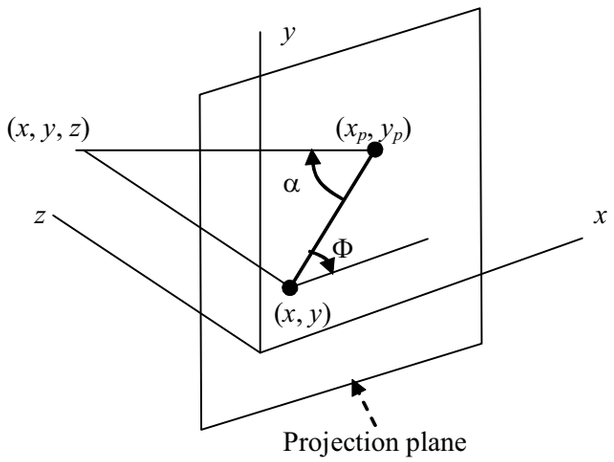


Fig. 5.5 *Illustration of Orthographic Projection*

If we look at orthographic projection, it just discards the z coordinates. Drawings in engineering frequently use top, front, side orthographic views of an object. The following diagram illustrates three orthographic views of an object (See Figure 5.6)

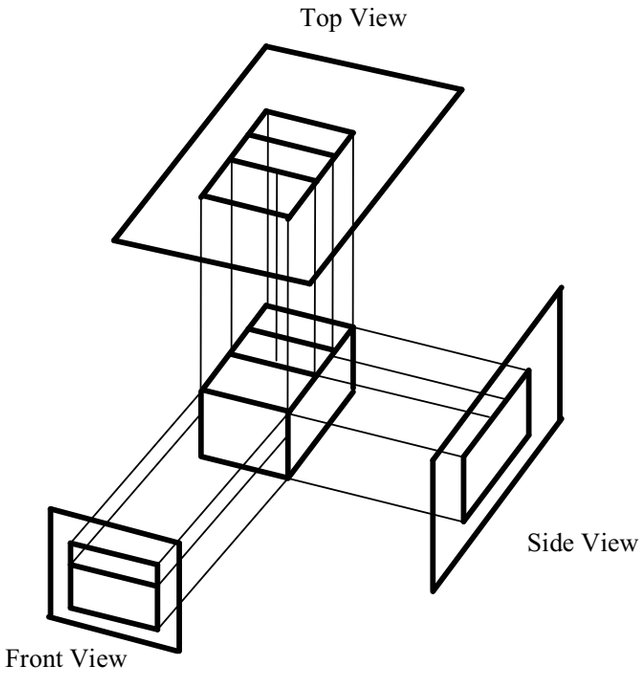


Fig. 5.6 *Orthographic View of an Object*

Orthographic projections that show more than one side of an object are called **axonometric** orthographic projection. The most common axonometric projection can be considered as an **isometric** projection where each coordinate axis is intersected by the projection plane in the model coordinate system at an equal distance.

5.2.2 Oblique Projections

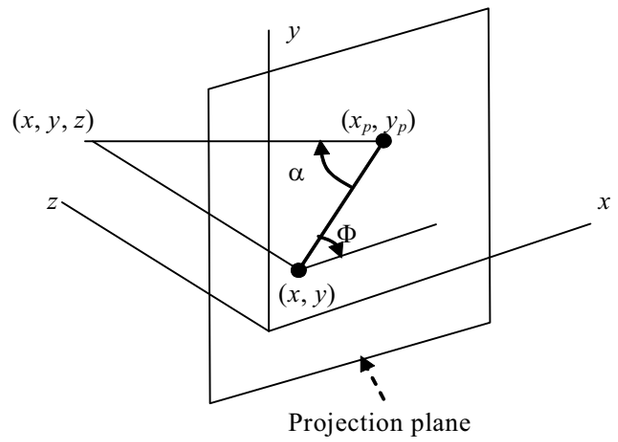


Fig. 5.7 *Illustration of Oblique Projection*

Case (*i*) If $A = 45°$, then $\tan A = 1 \Rightarrow L_1 = 1$. This is the case of **Cavalier** projection wherein lines the projectors are not perpendicular to the projection plane but are parallel from the object to the projection plane.

The projectors are defined by two angles A and d where: A = angle of line (x, y, x_p, y_p) with projection plane, d = angle of line with x-axis in projection plane L = Length of Line , then:

$$\cos d = (x_p - x)/L \Rightarrow x_p = x + L \cos d,$$
$$\sin d = (y_p - y) \Rightarrow y_p = y + L \sin d,$$
$$\tan A = z/L$$

Now we define

$$L_1 = L/z \quad \Rightarrow L = L_1 z,$$

Therefore,

$$\tan A = L/z = 1/L_1$$
$$x_p = x + z(L_1 \cos d), \text{ and}$$
$$y_p = y + z(L_1 \sin d)$$

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L_1 \cos q & L_1 \sin q & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now if the angle $A = 90°$ (the projection line is perpendicular to Projection Plane) then $\tan A = \infty \Rightarrow L_1 = 0$, so we have an orthographic projection. There are two special cases of oblique projection (perpendicular to the projection plane) that are projected with no change in length.

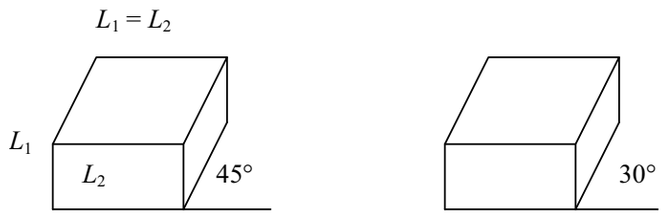


Fig. 5.8 *Two Cubes Cavalier Projection*

Case (ii): If $\tan A = 2$, then $A = 63.40°$, $L_1 = \frac{1}{2}$. The lines that are perpendicular to the projection plane are projected $\frac{1}{2}$ at length. This is also called a Cabinet projection.

$$L_1 = ½ L_2$$

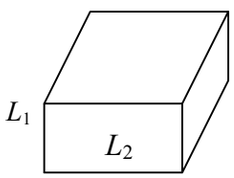


Figure 5.9 *Illustration of Cabinet Projection*

5.3 PERSPECTIVE TRANSFORMATION

More realistic drawings of objects can be made with perspective projections, because size–distance relationships are observed. Unlike parallel projections, the projectors converge at the viewpoint. A perspective transformation is determined by specifying a definite center of projection (COP) (also called viewpoint), and a plane of projection. Let $P(x, y, z)$, be any point in 3D, and COP is at $C(0, 0, -d)$ onto the –ve z-axis, as shown in Fig. 5.10.

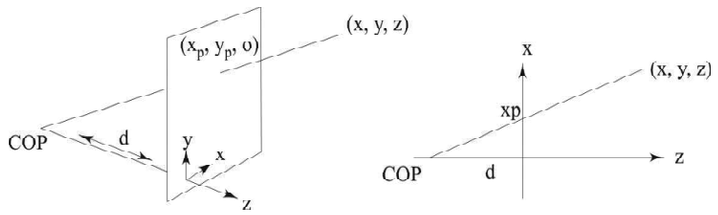


Fig. 5.10 *Standard perspective projection. The COP is onto the –ve z-axis at C(0,0,–d).*

The parametric equation of the line CP, starting from C and passing through P is

$$\begin{aligned} &= && C + t(P - C)\ 0 \le t \le \infty \\ &= && (0, 0, -d) + t\{(x, y, z) - (0, 0, -d)\} \\ &= && (0, 0, -d) + t\{(x, y, z + d)\} \\ &= && \{tx, ty, -d + t(z + d)\} \end{aligned}$$

When $t = t^*$, we get $P^* = \{t^*x, t^*y, -d + t^*(z + d)\}$. Since P^* lies on $z = 0$ plane, $-d + t^*(z + d) = 0, \Rightarrow t^* = \dfrac{d}{z + d}$. Hence P^* is

$$x^* = x\,\frac{d}{z + d}$$

$$y^* = y\,\frac{d}{z + d}$$

$$z^* = 0$$

Using homogeneous coordinates, the matrix representation is

$$[P^*][x^* \quad y^* \quad z^* \quad 1] = [P][x\ y\ z\ 1][T_{per}] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $[T_{per,z}]$ represents the standard perspective transformation matrix, also called one-point perspective transformation. Perspective projection is similar to perspective vision, in that faraway objects appear smaller as shown in Fig. 5.11(a). Here also, straight lines are projected to straight lines. Let l_1 and l_2 be

two straight lines parallel to each other, and also parallel to the plane of projection. Then the projection of l_1 and l_2 (l^*_1, l^*_2) will also be parallel to each other. If l_1 and l_2 are parallel to each other, and not parallel to the plane of projection, then the projection of l_1 and l_2 (l^*_1, l^*_2) will meet in the plane of projection as shown in Fig. 5.11(b).

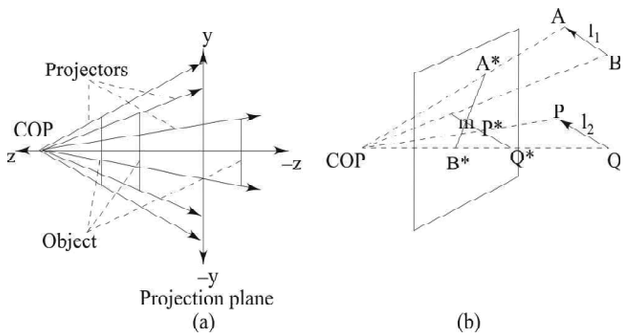


Fig. 5.11 *(a) Perspective projection. (b) Vanishing point m.*

In order to derive the single point perspective transformations along x- and y-axes, we can construct figures similar to 5.10, as shown in Fig. 5.12, but with the corresponding COPs at $(-d, 0, 0)$ and $(0, -d, 0)$ on the negative x- and y–axes, respectively.

The parametric equation of the line CP, starting from C and passing through P is

$$= \quad C + t(P - C)\ 0 \le t \le \infty$$
$$= \quad (-d, 0, 0) + t\{(x, y, z) - (-d, 0, 0)\}$$
$$= \quad (-d, 0, 0) + t\{(x + d, y, z)\}$$
$$= \quad \{-d + t(x + d), ty, tz\}$$

When $t = t^*$, we get $P^* = \{-d + t^*(x + d), t^*y, t^*z\}$. Since P^* lies on $x =$ 0 plane, $-d + t^*(x + d) = 0, \Rightarrow t^* = \dfrac{d}{x + d}$.

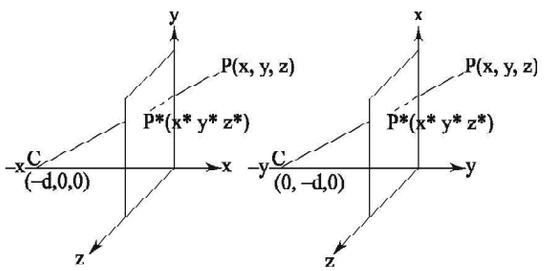


Fig. 5.12 *One-point vanishing.*

Hence P^* is $x^* = 0$

$$y^* = y\frac{d}{x + d}$$

$$z^* = z\frac{d}{x + d}$$

Using homogeneous coordinates, the matrix representation is

$$[P^*][x^* \quad y^* \quad z^* \quad 1] = [P][x \quad y \quad z \quad 1][T_{per,x}] \begin{bmatrix} 0 & 0 & 0 & \frac{1}{d} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, the one-point perspective transformation w.r.t. y-axis is therefore

$$[P^*][x^* \quad y^* \quad z^* \quad 1] = [P][x \quad y \quad z \quad 1][T_{per,y}] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This shows that the farther is a point in the negative z direction, the smaller is its projection. The length of a line connecting two points in the $z = 0$ plane is unchanged by the perspective transformation. If two points are behind the $z = 0$ plane (negative z) the length gets reduced, which is desirable. However, if one or both the points are in front of the $z = 0$ plane, the resulting enlargement of the image may make it necessary to change the window or viewport. Let the COP be equal to (d_x, d_y, d_z). The perspective projection of any point with $z = d_z$ becomes infinite, and that of a point with $z > d_z$ becomes inverted. Hence, it is simplest if we make all z values in the points matrix negative. The factors d_x and d_y take into account the horizontal and vertical components of the distance between points on the object and the COP or viewpoint. These effects can be neglected for simple projections. Hence, very satisfactory results are obtained with $d_x = d_y = \infty$.

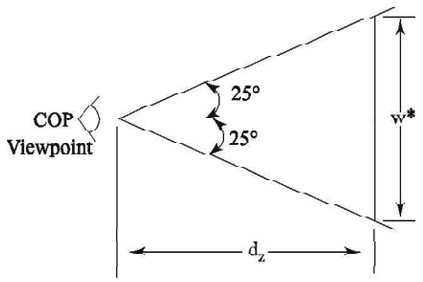


Fig. 5.13 *Normal Viewing Distance and Subtended Angle*

The basic guide for computing the viewing distance can be based on Fig. 5.17. The distance w^* is approximately equal to the desired screen projection of the longest dimension of the object. An angle of 50°, which is roughly the angle subtended by human vision, is used to produce an undistorted perspective. Thus the viewing distance d_z is estimated by

$$d_z = \frac{w^*}{2/\tan 25} \sim w^* \tag{5.1}$$

Perspective transformation with 4 × 4 matrix is consistent with the homogeneous coordinate system. However, when it is used, the resultant points matrix must be normalized to maintain unity in the fourth column.

The perspective points matrix cannot be rotated in three dimensions without distortion unless it is first multiplied by the inverse of the perspective transformation matrix. Actually, it is simpler to store an unmodified copy of the points matrix and discard the perspective-transformed points after display. Thus, the perspective projection is always a viewing transformation and never a modeling transformation.

One-, Two-, and Three-point Perspective

Perspective projection produces realistic views but does not preserve relative proportions of object dimensions. Projections of distant objects are smaller than the projections of objects of the same size that are close to the plane of projection or the COP as shown in Fig. 5.14. This characteristic feature (anomaly) of perspective projection is called *perspective foreshortening*.

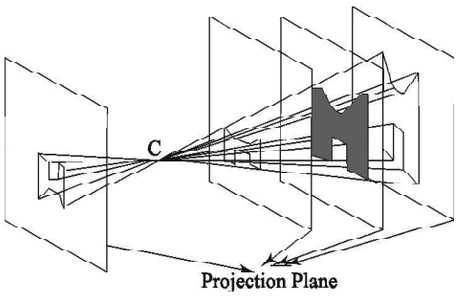


Fig. 5.14 *Perspective Anomaly: Perspective Foreshortening*

Objects behind the center of projection are projected upside down and backward onto the viewplane. This type of anomaly is called view *confusion* as shown in Fig. 5.15.

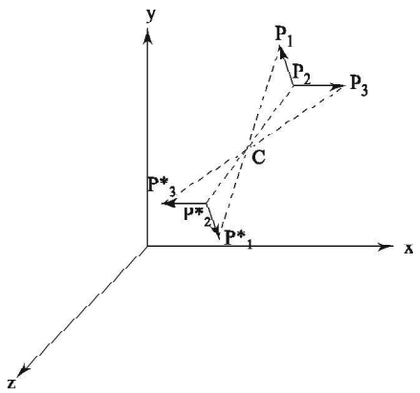


Fig. 5.15 *Perspective Anomaly: View Confusion*

We also encounter problem when a line segment joining a point which lies in front of the viewer to a point in back of the viewer is projected to a broken line of infinite extent. This happens for those points on a line which lies in a plane that passes through the center of projection. This creates *topological distortion* as shown in Fig. 5.16. Another anomaly of perspective projection is the illusion that certain sets of parallel lines, after projection, appear to meet at some point on the plane of projection as shown in Fig. 5.15. These points are called *vanishing points*. Computer graphics practitioners and engineers are familiar with the

classification of perspective drawings by the number of vanishing points, locations where parallel lines in an object appear to meet or converge.

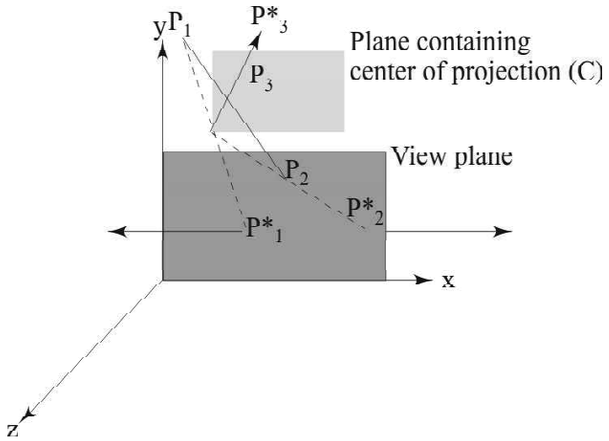


Fig. 5.16 *Perspective Anomaly: Topological Distortion*

Fig. 5.17 *Perspective Anomaly: Parallel Lines Appear to Meet at Infinity (Vanishing Points)*

For each set of the parallel lines which are not parallel to the plane of projection we get a vanishing point. Parallel lines not parallel to the viewplane appear to meet at some point. If in a perspective projection parallel lines remain parallel, the associated vanishing point is at infinity. Parallel lines that are parallel to the plane of projection are projected as parallel lines, but lines which are not parallel to the plane of projection appear to vanish at a distant point. There is an infinite number of these, one for each set of parallel lines in the object (of the infinite number of directions the lines can be oriented in). Principal vanishing points are formed by the apparent intersection of lines parallel to any of the three principal axes (*x*, *y* and *z* axes). The number of principal vanishing points is determined by the number of principal axes intersected by the plane of projection as shown in Fig. 5.18. If the COP is on one of the axes, parallel lines not parallel to the viewplane appear to meet at infinity in that direction, it is called one-point vanishing as shown in Fig. 5.18(a).

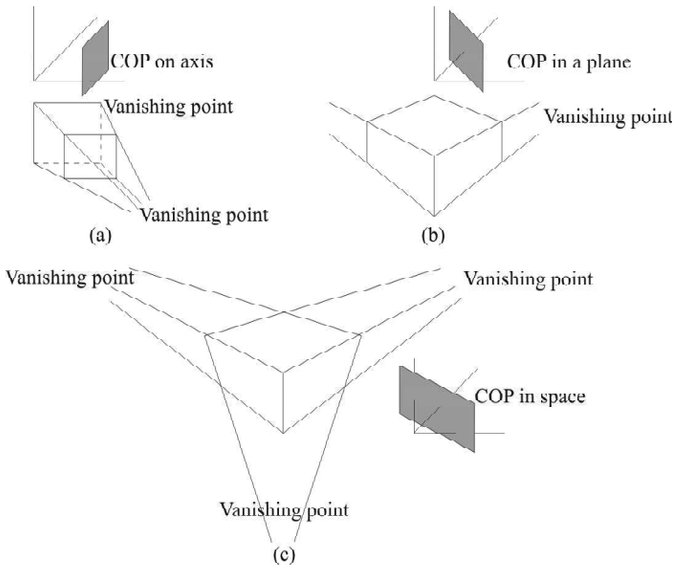


Fig. 5.18 *Vanishing Points: (a) One-Point (b) Two-Point (c) Three-Point*

We could observe it in daily life, when we stand in the middle of a long road, railway track, long corridor etc., as shown in Fig. 5.17. When the center of projection is in a plane, it is called two-point vanishing as two sets of lines, parallel to the plane of projection (axis) vanish as shown in Fig. 5.18(b). In a real life scenario, we could observe this phenomenon when we stand at a road corner (from where both sides are visible), sides of both the roads (parallel) appear to meet at distant points (on the two ends), also the objects on the respective sides gradually decrease in size as the distance from the center of projection increases, as shown in Fig. 5.19.

Fig. 5.19 *One-Point Vanishing*

When the center of projection is in 3D space then it is called three-point vanishing as all the lines parallel to the three principal axes vanish (Fig. 5.18(c)). In real-life scenario, this can be observed when we stand at the foot of a high-rise tower or building, and observe that all the sides parallel to the three principal axes appear gradually reducing in size in all the three directions as the distance increases from the viewer (Fig. 5.20).

(a) (b)

Fig. 5.20 *Real-Life Vanishing Points: (a) Two-Point, (b) Three-Point*

5.4 HIDDEN SURFACE AND LINES

A major consideration in the generation of realistic graphics display is identifying those parts of a scene which are visible from a selected viewing position. There are several approaches to solve this problem, as well as various algorithms that are already derived to identify visible objects efficiently for different types of applications. Some methods require too much memory, while others involve more execution time and some refer only to special types of objects. Selecting a method for a particular application can rely on factors such as complexity of the scene, available equipment, types of objects to be displayed, static or animated displays that need to be generated etc. The various algorithms are called visible-surface detection methods or hidden-surface elimination methods; although there can be subtle differences between identification of visible surface and elimination of hidden surfaces.

General Principles

Generally, the hidden surfaces need not be drawn. Therefore, if you can quickly compute which surfaces are hidden, then you can bypass them and draw only the surfaces that are visible. For example, if you have a solid six side cube, (Figure 5.21) at most, three of the six sides are visible at any time. So, at least three of the sides need not be even drawn because they are the back sides.

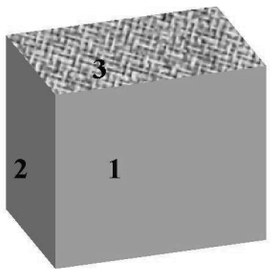


Fig. 5.21 *A Solid Cube*

There are two approaches to render a surface:

1. Object space
2. Image space

Object space algorithms work on the objects themselves before they are converted into pixels in the frame buffer. The resolution of the display device is irrelevant as this calculation is done at the mathematical level of the objects. For each object, the parts of the object which are visible is determined. This involves comparing the polygons in the object to other polygons in the object and to polygons in every other object if available.

Image space algorithms do their work while the objects get converted into pixels in the frame buffer. The resolution of the display device is significant here as this is done on the pixel basis. For each pixel in the frame buffer:

- Determine which pixel is closest to the viewer at that pixel location.
- Colour that pixel with the colour of that polygon at that location.

As you know that the object space algorithms is generally used with the vector hardware whereas the pixel-based (image space) algorithms tended are primarily used with raster hardware. When you talk about three dimensional transformations, the Z-values are also calculated along with the *x* and *y* values. Now these Z values will be used to determine which parts of which polygons (or lines). There are different levels of checking that can be done:

- Object
- Polygon
- Part of a polygon

Sometimes you may not like to cut out polygons that are behind other polygons. If the front-most polygons are transparent then you may see through it to the polygons that lie behind it.

Coherence

This idea of coherence was also used before in our line drawing algorithm. Local similarities that are exploited to reduce the amount of computation needed are:

1. **Face:** Properties (like colour, lighting) vary smoothly across a face or polygon.
2. **Depth:** Adjacent areas on a surface have same depths.
3. **Frame:** Images at successive time intervals tend to be similar.
4. **Scan line:** Adjacent scan line tend to have similar spans of objects.
5. **Area:** Adjacent pixels tend to be covered by the same face.
6. **Object:** If objects are separate from each other, then you only need to compare the polygons of the same object, and not one object to another.
7. **Edge:** Edges only disappear when they go behind another edge or face.
8. **Implied edge:** Line of intersection of two faces can be determined by the endpoints of the intersection.

Extents

It is often easier to deal with a simple version of the object in a 2D bounding box. You can convert a complex object into a simple outline, usually in the shape of a

box and then, work with the boxes. Every part of the object will fall within the bounding box. Checks can then be made on the bounding box to make quick decisions. For more details, checks would be made on the object in the box. There are several ways to define the bounding box. The simplest technique is to take the minimum and maximum x, y, and z values to create a box.

5.4.1 Back Face Detection and Removal

A simple example of an object space algorithm is the back-face removal (also called the Back-Face cut) where no faces on the back of the object are displayed. This algorithm removes about half of the total polygons in the image since almost half of the faces of objects are back faces. If you look at a left handed viewing system and if the value C is less than 0 then a back face in a right-handed system. So, the algorithm (for left- handed system) is:

1. Compute $\overline{N}$ for every face of the object.
2. If C (z component) is greater, than a back face then do not draw the back face, as shown in Figure 5.22:

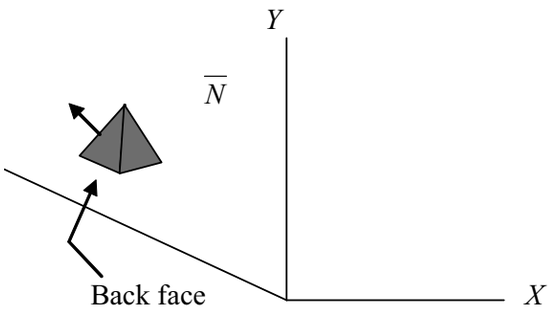


Fig. 5.22 *Illustration of Back Face of an Object*

This simple method is suitable for an orthographic projection only. It is a little more complicated for a perspective projection.

However, in case of Figure 5.22, the visible surfaces are different for perspective and orthographic projection. The sides are invisible for an orthographic projection, but not for a perspective projection. For a perspective projection, you need to determine whether the center of projection is inside or outside the planes of the polygons of the object. If the center of projection is outside then it is visible and if the centre of the projection is inside, then plane is not visible. There are two possible methods to determine this:

1. Put the centre of the projection into the plane equation and determine whether it is inside or outside. You should compute the plane equation before the perspective transformation.
2. If the angle between the plane normal (N) and the vector from any point on the plane to the center of projection, $V > 90°$ ($N.V < 0$) then that plane is not visible.

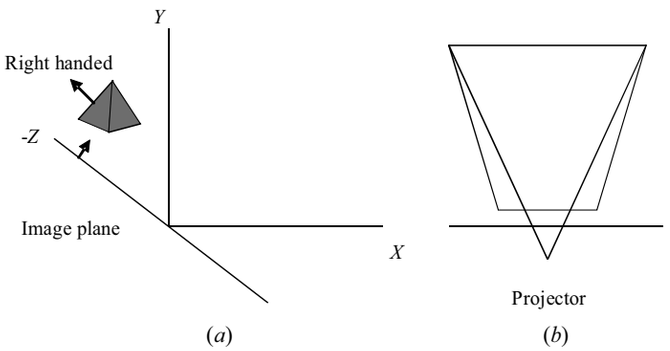


Fig. 5.23 *(a) Right Handed Image (b) Left Handed Image*

Limitations of Back-Face Removal Algorithm

1. It can only be used on solid objects modeled, for example, a polygon mesh. This is the most general modeling approach for scan line graphics systems. The renderer usually converts everything to polygons to render, even if objects are defined in a different manner, for example by parametric cubic patches or implicit equations.
2. It works well for convex polyhedra but not necessarily similar efficient for concave polyhedra.

Even if we use to apply another algorithm for visible surface determination, the back-face cut is a acceptable preprocessing step. For colour shading we should calculate the normals for all of the polygons anyway.

5.5 Z-BUFFERS ALGORITHMS

The *z*-buffer or depth buffer is the simplest visible surface or hidden surface algorithm. It is an image space algorithm. The *z*-buffer is a simple extension of the frame buffer idea. A frame buffer is used to store the attributes of each pixel in the image space. The *z*-buffer is a separate depth buffer used to store the *z*-coordinates, or depth of every visible pixel in the image space. In use, the depth or z-value of a new pixel to be written to the frame buffer is compared to the depth of that pixel stored in the *z*-buffer. If the comparison indicates that the new pixel is in front of the pixel stored in the frame buffer, then the new pixel is written to the frame buffer and the *z*-buffer updated with new *z*-value. If not, no action is taken. Conceptually, the algorithm is a search over x, y for the largest value of $z(x, y)$. This algorithm is frequently implemented for polygonally represented scenes and is also applicable for any object for which depth and shading characteristics can be calculated. Scenes can contain mixed object types and may be of any complexity (F. Bilotti. 'Design of a dual-polarization linear patch array via full-wave analysis'. Microwave and Optical Technology Letters).

The algorithm

Step-1 : Set the frame buffer to the background intensity or colour.

Step-2 : Set the z-buffer to the minimum z-value.

Step-3 : Scan converts each polygon into an arbitrary order.

Step-4 : For each pixel (x, y) in the polygon, calculate the depth $z(x, y)$ at that pixel.

Step-5 : Compare the depth $z(x, y)$ with the value stored in the z-buffer at that location

Step-6 : If $z(x, y) > z$-buffer(x, y), then write the polygon attributes to the frame buffer and replace z-buffer (x, y) by $z(x, y)$, otherwise no action is taken.

Scan-line Method

Scan-line visible surface and visible line algorithms are extensions of the scan polygon techniques. Scan line algorithms reduce the visible lines/ visible surface problems from the three dimensions to two. A scan plane is defined by the viewpoint at infinity on the positive z-axis and a scan line, as shown in Figure 5.24. The intersection of the scan plane and the three-dimensional scene defines a one-scan-line-high window.

Figure 5.24 shows the intersection of the plane with the polygons. The figure points that the visible surface problem is reduced to decide which line segment is visible for each point on the scan line. At first glance, it might appear that the ordered edge list algorithm is directly applicable.

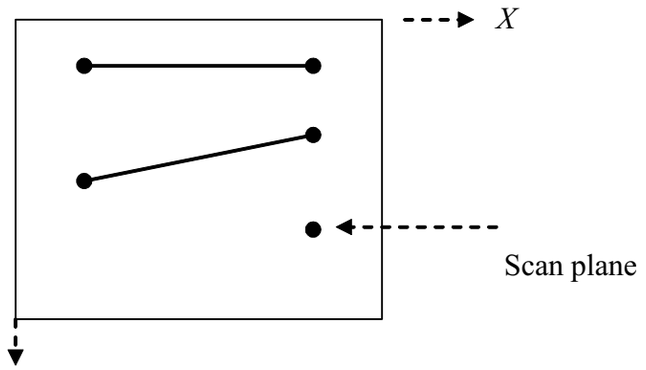


Fig. 5.24 *Illustration of Scan Plane*

However, Figure 5.25 shows that this gives incorrect results. For example, for the scan line shown in Figure 5.25, there are four active edges on the active edge list. The intersections of these edges with the scan line are shown by the small dots in Figure 5.25. Extracting the intersections in pairs causes the pixels between 1 and 2 and between 3 and 4 to be activated.

The pixels between 2 and 3 are not activated. This result is incorrect. A 'hole' is left on the scanline, where infact the scanline intersects two polygons.

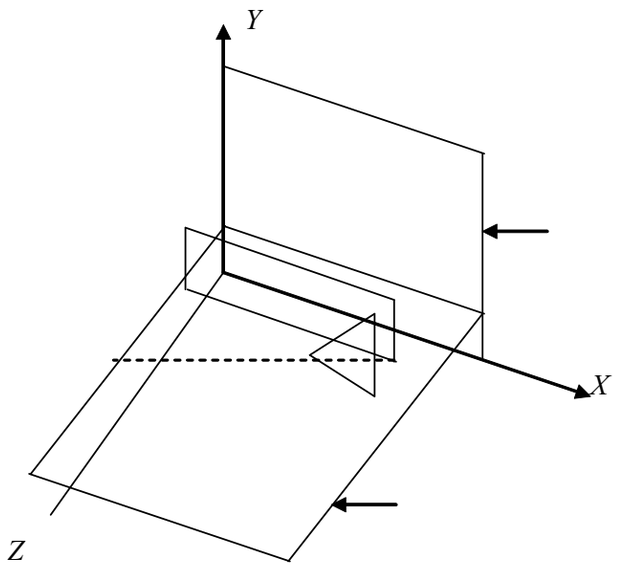


Fig. 5.25 *Another Illustration of Scan Plane*

A-Buffer Method

A-buffer method is an extension of the techniques which are used in depth-buffer method and represents an antialiased, area-averaged method used for the implementation of a surface-rendering system called REYES (which means Renders Everything You Ever Saw).

A demerit of the depth-buffer method is that it can only find one visible surface at every pixel position. In other words, it deals with opaque surfaces and can not accumulate intensity values for more than one surface, as is necessary if transparent surfaces are to be displayed. The A-buffer method expands the depth buffer such that each position in the buffer may reference a linked list of surfaces. Thus, several surface intensities can be taken into consideration at each pixel position, and object edges can be anti-aliased. Each position in the A-buffer has two fields:

- **Depth field:** Stores a positive or negative real number
- **Intensity field:** Stores surface-intensity information or a pointer value.

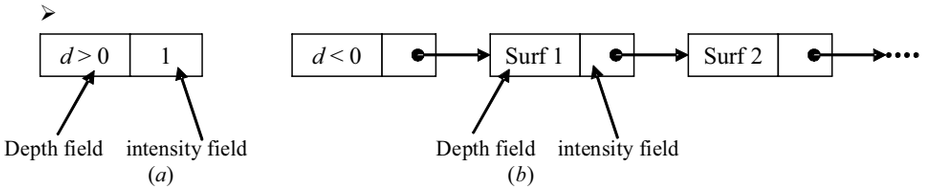


Fig. 5.26 *Organization of A-Buffer Pixel Position with (a) Single Surface Overlap and (b) Multiple Surface Overlap*

The number stored at that position (the depth field) is the depth of a single surface overlapping the corresponding pixel area. If the depth field is positive,

then the intensity field stores the 'Red Green Blue' components of the surface colour at that point and the per cent of pixel coverage, as shown in Figure 5.26(a). A negative value depth field indicates multiple-surface contributions to the pixel intensity. In this case, the intensity field stores a pointer to a linked list of surfaces as shown in Figure 5.26(b). The data for each surface in the linked list includes:

- 'Red Green Blue' intensity components
- The opacity parameter
- Depth
- Percentage of area covered
- The surface identifier
- The pointer to the next surface

A-buffer can be constructed by using the methods of the depth-buffer algorithm in the same manner. The scan lines are processed to find surface overlaps of pixels along the individual scanlines. The surfaces can be subdivided into a clipped area and polygon mesh against the pixel boundaries. It is possible to compute the intensity of each pixel as an average of the contributions from the overlapping surfaces by using the opacity factors and per cent of surface overlaps.

Depth-sorting Method

By using the image-space and object-space operations, the depth-sorting method performs the following basic functions:

1. Surfaces are sorted in decreasing order of depth.
2. Surfaces are scan converted in order such that starting from the surface of greatest depth lowest.

Sorting operations are taken in both image and object space, and the scan conversion of the polygon surfaces is only performed in the image space. This method for solving the hidden-surface problem is often called as the 'painter's algorithm'. In this method, first of all, the surfaces are sorted according to their distance from the view plane. The intensity values for the outermost surface are then stored into the refresh buffer. The surface can be painted with intensities onto the frame buffer over the intensities of the previously processed surfaces by taking each succeeding surface in turn.

Depending upon the depth, the painting polygon surfaces onto the frame buffer is carried out containing several steps. Suppose you are viewing along the z direction, the surfaces are ordered on the first path according to the largest *z*-value on each surface. The surface *S* with the greatest depth is then compared to the other surfaces in the list to determine if there are any overlaps in depth. If there exists no depth overlaps, then *S* is scan converted. Figure 5.30 demonstrates two surfaces that overlap in the *xy*-plane but they do not have depth overlap.

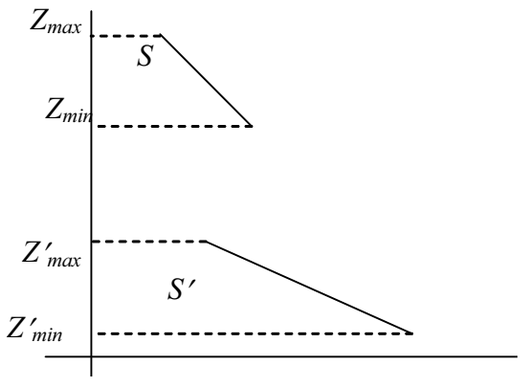


Fig. 5.27 *Two Surfaces having No Depth Overlap*

This process is then repeated for the next surface in the list. As long as no overlap is found, each surface is processed in depth order until all have been scan converted. If a depth overlap is found at any point in the list, then you have to make some additional comparisons to determine whether any of the surfaces should be reordered. The following tests for each surface overlapping *S*. If any one of these tests is true, then there is no need of any reordering of that surface.

The tests are listed in order of increasing difficulty:

1. For the two surfaces, the bounding rectangles in the *xy*-plane do not overlap.
2. Surface *S* is completely behind the overlapping surface relative to the viewing position.
3. Relative to the viewing position the overlapping surface is completely in front of S.
4. Projections of the two surfaces on the view plane does not overlap.

These tests are performed in the order listed and proceed to the next overlapping surface as soon as you find one of the tests is true. If all the overlapping surfaces pass at least one of these tests, then none of them is behind *S*. No rendering is then necessary and then *S* is scan converted.

Test 1 is performed in two parts. First, you check for overlapping in the *x*-direction, and next, you check for overlapping in the *y*-direction. If either of these directions represents no overlapping, then the two planes do not obscure one other. An example of two surfaces which overlap in *z*-direction but not in *x*-direction is represented in Figure 5.28:

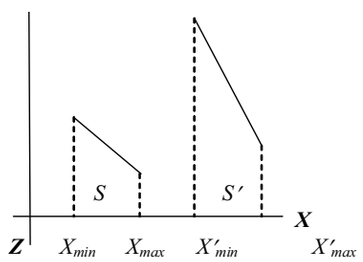


Fig. 5.28 *Two Surfaces having Depth Overlap but No Overlap in x-direction*

With an 'inside-outside' polygon test, you can perform tests 2 and 3. Then you can substitute the coordinates for all the vertices of *S* into the plane equation for the overlapping surface and check the sign of the result. If the plane equation are set up so that the outside of the surface is toward the viewing position, then *S* is behind *S*- if all the vertices of *S* are 'inside' *S',* as shown in Figure 5.29:

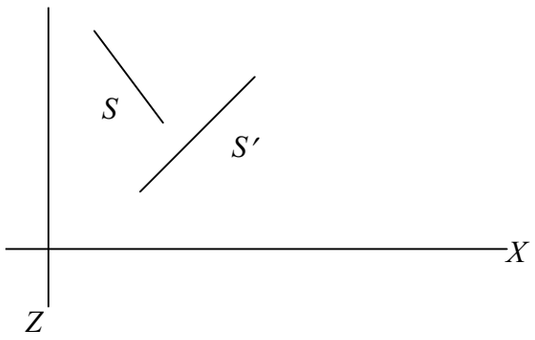


Fig. 5.29 *Surface S is Computed behind the Overlapped Surface S′*

If all vertices of S are 'outside' of *S*′, then *S′* is completely in front of S. Figure 5.10 shows an overlapping surface *S*′ which is completely in front of S, but surface S is not completely 'inside' *S*′ (test 2 failed).

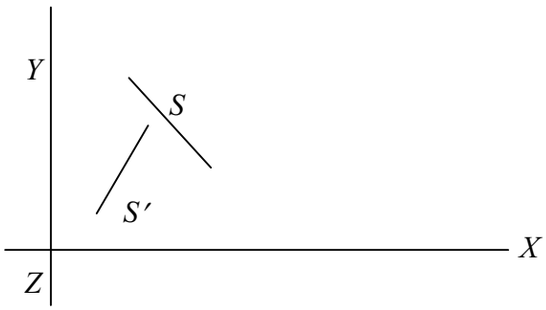


Fig. 5.10 *The Overlapping Surface S′ is Completely Outside Surface S*

If tests 1 to 3 have failed, then you can try test 4, which checks for intersections between the bounding edges of the two surfaces by using line equations in the *xy* plane. As shown in Figure 5.31, the two surfaces can or can not intersect although their coordinate extents overlap in the *x*, *y* and *z* directions.

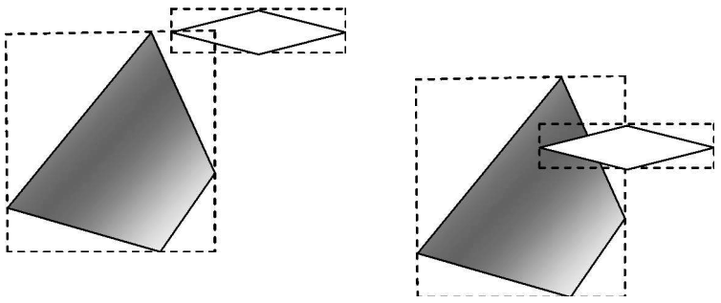


Fig. 5.31 *Two surfaces having Overlapping Bounding Rectangles in the xy-plane*

In case all the four tests fail with a particular overlapping surface S', then you should interchange surfaces S and S' in the sorted list. Figure 5.32 shows the example of two surfaces that would be reordered with this procedure.

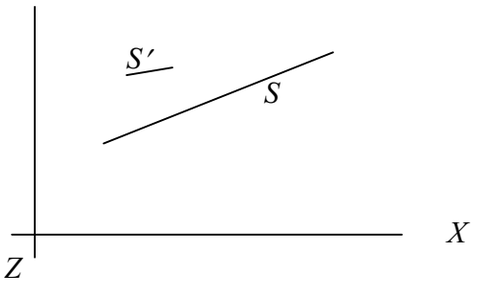


Fig. 5.32 *The Surface S with more Depth Obscures Surface S′*

At this point, you still do not know for certain, that you have found the farthest surface from the view plane. Figure 5.32 shows a situation in which you would first interchange S and S'.

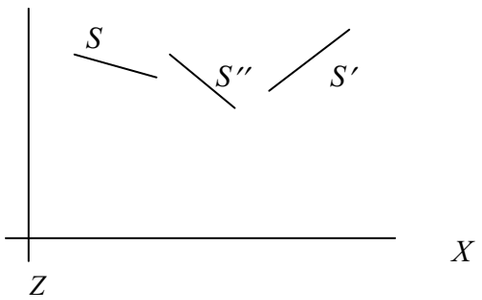


Fig. 5.33 *Three Surfaces entered into Sorted List (S, S′, S″) of Surfaces*

But since S'' obscures part of S', you need to interchange S'' and S' so that you can get the three surfaces into the correct depth order. So, you need to repeat the testing process for each surface that is reordered in the list.

5.5.1 Painter's algorithm

Painter's algorithm was introduced by **Hewells** in **1972**. The techniques used by these algorithms are **image space** and **object space**. The name of this algorithm is Painter's because its working is like a painter who creates an **oil 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.

The following are the steps of this algorithm:

1. Sorting of the various surfaces which is on the basis of their decreasing depth or we can say the largest value of z.

2. Now scanning to convert the various surfaces which is in the order starting with the surface which has greatest depth.
3. Comparing is to be done on the basis of various overlapping surfaces so that the user will determine which surface is to be kept visible.
4. In the refresh buffer enter the intensity value for the determined surface i.e. the surface which is determined to be visible.
5. The above process is going to be repeat for all the available surfaces.
6. However, if the overlapping is observed, then there is the need of the further tests like Z-Buffer or Depth-Buffer method.

Check Your Progress

1. What is vanishing point?
2. What do you mean by a parallel projection?
3. Define the term viewpoint.
4. What is A-buffer method?
5. State the basic functions of the depth-sorting method.

5.6 BINARY SPACE PARTITION

Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements. It can be seen as a generalisation of other spatial tree structures, such as *k*-d trees and quadtrees, one where hyperplanes that partition the space may have any orientation, rather than being aligned with the coordinate axes as they are in *k*-d trees or quadtrees. When used in computer graphics to render scenes composed of planar polygons, the partitioning planes are frequently chosen to coincide with the planes defined by polygons in the scene.

The specific choice of partitioning plane and criterion for terminating the partitioning process varies depending on the purpose of the BSP tree. For example, in computer graphics rendering, the scene is divided until each node of the BSP tree contains only polygons that can be rendered in arbitrary order. When back-face culling is used, each node therefore contains a convex set of polygons, whereas when rendering double-sided polygons, each node of the BSP tree contains only polygons in a single plane. In collision detection or ray tracing, a scene may be divided up into primitives on which collision or ray intersection tests are straightforward.

Binary space partitioning arose from the computer graphics need to rapidly draw three-dimensional scenes composed of polygons. A simple way to draw such scenes is the painter's algorithm, which produces polygons in order of distance from the viewer, back to front, painting over the background and previous polygons with each closer object. This approach has two disadvantages: time required to sort polygons in back to front order, and the possibility of errors in overlapping polygons. Fuchs and co-authors showed that constructing a BSP tree solved both of these problems by providing a rapid method of sorting polygons with respect to

a given viewpoint (linear in the number of polygons in the scene) and by subdividing overlapping polygons to avoid errors that can occur with the painter's algorithm. A disadvantage of binary space partitioning is that generating a BSP tree can be time-consuming. Typically, it is therefore performed once on static geometry, as a pre-calculation step, prior to rendering or other realtime operations on a scene. The expense of constructing a BSP tree makes it difficult and inefficient to directly implement moving objects into a tree.

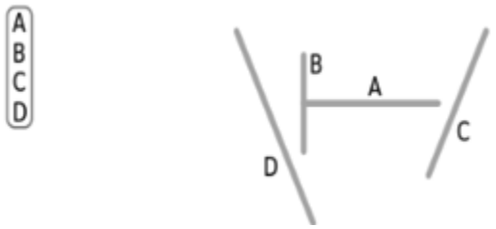
BSP trees are often used by 3D video games, particularly first-person shooters and those with indoor environments. Game engines using BSP trees include the Doom (id Tech 1), Quake (id Tech 2 variant), GoldSrc and Source engines. In them, BSP trees containing the static geometry of a scene are often used together with a Z-buffer, to correctly merge movable objects such as doors and characters onto the background scene. While binary space partitioning provides a convenient way to store and retrieve spatial information about polygons in a scene, it does not solve the problem of visible surface determination.

The canonical use of a BSP tree is for rendering polygons (that are double-sided, that is, without back-face culling) with the painter's algorithm. Each polygon is designated with a front side and a back side which could be chosen arbitrarily and only affects the structure of the tree but not the required result. Such a tree is constructed from an unsorted list of all the polygons in a scene. The recursive algorithm for construction of a BSP tree from that list of polygons is:

1. Choose a polygon *P* from the list.
2. Make a node *N* in the BSP tree, and add *P* to the list of polygons at that node.
3. For each other polygon in the list:
   - If that polygon is wholly in front of the plane containing *P*, move that polygon to the list of nodes in front of *P*.
   - If that polygon is wholly behind the plane containing *P*, move that polygon to the list of nodes behind *P*.
   - If that polygon is intersected by the plane containing *P*, split it into two polygons and move them to the respective lists of polygons behind and in front of *P*.
   - If that polygon lies in the plane containing *P*, add it to the list of polygons at node *N*.
4. Apply this algorithm to the list of polygons in front of *P*.
5. Apply this algorithm to the list of polygons behind *P*.

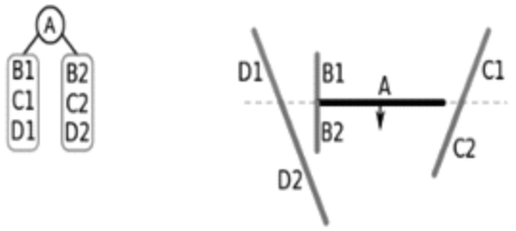
The following diagram illustrates the use of this algorithm in converting a list of lines or polygons into a BSP tree. At each of the eight steps (i.-viii.), the algorithm above is applied to a list of lines, and one new node is added to the tree.

Start with a list of lines, (or in 3D, polygons) making up the scene. In the tree diagrams, lists are denoted by rounded rectangles and nodes in the BSP tree by circles. In the spatial diagram of the lines, the direction chosen to be the 'front' of a line is denoted by an arrow.

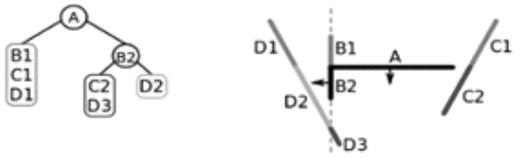


i. Following the steps of the algorithm above,

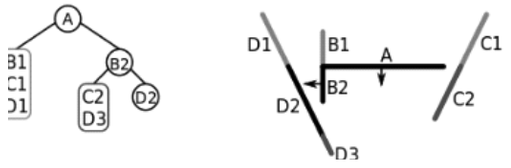
1. We choose a line, A, from the list and,
2. add it to a node.
3. We split the remaining lines in the list into those in front of A (i.e. B2, C2, D2), and those behind (B1, C1, D1).
4. We first process the lines in front of A (in steps ii–v),...
5. followed by those behind (in steps vi–vii).



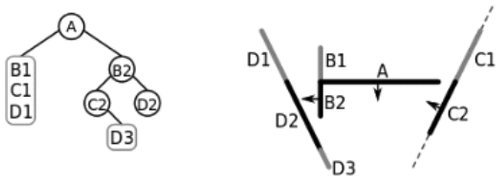
ii. We now apply the algorithm to the list of lines in front of A (containing B2, C2, D2). We choose a line, B2, add it to a node and split the rest of the list into those lines that are in front of B2 (D2), and those that are behind it (C2, D3).



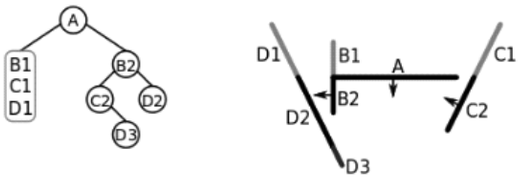
iii. Choose a line, D2, from the list of lines in front of B2 and A. It is the only line in the list, so after adding it to a node, nothing further needs to be done.



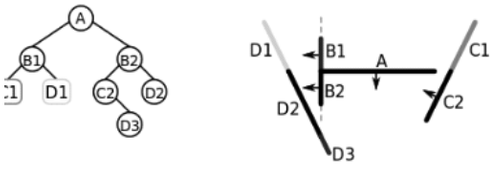
iv. We are done with the lines in front of B2, so consider the lines behind B2 (C2 and D3). Choose one of these (C2), add it to a node, and put the other line in the list (D3) into the list of lines in front of C2.



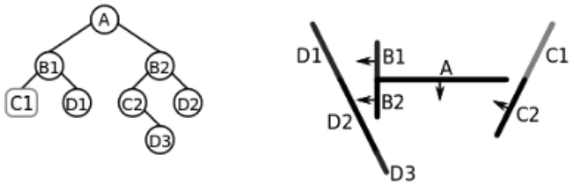
v. Now look at the list of lines in front of C2. There is only one line (D3), so add this to a node and continue.



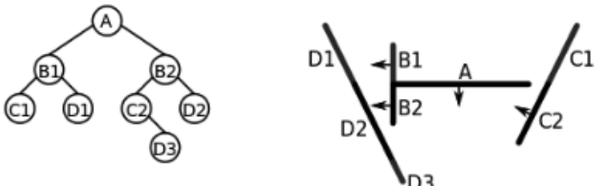
vi. We have now added all of the lines in front of A to the BSP tree, so we now start on the list of lines behind A. Choosing a line (B1) from this list, we add B1 to a node and split the remainder of the list into lines in front of B1 (i.e. D1), and lines behind B1 (i.e. C1).



vii. Processing first the list of lines in front of B1, D1 is the only line in this list, so add this to a node and continue.



viii. Looking next at the list of lines behind B1, the only line in this list is C1, so add this to a node, and the BSP tree is complete.



The final number of polygons or lines in a tree is often larger (sometimes much larger) than the original list, since lines or polygons that cross the partitioning plane must be split into two. It is desirable to minimize this increase, but also to maintain reasonable balance in the final tree. The choice of which polygon or line is used as a partitioning plane (in step 1 of the algorithm) is therefore important in creating an efficient BSP tree.

5.7 CUBIC BEZIER CURVE

Bezier curve is the spline approximation method that was developed by the French engineer Pierre Bezier to design automobile bodies. The Bezier spline has a number of properties that makes it highly useful and convenient for surface and curve design. Bezier curves and surfaces are easy to implement. For these reasons, Bezier splines are widely available in various graphics packages like CAD systems, and in assorted drawing and painting softwares.

Practically, a section of a Bezier curve can be fitted to any number of control points. The number of control points to be approximated and their relative position determines the degree of the Bezier polynomial. Similar to the interpolation splines, a Bezier curve can be specified with boundary conditions, with blending functions, or with a characterizing matrix. In case of general Bezier curves, the blending-function specification is the most convenient function. Suppose we are given $n+1$ control-point positions: $p_k = (x_k, y_k, z_k)$, with k varying from 0 to n. These coordinate points can be blended to produce the following position vector $P(u)$, which describes the path of an approximating Bezier polynomial function between P_0 and P_n:

$$P(u) = \sum_{k=0}^{n} P_k . BEZ_{k,n}(u), \qquad 0 \le u \le 1 \qquad ...(5.2)$$

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

$$BEZ_{k,n}(u) = C(n,k)u^k (1-u)^{n-k} \qquad ...(5.3)$$

where the $C(n, k)$ are the binomial coefficients defined as follows:

$$C(n.k) = \frac{n!}{k!(n-k)!} \qquad ...(5.4)$$

Equivalently, we can define Bezier blending functions with the recursive calculation as follows:

$$BEZ_{k,n}(u) = (1-u)BEZ_{k,n-1}(u) + uBEZ_{k-1,n-1}(u), \qquad n > k \ge 1 \qquad ...(5.5)$$

with $BEZ_{k,k} = u^k$, and $BEZ_{0,k} = (1-u)^k$. The vector equation 5.2 represents a set of three parametric equations for the individual curve coordinate as follows:

$$x(u) = \sum_{k=0}^{n} x_k BEZ_{k,n}(u) \qquad ...(5.6)$$

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

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

As a rule, a Bezier curve is a polynomial of degree one less than the number of control points used: three points generate a parabola, four points a cubic curve, and so forth. Figure 5.34 demonstrates the appearance of Bezier curves for various

selections of control points in the *xy*-plane (z = 0). With certain control-point placements, however, we obtain degenerate Bezier polynomials. For example, a Bezier curve generated with three collinear control points is a straight-line segment. And a set of control points that are all at the same coordinate position produces a Bezier 'curve' that is a single point. Bezier curves are commonly applied in painting and drawing packages, as well as CAD systems, since they are easy to implement and they are reasonably powerful in a curve design. Efficient methods to determine coordinate positions along a Bezier curve can be set up using recursive calculations. For example, successive binomial coefficients can be calculated as follows:

$$C(n,k) = \frac{n-k+1}{k} C(n,k-1)$$

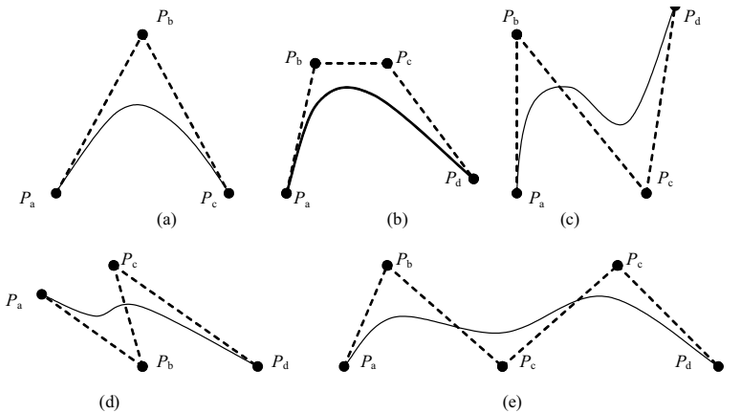


Fig. 5.34 *Examples of Two-Dimensional Bezier Curves Generated from Three, Four and Five Control Points. Dashed Lines Connect the Control-Point Positions*

The following example program illustrates a method for generating Bezier curves:

```
#include<stdio.h>
#include<graphics.h>
#include<bios.h>
#include<stdlib.h>
typedef double CoArr[4];
char buf[20];
//function definition for drawing Bezier curve
void BezierCurve(CoArr x, CoArr y, int n, int value)
{
    int i, a, b;
    double delta = 1.0/n;
    double t = 0, T;

setcolor(8); //defining color for curve
    for(i=0;i<4;i++)
       line(x[i],y[i],x[(i+1)%4],y[(i+1)%4]);
    for(i = 0; i<n; i++)
    {
    t = t + delta;
```

NOTES

```
    T = 1-t;
    a = x[0]*T*T*T+3*t*T*T*x[1]+3*t*t*T*x[2]+t*t*t*x[3];
    b = y[0]*T*T*T+3*t*T*T*y[1]+3*t*t*T*y[2]+t*t*t*y[3];
       putpixel(a,b,value);
    }
}
main()
{
    CoArr x, y;
    int val;
int gd = DETECT, gmode;
    int key;
    int flag = 1;
    double *movey;
double *movex;
char *str;
 x[0]=100; y[0]=100;
 x[3]=400; y[3]=100;
 x[1]=200; y[1]=50;
 x[2]=300; y[2]=50;
    initgraph(&gd,&gmode,"c:\\tc\\bgi");
//function calling for drawing Bezier curve
    BezierCurve(x,y,1000,15);
    movex = &x[1];
    movey = &y[1];
    while((key = bioskey(0))!=283)
    {
       cleardevice();
       switch(key)
       {
        case 3849:
          if(flag == 0)
          {
             flag = 1;
             movex = &x[0]; movey = &y[0];
          }
          else if(flag == 1)
          {
             flag = 2;
             movex=&x[1]; movey = &y[1];
          }
          else if(flag == 2)
          {
             flag = 3;
             movex = &x[2]; movey = &y[2];
          }
          else if(flag == 3)
          {
```

NOTES

```
        flag = 0;
        movex = &x[3]; movey = &y[3];
      }
      break;
    case 18432:
      *movey =*movey - 5; break;
    case 20480:
      *movey =*movey + 5; break;
    case 19200:
      *movex = *movex - 5; break;
    case 19712:
      *movex =*movex + 5; break;
    }
    if(flag ==0)
    {
      setcolor(4);
      circle(x[3],y[3],3);
    }
    else if(flag == 1)
    {
      setcolor(4);
      circle(x[0],y[0],3);
    }
    else if(flag == 2)
    {
      setcolor(GREEN);
      circle(x[1],y[1],3);
    }
    else if(flag == 3)
    {
      setcolor(GREEN);
      circle(x[2],y[2],3);
    }
    value = (int)x[1];
    itoa(val,str,10);
    setcolor(4);
    outtextxy(50,50,str);
    value = (int)y[1];
    itoa(val,str,10);
    setcolor(4);
    outtextxy(80,50,str);
    value = (int)x[2];
    itoa(val,str,10);
    setcolor(4);
    outtextxy(50,80,str);
    value = (int)y[2];
    itoa(val,str,10);
    setcolor(4);
    outtextxy(80,80,str);
    BezierCurve(x,y,1000,15);
}
```

```
return 0;
}
```

Properties of Bezier Curves

NOTES

A very useful property of a Bezier curve is that it always passes through the first and last control points. That is, the boundary conditions at the two ends of the curve are as follows:

$$P(0) = P_0 \qquad ...(5.7)$$

$$P(1) = P_n$$

These are the values of the parametric first derivatives of a Bezier curve at the endpoints. Three-dimensional points can he calculated from control-point ordinates as follows:

$$P'(0) - np_0 + np_1 \qquad ...(5.8)$$

$$P'(1) - np_{n-1} + np_n$$

Thus, the slope at the starting of the curve is across the line joining the first two control points, and the slope at the end of the curve is across the line joining the last two control points. In the same way, the parametric second derivatives of a Bezier curve at the endpoints are calculated as follows:

$$P''(0) = n(n-1)\{(p_2 - p_1) - (p_1 - p_0) \qquad ...(5.9)$$

$$P''(1) = n(n-1)\{(p_{n-2} - p_{n-1}) - (p_{n-1} - p_n)$$

Another important property of any Bezier curve is that it lies within the convex hull (convex polygon boundary) of the control points. This follows from the properties of Bezier blending functions: they are all positive and their sum is always 1.

$$\sum_{k=0}^{n} BEZ_{k,n}(u) = 1 \qquad ...(5.10)$$

so that any curve position is simply the weighted sum of the control-point positions. The convex-hull property for a Bezier curve ensures that the polynomial smoothly follows the control points without erratic oscillations.

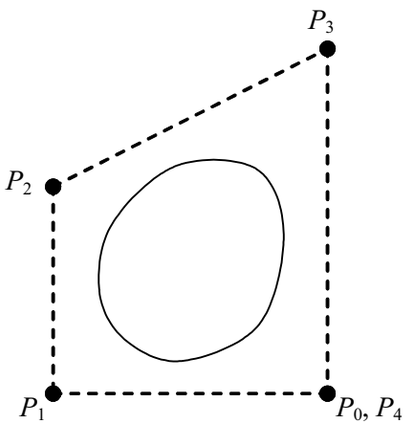


Fig. 5.35 *A Closed Bezier Curve Generated with the Same Starting and End Point*

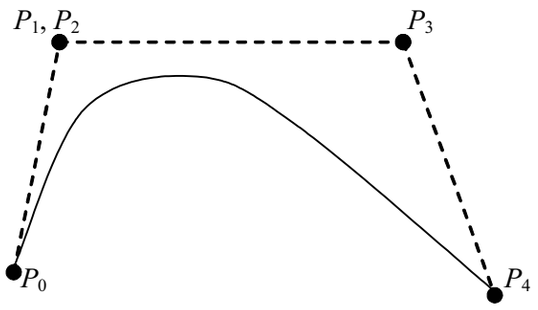


Fig. 5.36 *A Bezier Curve is made to Pass Closer to a Given Coordinate by Assigning Multiple (P_1, P_2) Control Points*

Design Techniques Using Bezier Curves

Closed Bezier curves are generated by specifying the first and last control points at the same position, as in the example shown in Figure 5.35. Also, specifying multiple control points at a single coordinate position gives more weight to that position. In Figure 5.36, a single coordinate position is input as two control points, and the resulting curve is pulled nearer to this position. A Bezier curve can fit any number of control points, but this requires the calculation of polynomial functions of a higher degree. When complicated curves are to be generated, they can be formed by piecing several Bezier sections of lower degree together. Piecing together smaller parts also gives us better control over the shape of the curve in small regions. Since Bezier curves pass through endpoints, it is easy to match curve sections (zero order continuity). Additionally, Bezier curves have the important property that the tangent to the curve at an endpoint is across the line joining that control point to the adjacent control point. Therefore, to obtain first-order continuity between curve sections, we can pick control points of a new section that will be along the same straight line as control points of the previous section this is shown in Figure 5.37. We obtain C^1 continuity by choosing the first control point of the new section as the last control point of the previous section and by positioning the second control point of the new section at position P'_0.

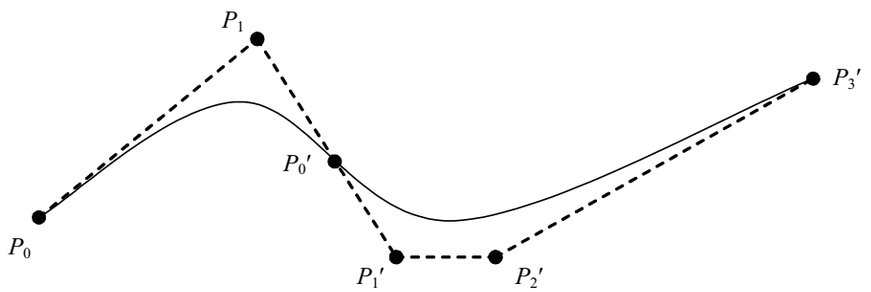


Fig. 5.37 *Piecewise Approximation Curve formed with Two Bezier Sections, Zero-Order and First-Order Continuity are Attained between Curve Sections by setting $P_0' = P_2$ and by making Points P_1, P_2 and P_1' Collinear*

Thus, the three control points are collinear and equally spaced. We obtain continuity between two Bezier sections by calculating the position of the third control point of a new section in terms of the positions of the last three control points of the previous section as. Requiring second-order continuity of Bezier

curve sections can be unnecessarily restrictive. This is especially true with cubic curves, which have only four control points per section. In this case, second-order continuity fixes the position of the first three control points and leaves us only one point that we can use to adjust the shape of the curve segment.

The Cubic Bezier Curves

Many graphics packages make available only cubic spline functions. This facilitates reasonable design flexibility and avoids the increased calculations required with higher-order polynomials. Cubic Bezier curves can be generated with four control points. The four blending functions for cubic Bezier curves that are obtained by substituting $n = 3$, for $k = 0, 1, 2, 3$ into equation 5.3 are as follows:

$$\begin{aligned} BEZ_{0,3}(u) &= (1-u)^3 \\ BEZ_{1,3}(u) &= 3u(1-u)^2 \\ BEZ_{2,3}(u) &= 3u^2(1-u) \\ BEZ_{3,3}(u) &= u^3 \end{aligned} \qquad ...(5.11)$$

The form of the blending functions determine how the control points influence the shape of the curve for values of parameter u over the range from 0 to 1. At $u = 0$, the only non-zero blending function is $BEZ_{0,3}$ which has the value 1. At $u = 1$, the only non-zero function is $BEZ_{3,3}$ with a value of 1 at that point. Thus, the cubic Bezier curve will always pass through control points p_0 and p_3. The other functions, $BEZ_{1,3}$ and $BEZ_{2,3}$, influence the shape of the curve, at intermediate values of parameter u, so that the resulting curve tends toward the points p_1, and p_2. Blending function $BEZ_{1,3}$ is maximum at $u = \frac{1}{3}$, and $BEZ_{2,3}$ is maximum at $u = \frac{2}{3}$.

Bezier Surfaces

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

$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u) \qquad ...(5.12)$$

with $p_{j,k}$ specifying the location of the by control points. Figure 5.38 illustrates two Bezier surface plots in which the dashed lines connect the control points. The control points at the surface are connected by dashed lines, and the solid lines show curves of constant values u and v. Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant v are plotted similarly.

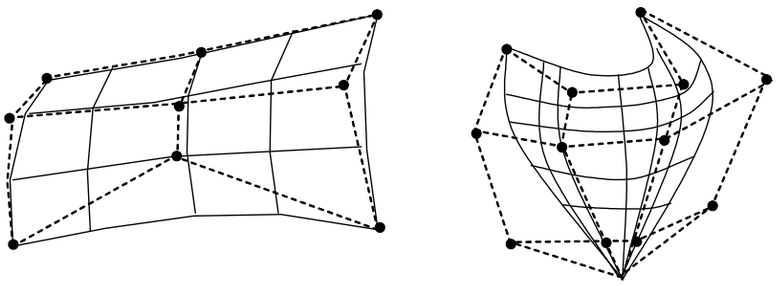


Fig. 5.38 *Bezier Surfaces Constructed for (a) m = 3, n = 3, and (b) m = 4, n = 4*

Bezier surfaces have the same properties as Bezier curves, and they provide a convenient method for interactive design applications. For each surface patch, we can select a mesh of control points in the *xy* 'ground' plane. Then we choose elevations above the ground plane for the z-coordinate values of the control points. Patches can then be pieced together using boundary constraints. Figure 5.39 illustrates a surface formed with two Bezier sections. The dashed lines in this figure connect specify control points. A smooth transition from one section to the other is assured by establishing both zero-order and first-order continuity at the boundary line. Zero-order continuity is obtained by matching control points at the boundary. First-order continuity is obtained by choosing control points along a straight line across the boundary and by maintaining a constant ratio of collinear line segments for each set of specified control points across section boundaries.

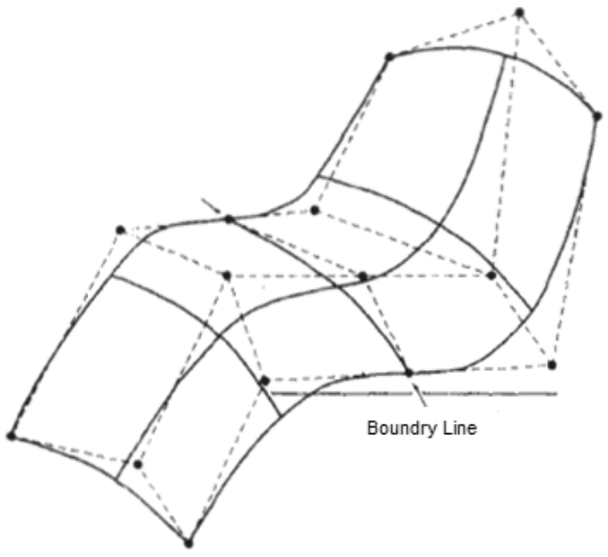


Fig. 5.39 *An Illustration of a Composite Bezier Surface Constructed with Two Bezier Sections, joined at the Indicated Boundary Line*

Example 5.7: A cubic Bezier curve is defined over the control points (1, 1), (2, 3), (4, 4) and (6, 1). Calculate the parametric mid points of this curve and show that its gradient *dy*/*dx* is 1/7.

Solution: The blending functions for these control points are as follows:

$$B_{0,3}(u) = (1-u)^3$$

NOTES

$$B_{1,3}(u) = 3u(1-u)^2$$

$$B_{2,3}(u) = 3u^2(1-u)$$

$$B_{3,3}(u) = u^3$$

The *x*-coordinate of the Bezier curve are as follows:

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

$$= (1-u)^3 + 6u(1-u)^2 + 12u(1-u) + 6u^3$$

Similarly the, y-coordinates of the Bezier curve are as follows:

$$y(u) = (1-u)^3 + 9u(1-u)^2 + 12u(1-u) + u^3$$

The parameter u lies between 0 and 1. Therefore the midpoint of the curve is u = ½. By putting this value in $x(u)$ and $y(u)$ equations we get parametric midpoint coordinates as (27/8, 23/8).

By differentiating the $x(u)$ equation with respect to u we get,

$$\frac{dx}{du} = -3(1-u)^2 - 12u(1-u) + 6(1-u)^2 - 12u^2 + 24u(1-u) + 18u^2$$

Similarly by differentiating the $y(u)$ equation with respect to u we get,

$$\frac{dy}{du} = -3(1-u)^2 u(1-u) + 9(1-u) + 9(1-u)^2 - 12u^2 + 24(1-u)u + 3u^2$$

The gradient at any point is given by,

$$\left(\frac{dy}{dx}\right)_u = \frac{\left(\frac{dy}{du}\right)_u}{\left(\frac{dx}{du}\right)_u}$$

The gradient at midpoint for u = ½ is,

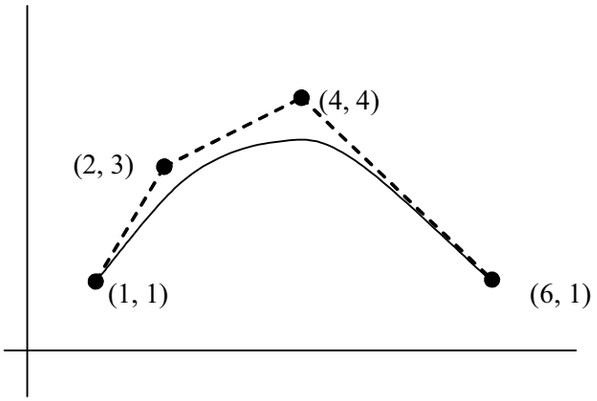
$$\left(\frac{dy}{du}\right)_{u=1/2} = \frac{3}{4}$$

and

$$\left(\frac{dx}{du}\right)_{u=1/2} = \frac{21}{4}$$

Then we have $\left(\frac{dy}{dx}\right)_{u=1/2} = \frac{1}{7}$ which is equal to the given gradient.

The cubic Bezier curve is given in Figure given below.



5.8 MULTIMEDIA

Multimedia refers to a mixture of interactive media or data types, predominantly text, graphics, audio and video simultaneously delivered by a computer. Actually a multimedia system implies a combination of specified hardware components with certain minimum capabilities and compatible software that has an interactive interface studded with different media elements. The difference between a multimedia system and a television (where we also get simultaneous presentation of multiple media) is that in television, the delivery is not interactive and the user cannot control the way things happen whereas a Personal Computer (PC) multimedia system allows the user to control the elements that are delivered with a varied degree of navigational freedom through the linked elements. As name implies, it can be defined as follows:

'*Multimedia is the integration of multiple forms of media*'

Though basically multimedia presentation is dependent on the processing power and data storage capacity of the computer, some basic hardware components that make a complete multimedia system are as follows:

- Devices, such as keyboard, mouse, joystick, touch screen by which the user can interact with the system.
- A high resolution screen and graphics accelerator card that can output good quality still images, video clips as well as animations, text and graphics.
- Speakers for speech and music output.
- Microphone for audio recording.
- Sound card and video grabber card to capture, digitize, and edit audio and video material.
- Compact Disk Read Only Memory (CD ROM) drive to play back pre-recorded source material.

5.8.1 Multimedia Application

In this section, you will see some of the major applications of multimedia. Multimedia is one of the most fascinating and fastest growing areas in the field of information technology. Text, pictures, animation, movies and sound — all these varied media are seamlessly blended, resulting in simple slide shows to dazzling, interactive presentations. However, before the advent of computers, multimedia

projects were difficult to put together. Computers enable to combine the media and can be stored for reuse. Multimedia is widely used in following areas:

1. **Virtual reality:** It is an artificial environment created with computer hardware and software, presented to the user in such a way that it appears and seems real. In virtual reality, the computer controls three of the five senses. Virtual reality systems require extremely expensive hardware and software and are confined mostly to research laboratories.
2. **Entertainment:** Multimedia technology is used in entertainment industry for creating real life like games for inserting sound, graphics, animation, etc. Many computer games even focus on educational entertainment providing some sort of learning in addition to play.
3. **Business Communications:** The business communications includes communications related to employee, product promotions, customer information and reports for investors. Multimedia technology improves the quality of presentation. A variety of options are available for business presentation such as, keyboard interactive station, interactive touch screen kiosk, sequential playback and continuous loop playback. Among these the interactive kiosks are considered the most handy allowing the user direct and active control over the information. Keyboard stations give visual backups to the verbal information that the speaker is giving by allowing the employee to step through the presentation with a customer. This presentation waits for the speaker to ask all the questions he wants; meanwhile, the presentation does not run's ahead.
4. **Knowledge Transfer:** This application implies the transfer of information with the facility of retention.
5. **Public Access:** This area includes facilities like tourist information system, railway time table enquiry, etc.
6. **Publishing Industry:** The publishing industry earlier had only printed text. But today due to the availability of multimedia the printed text is available online. For example, the videos of news are readily available along with the text. Encyclopedias have audio and video associated with them making the education more entertaining and easily understandable.
7. **Education:** Multimedia is used to produce Computer Based Training (CBT) courses and reference books like encyclopedia and almanacs. A CBT allows the user to go through a series of presentations, text about a particular topic and associated illustrations in various information formats. Video and audio capture of lectures has become a common practice to produce e-learning content. Simulations allow exploring experiments which would be too expensive or too dangerous to be conducted physically by students. Encyclopedias have audio and video associated with them making the education more entertaining and easily understandable.
8. **Communication Technology:** The services in this area include basic television services, interactive entertainment, digital audio, video on demand, home shopping through email, financial transactions using e-commerce, interactive single and multiuser games, digital multimedia libraries, electronic versions of newspapers, etc. Cable TV and telephone companies, dot com companies, publishing industry, etc., provide the main infrastructure for these facilities. You can exchange text, image and video information in a computer-based multimedia conference.

9. **Engineering:** Software engineers may use multimedia in computer simulations for anything from entertainment to training such as military or industrial training. Multimedia for software interfaces are often done as collaboration between creative professionals and software engineers.
10. **Commercial:** Much of the electronic old and new media used by commercial artists is multimedia. Exciting presentations are used to grab and keep attention in advertising. Business to business and interoffice communications are often developed by creative services firms for advanced multimedia presentations beyond simple slide shows to sell ideas or liven-up training.
11. **Mathematical and scientific research:** In mathematical and scientific research, multimedia is mainly used for modeling and simulation. For example, a scientist can look at a molecular model of a particular substance and manipulate it to arrive at a new substance.
12. **Medicine:** In Medicine, doctors can get trained by looking at a virtual surgery or they can simulate how the human body is affected by diseases spread by viruses and bacteria and then develop techniques to prevent it.

5.8.2 Multimedia Hardware

The physical components of a computer are called hardware. A set of programs or instructions written for a computer is called software. Following hardware and software requirements are essential for creating multimedia applications:

Table 5.1 summarizes the list of medium tools to develop the multimedia applications.

Table 5.1 *Medium Tools for Multimedia Applications*

Medium Tools	Required Software
For text	Microsoft Word and Corel WordPerfect.
For graphics (vectors)	CorelDRAW, Adobe Illustrator, Macromedia Fireworks, Adobe ImageReady and Macromedia Flash.
For image (bitmap)	Adobe Photoshop, Jasc Paint Shop Pro and Macromedia Fireworks.
For audio	Sony Sound Edit Pro, Sony Sound Forge for Windows and Sony Acid and Cakewalk products.
For synthetic video (animation)	AutoDesk AutoCAD, Discreet 3-D Studio (MAX), Virtus 3-D Website Builder, Macromedia Flash and Electric Image Amorphium Pro alias Maya.
For capturing video	Adobe Premiere, Avid, Media products, Ulead Media Studio Pro, Microsoft MovieMaker and Apple iMovie.
For authoring systems	Macromedia Director, Macromedia Dreamweaver, Click2learn Toolbook, Microsoft FrontPage, Adobe Page Mill and Microsoft PowerPoint with Producer.

Table 5.2 summarizes the list of software and their function used in creating multimedia applications.

Table 5.2 *Software Used in Multimedia and their Function*

Software	Function
Adobe Macromedia Authorware	This software uses visual rich media solution to create online virtual multimedia applications.
Chrome Library	This software includes free models and samples which are used in the 3-D behaviours of Chrome library set. This set contains 3-D behaviour of 60 Shockwave which has been designed for 3-D graphic designers. It is used to set up the scenes in fast manner.
Lingo	Lingo software language comes with integrated package which includes expressing the optimization models. This software provides animation, sound, scripting and shockwave 3-D for Lingo language.
Shockwave Demo	This software represents the complete operation for audio data that pushes up the sound object with audio engine.
Adobe Director	This software contains audio operations with 5.1 channel and the operations involve sound mixing with Digital Signal Processor (DSP). This operation filters the Motion Picture Experts Group (MPEG) Layer 4 (MP4), H.264, F4V, Flash Video (FLV) video streaming formats. The powerful 3-D format supports the Google SketchUp importer and byte array datatype.
VRML tools	These tools basically refer to various software, such as Chisel (Java based optimization tool), Seamless3D (open source 3-D modelling software that exports and imports VRML files), SwirlX3D editor (visual authoring system supports VRML using graphical environment), TriVista technology (supports 3-D ImageScene and 3-D ImageCube tools for modifying and displaying the photographs), VizUp (3-D models supporting VRML and other 3-D tool formats).
Digital ArtForms	This software is used in 3-D interaction, immersion and real time SmartScene.
Internet Explorer or Mozilla Firefox	These two executable files refer to prime browsers which provide lease line connection to high broadband services for Internet connection.
Adobe FantaMorph	This powerful software provides fantastic image to morph and warp the movie in real time 3-D. It accelerates hardware and renders the speed easily at the rate of frame per second (fps). This high speed makes the final play in real time without exporting the file from other application. This software supports Bit Map Picture (BMP), Joint Photographic Experts Group (JPEG), Portable Network Graphics (PNG), Truevision Graphics Adapter (TGA), Personal Computer eXchange (PCX), Tagged Image File Format (TIFF) and 32-bit alpha formats. The exported image sequences supported by this software are Adobe Flash Player, animated GIF, image sequence, etc. The image can be cropped, rotated, flipped and adjusted with the help of this software.
Microsoft DirectX	This software is a set of Application Programming Interfaces (APIs) which allow applications to gain access directly to a system's multimedia hardware.

A plug-in is a small program which extends the capabilities of another program. This program is referred to as a player for extending the animations, graphics and music capabilities of the Web browser. If the Web page is encountered, you need a special plug-in for processing it smoothly. Many plug-ins are small programs and are easy to download. Table 5.3 shows the popular applications of plug-ins used in creating multimedia applications.

Table 5.3 *Plug-ins Applications and their Function*

Plug-in Applications	Functions
Beatnik	Beatnik supports Rich Music Format (RMF), Musical Instrument Digital Interface (MIDI), Music Module (MOD), Audio Interchange File Format (AIFF), Waveform Audio Format (WAV) and MPEG1 or MPEG2 Audio Layer III (MP3). Both Windows and Mac versions are available for Beatnik.
Acrobat 4.0	The Portable Document Format (PDF) files can be viewed easily and navigated too in almost all the leading Web browsers. In capturing plug-in, any TIFF image can be scanned to PDF format or printed document can be scanned using Optical Character Reader (OCR) to turn it into PDF format.
QuickTime	QuickTime delivers the multimedia documents, such as videos, audios, soundtracks of MIDI soundtracks, 3-D animation and virtual reality. This plug-in enables contents of QuickTime and QuickTime VR for viewing directly within a browser.
Shockwave	The multimedia files can be created using Director of Macromedia. This plug-in has compatibility with Internet Explorer.
RealPlayer	The RealPlayer is a live player for RealVideo and RealAudio on demand and functions with no delay in download.
VivoActive Player	This delivers audio and video on demand from any website and offers VivoActive content.
Netscape Browser plug-ins	This plug-in is organized into various categories, such as audio and video, business and utilities, image viewers, 3-D and animation and presentations.

Following hardware are needed for developing the multimedia applications:

- **Media Control Interface:** The Media Control Interface (MCI) provides applications created for Windows operating system with device independent capabilities for controlling media devices, such as audio hardware, videodisc players and animation players. This interface works with MCI device drivers to interpret and run MCI commands, such as PAUSE, PLAY and STOP operations. MCI uses the same command to begin playback of a waveform audio file, a videodisc track and an animation sequence. It also provides extended commands for using particular device types with unique capabilities, such as a frame based time format used in animation.
- **Windows Driver Model:** Windows Driver Model (WDM) audio class architecture performs all audio processing in kernel mode. A Universal Disk Format (UDF) DVD file reader, splitter and navigator provides access for DirectShow clients to separate video and audio streams. WDM streaming is directly based on the DirectShow model of user mode filters. In the context of WDM streaming, a filter represents a kernel mode driver. This allows data streams to be passed through the operating system and to move at a faster pace because CPU cycles are not used to pass the information back and forth between the user mode and kernel levels.
- **Sound Blaster Card:** Sound blaster cards are used in high technology in gaming and videos. There are several models including Peripheral Component Interconnect (PCI) Express, Audigy SE and Wireless Audio. PCI Express cards require a motherboard which allows PCI Express components. For the installation of the regular Sound blaster cards, a motherboard with normal

PCI slots is a requirement. The wireless audio installs in desktop as well as laptop computers with a Universal Serial Bus (USB) connection.

- **Musical Instrument Digital Interface:** The Musical Instrument Digital Interface (MIDI) was developed to provide a standard way of interfacing music controllers, such as keyboards to sound generators and synthesizers and drum machines. MIDI is a half duplex 5mA current loop which carries an 8-bit serial data stream at a bit rate of 31.25 kilobaud. Measurement mA stands for milli-Amp. It communicates over 16 channels allowing up to 16 MIDI instruments to be played from one interface. When a key is pressed on a keyboard, a message 'Note on message down' is sent to the MIDI cable instructing the receiving device to play a note. The message consists of three elements known as a status byte, a note number and a velocity value. The status byte contains information about the event type and the channel it is to be sent on. The note number describes the key that was pressed, let us say 'C' and the velocity value indicates the force at which the key was struck. The receiving device will play this note until a note off message is received containing the same data.
- **3-D Audio:** 3-D audio systems refer to a hardware interface which is implemented with a pair of conventional loudspeakers. The general approach to a 3-D audio system is to reconstruct the acoustic pressure at the listener's ears that would result from the natural listening situation to be simulated. 3-D audio technology uses following technologies:
  - **Volumetric Sound Sources:** When developers define an audio file to a sound source, the sound source must have a location so that it can be rendered in relation to the listener.
  - **MP3 Playback:** Audio streams for 3-D audio have to be Waveform Audio Format (WAV) files. MP3 files can be used, thereby both reducing their associated storage space and increasing their quality.
  - **Reverb:** An Aureal's geometric reverb will work on Vortex2 cards as well as automatically translating to Environmental Audio Extensions (EAX) or I3DL2 if a sound card does not have the appropriate A3D support.
  - **EAX:** It is used to produce echo sound waves which bounce off walls in a room. It helps in identifying the environment. EAX 1.0 was designed to provide developers with the ability to create a realistic sense of distance between the player and audio events. This simply creates predefined reverb effects for a variety of environments with different characteristics of reflections and reverberation, and different room types and room size. EAX 3.0 introduced the ability to morph between environments that allows developers to position and control clusters of early reflections as well as one shot reflections for ricochet effects and make full use of technologies, such as Head Related Transfer Functions (HRTF) for synthesizing positional audio on a single pair of speakers.
  - **MPEG Audio:** This hardware interface refers to digital audio encoding format using a form of lossy data compression. It is a common audio

format standard for consumer audio storage and digital audio compression for transferring the playback of music on digital audio players. The MPEG audio standard provides psychoacoustic models. As a result, there are many different MP3 encoders available, each producing files of differing quality. During encoding, 576 time domain samples are taken and are transformed to get 576 frequency domain samples. If there is a transient then the 192 samples are taken instead of 576 frequency domain. This is done to limit the temporal spread of quantization noise accompanying the transient.

5.8.3 Basic Tools in Multimedia

Text, Graphics, image, video, audio are considered as prime components of multimedia. They are discussed below:

Text

Text is one of the fundamental building blocks of multimedia applications. Other elements, such as graphics, sound, animation and video should be integrated with text using multimedia technology to render true potential to a multimedia presentation.

Apart from selecting the font the size and color of text to establish the basic look of the text, you can use different character formatting features to emphasize text. Some of the most widely available character formats are – **Boldface**, *Italic*, Underline, Super^{script}, Sub$_{script}$, Shadow, Emboss, Engrave, Strikethrough, Color, etc. You can make text more dramatic and interesting by adding special effects and styles, such as animation, extrusion, textured fills, shades, anti aliasing, text on a curve, 3-D text, etc.

When text lives in a computer, instead of on printed pages, the computer's powerful processing capabilities can be applied to make the text more lively, i.e., interactive. Some times when someone excites the text with a mouse it may quickly transform to an image or a music may be heard or more interesting informations may come up. This is what hypertext is.

The meaning of the word 'Hyper' is something close to 'extra' or 'beyond'. That is, there is something more behind the text you see. It may be connected to another related text material which again may have links to some new set of words or sections or even whole new documents and so on. Hypertexts are marked in some visible way (colored or underlined) to differentiate from ordinary text. On positioning the cursor over hypertexts, it changes to a pointing finger. If you click the text you can jump directly to additional available information through a predefined cross reference or link. Anticipating possibilities of reference reading or audiovisuals, several such links are embedded beforehand in a hyperdocument. A particular piece of information in a hypertext system can thus be approached from a variety of reference points or nodes. For example, you must have experienced navigating through the **Help** module of Microsoft Word or Windows or any other good package. All the terminologies in a page that need further explanation or illustration are hyperlinked to relevant help pages or sections.

In order to accommodate the text to the learner, rather than the learner to the text or its implicit structure, the text structure should be under the control of the learner. And that is what hypertext provides the users with. It allows learners to browse through the text material in a way that suits them – there is no predetermined order as such in which the text is to be read. One cannot do this kind of nonlinear and associative navigation in a sequentially organized book.

When words or texts are linked with other words or text you have a hypertext system. But when text, graphic and audio elements are cross-linked to each other enabling the user to access one element from the other then what you have is interactive multimedia or *hypermedia*. So like hypertext document hypermedia is a structured multimedia document where one navigates seamlessly among text, image, audio, video and animation. Suppose you click a hypertext 'Tiger' – a thumbnail image of a tiger comes up along with more information about tiger. Further click the small image it may enlarge or it may animate showing the movement of the tiger or it may roar. Such sensitive images, the hypertexts are hyperlinked to some other multimedia elements, known as hyperpicture. You will understand which are hot images because the cursor when positioned on such images will change to a pointing finger cursor.

Hypertext Markup Language (HTML) is a document layout and hyperlink-specification language that is used to create hypertext documents and Web pages. Almost all documents viewed on the World Wide Web (WWW) are HTML documents. It tells how to display the contents of the document including text, images and other supported media.

Graphics

Images can be generated and stored in the PC in typically two different ways. One is called vector art and the other is referred to as bitmapped.

A piece of vector art is a file that contains descriptions of how to generate the image but not the actual image itself. A vector art program generically called drawing program creates a sequential list of graphic commands to draw lines, curves, text, etc., with associated parameters, such as screen location, size, color, rotation angle, width, style, etc. This type of list file is often referred to as a display list / *file*. Such a file must be rasterized before it can be presented as an actual image on screen.

When you edit a vector graphic, you modify the properties of the lines and curves that describe its shape. You can move, resize, reshape and change the color of a vector graphic without changing the quality of its appearance. Vector graphics are resolution independent, meaning they can be displayed on output devices of varying resolutions without losing any quality.

Mostly, the Computer Aided Design (CAD) packages, business packages for drawing charts and graphs and some Desktop Publishing (DTP) packages, for example, CorelDRAW use vector files of specific formats. Some of the vector file formats commonly used in the Information Technology (IT) industries are as follows:

- Postscript file
- Computer Graphics Metafile (*.*CGM*)
- Windows Metafile (*.*WMF*)

- HPGL or, Hewlett Packard Graphics Language (*.*PLO*)
- Data Exchange Format (*.*DXF*)

Postscript files, developed by Adobe are generated by DTP packages and authoring systems while the Windows Meta File (WM File) was developed by Microsoft and is an excellent format for image interchange between Windows applications. Hewlett Packard Graphics Language (HPGL) is an interpreted vector description language meant for the plotters and Drawing Exchange Format (DXF) that is the most widely accepted format for interchange of engineering graphics data between different CAD packages, for example, AutoCAD.

Another kind of vector image specifies the content in full three-dimensional form. Here the objects are not what gets drawn in the image. Instead, a view of those objects is drawn. This substantially more complex process is called 3D-rendering.

A bitmapped image, in contrast, has in the file the actual pixel image data. That is, it simply holds the color number for each dot or pixel in an image.

When you edit a bitmap graphic, you modify pixels rather than lines and curves. Bitmap graphics are resolution dependent because the data describing the image is fixed to a grid of a particular size. Editing a bitmap graphic can change the quality of its appearance. In particular, re-sizing a bitmap graphic can make the edges of the image ragged as pixels are redistributed within the grid. Displaying a bitmap graphic on an output device that has a lower resolution than the image itself also degrades the quality of its appearance.

(a) ***(b)***

Fig. 5.40 *Vector and Bitmapped Image of a Leaf*

In Figure 5.40(a), the vector image of a leaf is described by points through which lines and curves pass, creating the shape of the leaf's outline. The color of the leaf is determined by the color of the outline and the area enclosed by the outline.

In Figure 5.39(b), the bitmapped image of a leaf is described by the specific location and color value of each pixel, creating a more realistic image.

Video

Just like text, audio and still image, digital video is also a powerful element of multimedia systems. To understand how digital video is used as a media we should study some fundamental aspects of analog video technology.

Basically video or motion pictures are created by displaying images depicting progressive stages of motion at a rate fast enough so that the projection of individual

images overlap on the eye. Persistence of vision of human eye which allows any projected image to persist for 40-50 ms requires a frame rate of 25-30 frames per second to ensure perception of smooth motion picture.

Following terminologies are used in a video display:

- Horizontal resolution is the number of distinct vertical lines that can be produced in a frame.
- Vertical resolution is the number of horizontal scan lines in a frame.
- Aspect ratio is the width to height ratio of a frame.
- Interlace *ratio* is the ratio of the frame rate to the field rate.

Constitution wise the three types of video signals are component video, composite video and S-video. Most computer systems and high end video systems use component video whereby three signals R, G and B are transmitted through three separate wires corresponding to red, green and blue image planes respectively.

However, because of the complexities of transmitting the three signals of component video in exact synchronism and relationship these signals are encoded using a frequency interleaving scheme into a composite format that can be transmitted through a single cable. Such format known as composite video used by most video systems and broadcast TV, uses one luminance and two chrominance signals. Luminance (Y) is a monochrome video signal that controls only the brightness of an image. Chrominance is actually two signals (I and Q or U and V), called color differences (B–Y, R–Y) and contains color information of an image. Each chrominance component is allocated half as much band width as the luminance, a form of analog data compression which is justified by the fact that human eyes are less sensitive to variations in color than to variations in brightness. Theoretically there are infinite possible combinations (additive) of R, G, B signals to produce Y, I, Q or Y, U, V signals. The common Committee Consultatif International Radiotelecommunique (CCIR) 601 standard defines the following:

$$\text{Luminance (Y)} = 0.299R + 0.587\,G + 0.114B$$

$$\text{Chrominance (U)} = 0.596R - 0.247\,G - 0.322B$$

$$\text{Chrominance (V)} = 0.211R - 0.523G + 0.312B$$

The inverse of the above transformation formula gives the following:

$$\text{Red (R)} = 1.0\,Y + 0.956\,U + 0.621\,V$$

$$\text{Green (G)} = 1.0\,Y - 0.272\,U - 0.647\,V$$

$$\text{Blue (B)} = 1.0\,Y - 1.061\,U - 1.703\,V$$

Unlike composite video, S-video separated video or super video as S – VHS uses two wires, one for luminance and another for a composite chrominance signal. Component video gives the best output since there is no cross talk or interference between the different channels unlike composite video or S-video.

In a television transmission system, every part of every moving image is converted into analog electronic signals and is transmitted. The VCR can store TV signal on magnetic tapes, which can be played to reproduce stored images. There are three main standards for analog video signals used in television transmission:

National Television Standards Committee (NTSC), SÉquential Coleur a Mémoire (SECAM) and Phase Alternating Line (PAL).

Digital Video: For video to be played and processed in computers it needs to be converted from analog to digital representation. Video digitization is achieved just like audio digitization by sampling the analog video signal at preset frequency and subsequently quantizing the discrete samples in digital format. This is done with the help of analog to digital converter or ADC.

There are two kinds of possible digitization or digital coding - *composite coding* and *component coding*. In composite coding all signal components taken together as a whole are converted into digital format. In component coding each signal component is digitized separately using different sampling frequency (13.5MHz for luminance, 6.75 MHz for chrominance).

Based on whether the ADC is inside a digital camera, or in an external unit or inside the computer there can be three types of digital video systems – *Digital camera based system*, *external ADC system* and *video frame grabber card based system*.

Digital Video Standards

To improve the picture quality and transmission efficiency new generation television systems are designed based on international standards that exploit the advantage of digital signal processing. These standards include *High Definition Television* or *HDTV*, *Improved Definition Television* or *IDTV*, *Double Multiplexed Analog Components* or *D2-MAC*, *Advanced Compatible Television*, *First System* or *ACTV-I*. HDTV standard that supports progressive (non-interlaced) video scanning, has much wider aspect ratio (16:9 instead of 4:3), greater field of view, higher horizontal and vertical resolution (9600 and 675 respectively in USA) and more bandwidth (9 MHz in USA) as compared to conventional color TV systems.

Video Compression

Out of different multimedia elements the need for compression is greatest for video as the data volume for Full Screen Full Motion (FSFM) video is very high. Frame size for NTSC video is 640 pixels by 480 pixels and if we use 24 bits color depth then each frame occupies $640 \times 480 \times 3$ bytes, i.e., 900 KB. So each second of NTSC video comprising 30 frames occupies 900×30 KB which is around 26 MB and each minute occupies $26 \times 60 \approx 1.9$ GB. Thus a 600 MB CD would contain maximum 22 seconds of FSFM video. Now imagine the storage space required for a 2-hour movie. So the only way to achieve digital motion video on PC is to reduce or compress the redundant data in video files.

Redundancy in digital video occurs when the same information is transmitted more than once. Primarily in any area of an image frame where same color or intensity spans more than one pixel location, there is *spatial redundancy*.

Secondly when a scene is stationary or only slightly moving, there is redundancy between frames of motion sequence – the contents of consecutive frames in time are similar, or they may be related by a simple translation function. This kind of redundancy is called *temporal redundancy*.

Audio

What we call sound wave is a repeating pattern of changes of pressure in the air. The air particles transmitting such a wave oscillate in the direction of propagation of the wave itself. The waves reach our inner ears and affect the eardrums pushing them in and out at same frequency thereby creating impulses in the nerves. These impulses are then carried to the brain which in turn registers the sound.

The quality of sound we can perceive depends largely on two important parameters – the *frequency* and the *amplitude* of sound wave. The number of complete vibrations in a second performed by a particle in the path of the wave is called the *frequency* of sound measured by the unit Hertz (Hz). Sounds of higher frequencies are sharper and shriller than those which are of lower frequencies and hence termed as *high pitched* sound. On the other hand *amplitude* of a wave means its wave height – the maximum displacement suffered by a particle from its mean (or equilibrium) position and it is proportional to the amount of energy carried by the wave. The greater the amplitude of a sound wave the louder it sounds to the ear.

Though sound waves are invisible, we can trap and view the instantaneous shape of the wave in an oscilloscope. The simplest sinusoidal shape of sound wave is generated when we whistle. But waves that musical instruments generate are more complex typically jagged and irregular. However each instrument generates a characteristic waveform and that is why a musical note played on a flute sounds different from the same note played on a violin.

Using a *microphone,* sound wave (or acoustic energy) can be converted into an electrical wave (or electrical energy) that will faithfully follow the original sound wave with identical frequency and amplitude. The electrical counterpart or the *analog sound* signal (in terms of few milli-volts) can then be boosted to enhance the audibility or loudness electrically using an *amplifier*. Again analog sound can be translated to digital form, i.e., in terms of 0's and 1's to be able to store and process sound in digital PC chips or to transmit them over digital communication lines. The process of digitizing sound is done through an Analog to Digital Converter (ADC) and after the sound is processed it is converted back to analog form using a Digital to Analog Converter (DAC) to put into the stereo system (refer Figure 5.41).

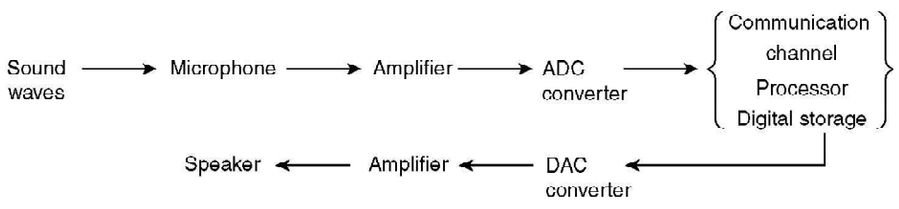


Fig. 5.41 *Flow of Digitizing Sound*

ADC follows the analog input signal takes a snapshot of its value at particular pre-determined instances called sample time and holds onto the value until

conversion is complete and then outputs a binary number. The DAC accepts a number and holds it until the value is converted to electrical signal. Following technologies are used in digitizing sound:

Sampling Rate

Sampling means that the amplitude of the audio analog signal is measured and stored at some point in time. The measured signals are still analog signals which are then converted to digital signals. Unlike the simplest possible analog signal sited in the example, most musical analog signals are not of repeating pattern and in order to follow them faithfully sampling must be done many times per unit time. The higher the time rate of sampling the more likely it is to trap the delicate changes in the original sound wave amplitude.

The optimum sampling rate is governed by the *Nyquists sampling theory*. It states that if a signal contains frequency component upto some frequency n then sampling frequency must be at least $2n$ in order to reconstruct the signal properly. Today's CD system uses a sampling rate of 44.1 kHz (44,100 times per second) which allows for a maximum frequency content (also called *Nyquist frequency*) of 20 kHz in the wave that has been sampled. Human speech can be effectively reproduced at a rate of 5.5 kHz which requires a sampling rate of atleast 11 kHz.

16-bit Sound

The resolution, with which the sampled amplitudes are stored, known as *sampling resolution*, is another important factor besides sampling rate that determines the quality of digitized sound. Resolution refers to the number of different amplitude levels used to represent the analog signal.

If an ADC of 8-bit resolution is used to digitize audio signals, the output value will range from –128 to +127, i.e., each output sample will be stored as one of the 256 (2^8) possible levels. However this is not an ideal quantization for good quality sound. On the other hand if we use a 16-bit ADC to digitize sound then it can produce at the best 65,536 (2^{16}) levels of sound amplitude between the minimum and maximum. Still the resulting digitized waveform of stepped nature will be an approximation to the original waveform (refer Figure 5.42). The error due to approximation is called quantization error.

If x bits are used to represent each sample, i.e., the sample resolution is x-bit, the number of quantization levels Q is given by,

$$Q = 2^x$$

If A is the maximum amplitude in the analog signal, the quantization error E_Q is given by,

$$E_Q = A/Q = A/\ 2^x$$

Modern digital audio systems use 16-bit number called word to produce reasonably high quality sound. However for each sample that we take of the original wave, we have to record 16 digital signals [0 or 1] and all 16 will be needed to reconstitute the original wave. Each 16-bit binary number output (samples) from the ADC is stored in sound files on any digital storage media thus forming a sequence of binary numbers.

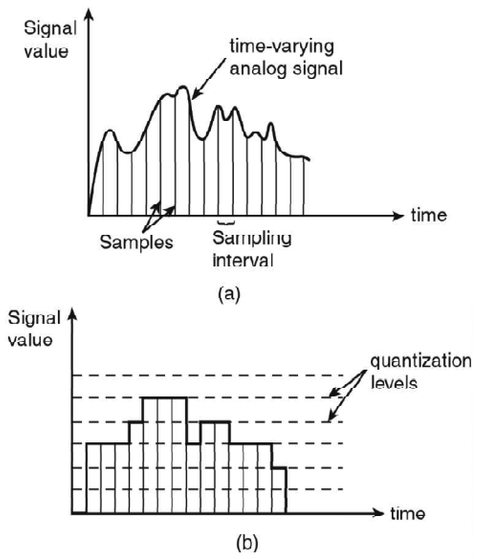


Fig. 5.42 *Signal and Time Graph*

5.8.4 Multimedia Building Blocks(Media/Forms/ Elements)

Authoring Tools

The key to successful multimedia production is a seamless integration of multimedia elements for graphic design, content management, production and packaging. The whole process of developing a multimedia package is called authoring. An **authoring system** is a collection of software tools that help in various aspects of multimedia production. Multimedia elements are woven together using authoring tools. These tools are designed to manage individual multimedia elements and provide user interaction. Multimedia authoring software can be used to make:

- Video productions.
- Animations.
- Games.
- Presentations.
- Interactive kiosk applications.
- Interactive training.
- Simulations, prototypes and technical visualisations.

Steps in Multimedia Production

The following steps can be followed in multimedia production:

1. ***Media Capture:*** Multimedia authoring systems streamline data capture by providing interfaces to a range of image and data capture devices.
2. ***Media Conversion:*** Images, audio clips, animation, sequences and video clips exist in a variety of formats. A well-equipped multimedia authoring system will include a set of utilities for converting between many of the commonly used formats.

3. ***Media Editing:*** After data has been captured and converted to the native format of the authoring system, it may need some polishing before it is suitable for presentation. For instance, 'noise' can be removed from audio clips, images can be touched up, etc. Multimedia authoring systems provide media – specific editors for these operations.
4. ***Media Composition:*** The core of a multimedia authoring system includes a tool for combining media and specifying their spatial (one image being juxtaposed or placed side-by-side within a second) and temporal (when an audio track is added to a visual sequence) relationships.

Elements of Multimedia

The following are the various elements of multimedia:

1. Text

Text, containing words and symbols, is the most common form of communication. It is one of the popularly used mediums of appearance that is used to deliver information accurately and in detail. Usually text provides the core structure to the package. Words are vital elements of multimedia that can appear in the titles, menus, navigation aids and in the content of a multimedia application or project. It is most essential to use words that have the most precise and powerful meanings to express what you need to convey.

A major drawback of using text is that it is not user friendly as compared to the other elements of multimedia. For example: it is harder to read from a screen for long, as it tires the eyes more than reading it in its print version.

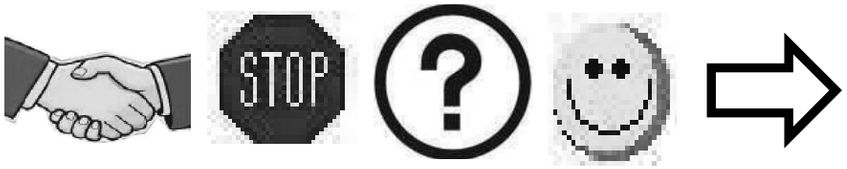
2. Designing with Text

From a design perspective, the choice of font size, style and other text attributes needs to be related both to the complexity of the message and to its venue.

Some useful tips for designing the text in your multimedia application:

- Use legible fonts that can be easily read.
- Vary the font size and style according to the importance of your message.

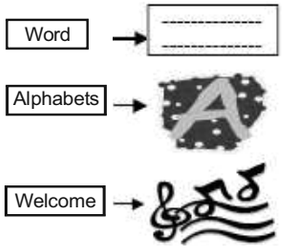
- Indent your paragraphs wherever required.
- Explore the effects of different colors and shadows to add depth to your application.
- Use menus for easy navigation and meaningful words for menu items.
- Use buttons, icons or symbols for user interaction.



- You can also use stylish fonts for displaying attention-grabbing results.

3. Hyperlinks

Hypertext is the organisation of information units into connected associations that a user can choose to make. An instance of such an association is called a **Hyperlink.** When a user clicks on such a link, more information on the particular topic is displayed. It, therefore, provides the user an option of reading as much information as required. Hyperlinks can contain cross-linking of words not only to words but also to images, videos or sound files. Hyperlinks are used for non-linear navigation, which is not an option available in a sequentially organized book.



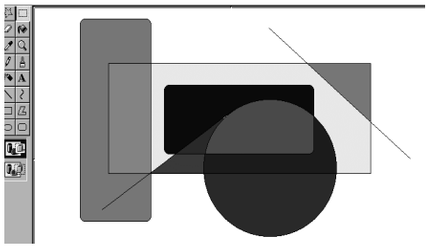
4. Graphics

Pictures/graphics enhance the overall look of a multimedia package. Pictures express more than normal text and are generally considered as the most important element of a multimedia application.

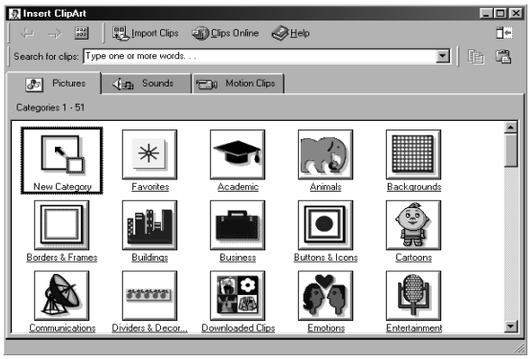
It is often noticed that a Webpage containing numerous images takes longer to download than a simple text based webpage. Image files are, therefore, compressed to save memory and disk space of your computer. **GIF** (Graphics Interchange Format), **JPEG** (Joint Photographic Experts Group) and **PNG** (Portable Network Graphics) are examples of compressed image file formats.

Pictures can be created using any of the following ways:

- You can use drawing tools like MS paint to create simple pictures. Paint allows to create or assemble pictures by drawing straight, wavy or curved lines, using shapes like squares, circles and polygons or simply by freehand drawing.



- You can insert images from the ClipArt gallery. A ClipArt collection typically contains a series of images for different categories. ClipArt is available through CD-ROMs or from the Internet.



- You can use scanner (Figure 5.43(b))or digital camera (Figure 5.43(a)) to capture original pictures in digitised form. You can also scan images, created using traditional methods like watercolours, crayons etc.

(a) Digital Camera **(b) Scanner**

Fig. 5.43 *Capturing of Original Pictures in Digital Form*

- If you are still not satisfied with scanned pictures or images downloaded from the ClipArt gallery, you can use image editing tools to manipulate images according to your taste. Picture properties that can be manipulated using these tools include brightness, contrast, color, depth, hue and size. Apart from correcting the blemishes in your image, you can add additional effects like filters, shadows, patterns, 3D-effects and many more. Image editing tools are indispensable for excellent multimedia production. Adobe Photoshop, Microsoft PhotoDraw are some examples of image editing tools.

Some useful tips for adding images in your multimedia application:

- Do not overload your application with too many images. This would not only make your application look clumsy, but it would also consume excessive computer resources.
- Do not include heavy (oversized) graphics.
- Use context – sensitive images.
- To the extent possible, use compressed image file formats. Avoid using unoptimised graphics.
- Pick the right color or combination of colors for your multimedia application. Color scheme used for the text should blend well with the images.

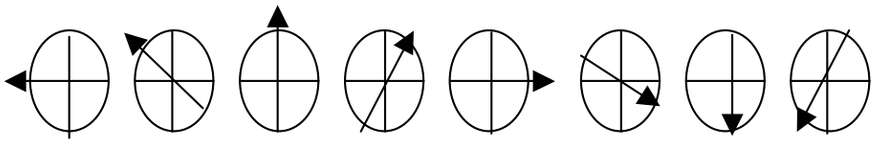
Animation

Animation gives visual impact to your multimedia application. In simple terms, it can be defined as an entity moving across the screen. This entity could be a text object or an image. An animation consists of a series of rapidly changing objects,

NOTES

which when blended together gives an illusion of movement. The speed with which each object is replaced by the next one is so rapid that the eye perceives this as motion.

Example: Consider the process of rotating a wheel, the position of the arrow changes so rapidly that it gives an illusion of spinning.



Animation Tools: MS PowerPoint is a tool used for creating primitive animations. Visual effects like wipes, dissolves, fades and zooms can be added to any object. For example: you can make a text to fly from top or left. Such effects are available with almost all authoring packages. You can create complex animations using tools like Director, 3D Studio Max, CompuServe and Shockwave. Such animations can be ported across platforms and applications by making use of suitable translators.

Some useful tips for adding animations in your multimedia application:

- Before you create an animation, organise its execution into a series of logical steps. First choose the objects in your presentation that you want to animate and then decide the sequence of animation. In case of complicated presentations, writing a detailed script of the list of activities will prove useful.
- You can animate one or all objects of an application. As mentioned earlier, applications intended for the Web should not contain too many animations as it would affect the download time.
- Add user interactivity wherever essential to the application.
- You can combine animations with live sounds for catching the user's attention.

Sound

Sound is used to set the rhythm or a mood in a package. Speech gives an effect of a language (pronunciation) for instance. Proper usage of sound can make all the difference between an ordinary multimedia presentation and a professional one.

Sound Types: Musical Instrument Digital Interface (**MIDI**) is a communication standard developed in early 1980's for electronic musical instruments and computers. A MIDI file consists of a list of commands that represent the recordings of musical actions. When these commands are sent to a MIDI Playback device, a sound is produced. The main disadvantage of MIDI data is that it is not digitised. In contrast, **Digital Audio** is a recording that depends on the capabilities of your sound system. Digital audio data are the actual representations of sound data and are stored in the form of thousands of individual numbers known as **Samples**. Digital sound is used for music CDs.

Creating Sound: Sound can be recorded using a microphone, a synthesiser, or any other medium like tape or cassette player and then be digitised using audio digitising software. Therefore, sound may be digitised from any source – natural

or pre-recorded. Digitised sounds are stored as wave (.WAV) files (Windows Platform). These can then be played using Windows Media player.

Some useful tips for adding sounds to your multimedia application:

- A distorted recording sounds terrible, so before a sound file is added it must be tested for clarity. If required, it must be edited using Audio-Editing tools like Wave Studio Sound or Macromedia's Sound Edit 16-2. It may be worth noting that higher the sound quality, the larger would be the file size.
- Decide the kind of sound you need, such as background music, special sound effects or spoken dialogue.
- Test the sound, to ensure they are synchronised properly with images and or animations.

Video

If pictures can paint a thousand words, then motion pictures can paint a million. Digital video is the most engaging of multimedia venues and is a powerful tool for bringing users close to the real world.

NTSE (National Television Standards Committee), PAL (Phase Alteration Line), SECAM (Sequential Color and Memory), HDTV (High Definition Television) are commonly used broadcast and video standards across the globe.

Current television is based on analog technology and fixed international standards for the broadcast and display of images. Computer video, on the other hand is based on digital technology and other standards for displaying images. Digital Video is produced using analog video as a base. The conversion of analog video into its digital equivalent requires a special hardware called **Video Capture Card.**

Video data is also compressed using different compression techniques. MPEG (Motion Pictures Expert Group), JPEG (Joint Photographic Experts Group), P*64, real video are examples of commonly used compressed video formats.

Some useful tips for adding video clips to your multimedia application:

- Video clippings which are not appropriately designed can degrade your presentation rather than add value to it. Carefully planned, well-executed clips can make a dramatic difference in a multimedia presentation.
- Titles used in video clips should be plain enough to be easily read.
- Avoid making busy title screens; use more screen if required.
- Again, any multimedia element that is added to an application intended for Web should be compressed to support quick and easy download.

Cross Platform Compatibility

In computer technology, cross platform or multi-platform refers to the unique characteristic of computer software which enables it in implementing methodologies for inter-operating on several computer platforms. Typically, the cross platform software can be classified into following two types:

- First that supports specified creation or compilation for each platform that it holds.
- Second that can be directly run on any platform without any specification, such as software written in an interpreted language and precompiled portable bytecode which are universal and standard components supported by all platforms for the interpreters or run-time packages.

The cross platform compatibility specifies the software or hardware capability that can run indistinguishable on different platforms. For example, nowadays numerous applications for the Windows and the Macintosh have the ability to create binary compatible files which indicate that users can switch from one platform to the other without changing the data into a new format because the data will be supported due to cross compatibility feature. Hence, the cross platform computing is gaining significant importance since local area networks use advanced technology and are compatible to link computer systems of different types.

In multimedia, the cross platform is the unique capability of an application that helps in accurately performing on a range of computers, operating systems and Web browsers. Basically, it includes the following:

- Hardware Issues.
- Operating Systems.
- Environment.
- Software.
- Elements.
- Design.
- Publication.

Hardware Issues: It includes the following:

- **Device Type:** PC/Laptop (including Tablet PCs)/Mobile Devices (Phones, PDAs).
- **Device Performance:** Based on the performance of Processor/Hard Drive/Optical Drive.
- **Available Peripherals:** Mouse, Keyboard, Stylus (PDAs), Screen (Size, Touch Screen), Speakers, Microphone (for Voice Input), Hand Controllers, Joysticks.
- **Media:** CDs/DVDs, Internet Connection (Broadband/Dial-Up)
- **Monitor/Screen Resolution for a PC:** 2005 - 800 × 600 Pixels, 2010 – 1024 ´ 768 Pixels, 2o11 - Superior Web Solutions.

Nowadays maximum numbers of people use their mobile devices to surf the Internet which are equipped with advanced technologies and software.

Operating Systems: It includes the Windows/Macintosh/UNIX. Some of the 'old' multimedia applications are specifically designed for different screen resolutions and quality of color. Hence, must be run in compatibility mode or on a virtual machine.

Environment: It includes the following:

- Internet versus CD-ROM/DVD

  - **Graphics:** File Sizes, Types, such as gif, jpg, png on Web.
  - **Video:** File Sizes, Codec Availability.
  - **Updates:** For CD-ROM - Fixed, For Internet - Easily Changed.
  - **Errors:** For CD: To Remaster, For Internet - Easily Changed.
- To build an iPhone application requires an Apple Computer, Intel only.
- Web Issues: Corporate firewall policies may restrict access to certain sites, such as game sites and social networking sites which are often blocked.

Software: It includes the following:

- **Browser Issues:** Browser Specific Tags (I.E.- Marquee), HTML-5 Multimedia Tags and JavaScript capabilities, such as Google Chrome Experiments.
- **Plug-Ins:** Adobe Flash Player Version, Microsoft Silverlight
- **Application Specific:** Flash capable to create standalone exe/swf files or html/swf files.
- **Supported File Types:** The following file types are supported in multimedia on the Internet:
  - **Images:** On Web - gif, jpg, png and animated png.
  - **Videos:** Microsoft WMV Format for Windows, DivX (usually wav), Apple Quicktime (mov) and MPEG.

Elements: It includes the following for the Web:

- **Image Size:** Use thumbnails for images and also uses file compression (jpg), and resize the image.
- **Music:** Background sound may clash with foreground sounds or user preferred music.
- **Music Formats:** It includes MIDI, WMA (not Linux), MP3 (Linux).
- **Font Choices:** Different fonts for PC/Mac/Linux, UNIX, for example, Arial, Helvetica, Times New Roman, Calibri, etc.

Design: It includes the following:

- Different conventions for different systems.
- Different interfaces, for example Google Android and Microsoft Windows Phone 7.
- Colors can be handled differently in different systems.
- Web page colors may vary in different Web browsers on different platforms. The 216 of 250 possible index colors are safe for use.
- Different available default fonts. Text can be converted to a graphic or CSS font families that are currently used.
- Guides to develop an application to the lowest common specification.
- Different languages. Text is easier to convert though may require a language pack to support special characters, such as Chinese, Japanese, Russian, etc.

Publication

- CD-ROM/DVD requires burning to CD/DVD, producing labels and distributing which includes marketing.
- Internet requires uploading to Web server usually using FTP.
- Internet based applications require a technology infrastructure, as follows:
  - o Choice of Web Host.
  - o Cost of Uploading/Downloading Files and Capacity.

HTML Multimedia

Typically, multimedia on the Web is pictures, sound, music, films, videos and animations. Nowadays, the Web browsers support various multimedia formats because the recent Web pages have embedded multimedia elements.

Browser Support: The first Internet browsers only supported text which was limited to a single font in a single color. Later the new browsers were developed which had support for different colors, fonts and text styles and also support for pictures. Various browsers support sounds, animations and videos in different ways. Some directly support multimedia elements whereas some need an extra helper program, such as a plug-in.

Multimedia Formats: Multimedia elements, such as sounds or videos are stored in media files. The most universal way to determine the type of a file is to check the file extension. When a browser observes the file extension as .htm or .html, it considers the file as an HTML file. The .xml extension specifies that it is an XML file and the .css extension specifies that it is a style sheet file. Pictures are distinguished by extensions .gif, .png and .jpg. Multimedia files have their specific formats with extensions .swf, .wav, .mp3 and .mp4.

Commercial Tools

Multimedia presentation is of two types, live or recorded. In recorded multimedia presentation, the interaction is possible through a navigation system while in a live multimedia presentation the interaction is done with the help of a presenter or performer. The example of a live multimedia performance is a laser show. There are various commercial tools of multimedia technological or digital multimedia technology that enhance the users' experience to convey information, for example technology involving illusions of taste and smell. The following are some examples of commercial tools that can be used to develop a multimedia presentation:

Drawing, Painting and Graphic Tools

(i) Adobe Photoshop: The Photo Editing Tool

Adobe Photoshop is a graphics editing program developed and published by Adobe Systems. The new advanced version of Adobe 2003 'Creative Suite' is reframed, rebranded and named as Adobe Photoshop CS. Adobe Photoshop CS6 is the 13th major release of Adobe Photoshop. Adobe Photoshop is released in two specific editions: Adobe Photoshop and Adobe Photoshop Extended with the Extended extra 3D image creation, motion graphics editing and advanced image analysis features. In 2011, a version of Adobe Photoshop was specifically

released for the Android operating system and the iOS operating system.

Adobe Photoshop is a standardized tool used to develop and edit raster (bitmap) graphics. Using it, the user can draw, paint, process or retouch the photographs, develop designer solutions, create Web graphics, design program interfaces and it also helps in Web page development.

(ii) CorelDRAW

CorelDRAW is a vector graphics editor developed and marketed by Corel Corporation of Ottawa, Canada. It is the alternate name of Corel's Graphics Suite, which bundles CorelDraw with a bitmap image editor, Corel PhotoPaint and other graphics related programs. CorelDRAW was originally developed for Microsoft Windows 3 and currently runs on Windows XP, Windows Vista and Windows 7. CorelDRAW is capable of handling multiple pages along with multiple master layers. It is very useful in creating and editing multi-article newsletters, documents, etc. Besides, the other items, such as business cards, invitations, etc., can be designed to their final page size and imposed to the printer's sheet size for cost-effective printing. An additional print and merge feature permits full personalization applications, such as numbered raffle tickets, individual invitations, membership cards, etc.

(iii) Corel PhotoPaint

Corel PhotoPaint is a component of the CorelDRAW Graphics Suite and is used to exchange data with other programs in the suite, including Corel CONNECT (Version X5) which enables users to share files between different computer software and the different drives on the user's computer. CorelDRAW and Corel PhotoPaint are copy-paste compatible.

A Corel PhotoPaint is a specific program used for the development and processing of raster (bitmap) graphics. Basically, it comes in a CorelDraw package intended for creation and processing of graphic elements and it can be used as an alternative to Photoshop. The native format of Corel PhotoPaint is .cpt which stores image data as well as information within an image including objects (layers in some raster editors), color profiles, text, transparency and effect filters. The program can open and convert vector formats from CorelDRAW and Adobe Illustrator and can also open other formats including .png, .jpg and .gif files. Corel PhotoPaint is available in English, German, French, Italian, Dutch, Spanish, Brazilian Portuguese, Swedish, Finnish, Polish, Czech, Russian, Hungarian and Turkish.

(iv) Macromedia Fireworks

Macromedia Fireworks is a unique program specifically developed for processing and development of raster graphics especially intended for the Internet pages. Generally, it comes in a package with programs, such as Flash and Dreamweaver, and exceptionally harmonizes their functionalities. It is closer to Adobe ImageReady.

(v) Interactive Media: Adobe Flash

Adobe Flash was formerly termed as Macromedia Flash. It is a unique multimedia platform specifically used to add animation, video and interactivity to Web pages. Flash is frequently used for advertisements, games and flash animations for broadcast

purposes. Recently, it is considered as a tool for Rich Internet Applications or RIAs. Flash manipulates vector and raster graphics to provide animation of text, drawings and still images. It supports bidirectional streaming of audio and video, and it can also capture user input via mouse, keyboard, microphone and camera. Flash contains an object-oriented language called ActionScript and supports automation through the JavaScript Flash Language (JSFL).

Flash Player is also available to handset manufacturers for smart phones. The Adobe Flash Professional multimedia authoring program is used to create content for the Adobe Engagement Platform, such as Web applications, games and movies, and content for mobile phones and other embedded devices.

Multimedia Standards

The term 'Standards' refers to the specifications made by the systematic efforts approved by official standardization federations committed to the issue and can sometimes be termed as *official standards*. In some specific cases it is termed as *de facto standards* when it is widely accepted by the industry and/or the public.

The multimedia standards include the following:

- CCITT/ISO (now ITU – T) standards for multimedia include F.700, G.711, G.721, G.722, G.725, H.221, H.242, H.261, H.320, HyTime, IIF, JBIG, JPEG, MHEG, MPEG, ODA, T.80, X.400, G.723, G.726, G.727, G.728, G.764, G.765, H.200, H.241, H.243, T.120.
- Internet standards include IP Multicast, MIME, RTP, ST-2, RFC 741, Xv and mvex.
- W3C standards.
- Proprietary standards are Bento, GIF, QuickTime, RIFF, DVI, MIDI.

There are semi-official standards between official and de facto, such as the Internet and W3C standards.

Image and Video Standards

Unprocessed image and video in digital form take up enormous amount of space as compared to text. Usually, a character is one byte and a page can be several hundreds. A color image of the size of a small standard VGA screen occupies 640 ´ 480 ´ 3 bytes, i.e., the number of pixels multiplied by the number of bytes per pixel, one for each of the R, G and B channels. It is extremely important to reduce this size for storage and transmission purposes which can be achieved by coding the image in a compressed way at one end and then decoding it to an uncompressed form at the other end. Thus, the image and video compression standards must be adopted for transmission.

(i) JPEG

JPEG stands for Joint Photographic Experts Group, the original name of the committee that specified the standard. Now the body is officially called ITU – T JTC1/SC2/WG10. JPEG is a *lossy* compression format which means that the decompressed image is usually worse than the original. The compression parameters can be adjusted by the user, trading off file size against output image quality. Recently, a new version called **JPEG2000** has become an official standard. It is based on

wavelet transforms and the applications are more ambitious than for the original JPEG.

(ii) MPEG

MPEG stands for Moving Pictures Expert Group and is the popular name of the ISO/IEC committee working on digital color video and audio compression, ITU – T JTC1/SC2/WG11, established in 1988. According to MPEG home page, the group is 'in charge of the development of standards for coded representation of digital audio and video'. MPEG has the following versions:

- MPEG-1, a standard for storage and retrieval of moving pictures and audio on storage media. The products, such as Video CD and MP3 are based on it.
- MPEG-2, a standard for digital television. Digital Television set top boxes and DVD are based on it.
- MPEG-4 version 1 and 2, a standard for multimedia applications for the fixed and mobile Web.
- MPEG-7 a content representation standard for multimedia information search, filtering, management and processing.

(iii) MHEG

MHEG stands for Multimedia and Hypermedia Information Coding Experts Group, officially called ITU – T JTC1/SC2/WG12. The objective of the standard was to develop a Coded Representation of Multimedia and Hypermedia Information. The standard specifies a coded representation of final form of multimedia/ hypermedia information objects to be interchanged as units within or across systems by any means of interchange, from storage devices to telecommunication and broadcast networks. These objects define the structure of multimedia/hypermedia presentation in a system independent way. These MHEG objects provide functionality for final form representation, support for systems with minimal resources, interactivity and multimedia synchronization, real-time presentation and interchange.

Check Your Progress

6. What is Binary Space Partitioning (BSP)?
7. List the properties of the Bezier curve.
8. How many control points are required to generate cubic Bezier curves?
9. How does multimedia helps in medicine?
10. What is the importance of text in multimedia?

5.9 DVI INDEO

The Digital Display Working Group created the Digital Video Interface (DVI) technology that accommodates for interconnecting the single connector to both digital and analog interfaces. The DVI technology supports high-speed connection (digital) to display the videos and animations. It facilitates a common interface for

all the display devices. The high-speed digital signals in DVI technology support up to 350 Mpixels/sec as well as SVGA. The plug and play service is supported via hot-plug detection in this interface. It is frequently being used in LCD displays. The nature of operation of this technology is implemented in the pure digital form. Therefore, intermediate analog signals are not needed to use the DVI technology. It encounters analog-to-digital conversion to remove the synchronization of the operations and aliasing problems. Conceptually, DVI implies a collection of video interface through which LCD flat monitors get good quality modern video-graphics cards. For this, DVI cables are being used including both VGA and DVI output port (see Figure 5.44).

Fig. 5.44 *Digital Video Interface*

The technology behind DVI uses Transmitting Minimized Differential Signaling (TMDS) to transmit the desired data over DVI connection. One TMDS is linked to transmit the data but sometimes dual links and two TMDS channels are also preferred to link if it is assembled to the system unit. The three data channels (RGB fiber optics) are maintained by a single link in which one channel (clock control channel) is used to access the demanded video even if the dual link is connected to the system unit (see Figure 5.45).

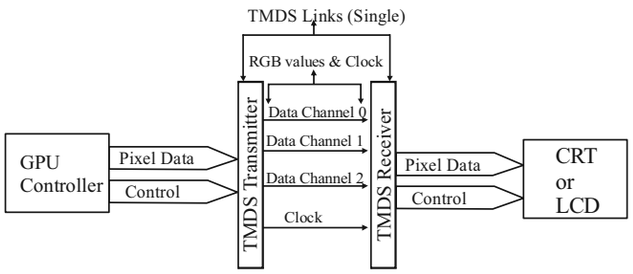


Fig. 5.45 *Data Channels Linked with DVI Connection*

In Figure 5.45, the Graphics Processing Unit (GPU) controller is used to send the pixel data that is passed to various data channels. The data channels are named data channel 0, data channel 1 and data channel 2 that are bound with time constraints. After controlling and checking, each pixel is used in graphics and the specific data are sent to the CRT or LCD monitors to be processed that are to be viewed by the customers/viewers. The TDMS link of 10-bit is generally operated till 165 MHz. It can also be linked up to 1.65 Gbps of bandwidth. A digital flat panel is displayed by 1920×1080 screen resolution and the electric power is generated at 60 Hz. Basically, this flat panel supports dual-link TMDS because dual link supports 2Gbps of bandwidth and is operated to match every second link to the previous one. The dual TMDS uses 2048 × 1536 screen resolution to achieve better graphics for the DVI technology.

Working of DVI

The system unit (PC/VGA/CRT) creates and transmits the video signal in the form of digital signals (0 and 1). The digital CRT monitor displays the analog signals but the video card of the system unit (VGA connection) converts the digital signals to analog ones to display the video data. The role of DVI technology is important at this step because the LCD monitor uses the graphics interpreter with the help of the DVI connection and changes analog signals back to digital ones.

In Figure 5.45, most of the video cards are used with DVI technology to convert the digital-to-analog signal and then convert analog-to-digital signal in the LCD display unit. The data in the PC is sent to the processing of electronic signals and then it appears on the LCD monitor.

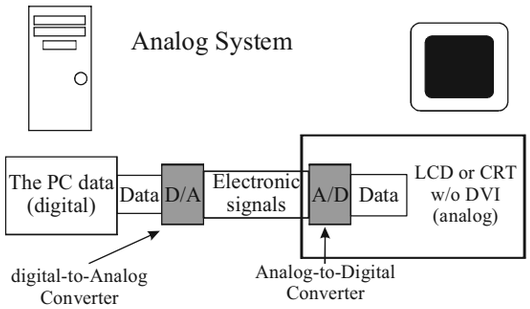


Fig. 5.45 *LCD Monitor with DVI Technology*

The image quality is not good and the display unit shows a lower resolution. The function of DVI is to remove the bad quality of graphics that appear in the video or animation.

Types of DVI

The DVI represents the mode of video interface technology, which is used to optimize Digital Flat Panel (DFP) standards. The types of DVI connections are as follows:

1. DVI-D (True Digital Video)

In the DVI-D system, the cables are directly connected to the source video, for example, video cards and digital LCD monitors. It offers a high quality image in comparison to the analog image. The analog signal is sent to the monitor and then changed to the digital format. It enhances the source connection and omits the process of analog conversion.

2. DVI-A (High-Resolution Analog)

In the DVI-A system, the cables are used to transmit the DVI signal to display in the analog format for the LCDs and CRT monitors.

3. DVI-I (High-Resolution Analog)

In this system, the cables are integrated and are used to transmit the digital source signal to a digital display or it can change the analog source to an analog display. It makes the DVI-I connection flexible and better than other DVI technologies.

Features of DVI Technology

The features of DVI technology are as follows:

1. It uses proprietary chips and the data compression method to create a form of multimedia that is to be integrated into the desktop system unit.
2. It is helpful to play back the full motion videos, multiple stereo sound tracks, live television shows, color graphics, etc.
3. It incorporates and stores DVI into the storage devices for desktop system unit and Winchester hard drives.
4. It plugs the interface boards to distribute the available expression slot on the motherboard and then installs the software.
5. It provides an enhanced technique manifesting the text-oriented e-mail and marks the sender of the message with the motion video communication.
6. It supervises the information tracking to transmit the videographic instructions for subordinating the messages sent, as per VoD technology.
7. It is used to send and receive the applications of videographic presentations, videographic voice mail, audio–visual databases, audio–visual references, sales messages, etc.

5.10 GRAPHIC FILE FORMATS

There are several graphic image file formats that are used by most of the graphics systems. The following are the most regularly used formats:

Web Document Images

Web document images can be of two types as follows:

(a) ***Graphics Interchange Format (GIF):*** Images made using this format use a fixed colour palette which is limited to only 256 colours. This format downloads small, compressed files quickly from the Web. This format is most suitable for images with solid colours or uniform colour areas such as illustrations and logos.

(b) ***Joint Photographic Experts Group (JPEG):*** These files are used for photographic (continuous colour tone) images, i.e., those images that have a continuous colour tone. Unlike the Graphics Interchange Format files the Joint Photographic Experts Group format takes advantage of the full spectrum of colours available to the display unit. The JPEG format also uses compression for making smaller files and for obtaining faster downloads over the World Wide Web. However, unlike the compression method used in GIF files, the JPEG compression is also 'lossy compression' which means it discards some data in the decompression process. Once a file is saved in the JPEG format some data is lost permanently. But this does not affect the image.

Printed Documents

The following are the two types of printed documents.

(a) ***Encapsulated PostScript (EPS):*** It is an image file format used for both vector graphics and bitmaps. EPS files have a PostScript description of the graphic data within them. The EPS files are exclusive in that the graphics users use them for bitmap images, vector graphics, type, or even entire pages.

(b) ***Tagged-Image File Format (TIFF):*** Such files are used for bitmap format only. The TIFF formats are the files that are supported by virtually all graphics applications.

Check Your Progress

11. Why are static bitmapped images compressed?
12. What is an interlace ratio?
13. Define the term authoring tool.
14. How will you define hyperlinks?
15. Give any one use of DVI technology.

5.11 ANSWERS TO 'CHECK YOUR PROGRESS'

1. If a set of lines is parallel to one of the three principal axes, the vanishing point is called an axis vanishing point.
2. Parallel projection the angle made by the direction of projecting lines with the projection plane or view plane.
3. A perspective transformation is determined by specifying a definite center of projection (COP) (also called viewpoint), and a plane of projection.
4. A-buffer method is an extension of the techniques which are used in depth-buffer method and represents an antialiased, area-averaged method used for the implementation of a surface-rendering system called REYES (which means Renders Everything You Ever Saw).
5. By using the image-space and object-space operations, the depth-sorting method performs the following basic functions:
   - Surfaces are sorted in decreasing order of depth.
   - Surfaces are scan converted in order such that starting from the surface of greatest depth lowest.
6. Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements.
7. The following are the properties of a Bezier curve:
   - It always passes through the first and last control points.
   - It lies within the convex hull (convex polygon boundary) of the control points.

NOTES

- The tangent to the curve at an endpoint is across the line joining that control point to the adjacent control point.

8. Cubic Bezier curves can be generated with four control points.
9. In Medicine, doctors can get trained by looking at a virtual surgery or they can simulate how the human body is affected by diseases spread by viruses and bacteria and then develop techniques to prevent it.
10. Text is one of the fundamental building blocks of multimedia applications.
11. Static bitmapped images are often compressed to reduce the file sizes and thus to save some disk space and shorten the time it takes to transfer those files over a communication link.
12. Interlace ratio is the ratio of the frame rate to the field rate.
13. An authoring system is a collection of software tools that help in various aspects of multimedia production.
14. Hypertext is the organisation of information units into connected associations that a user can choose to make. An instance of such an association is called a Hyperlink.
15. DVI technology is helpful to play back the full motion videos, multiple stereo sound tracks, live television shows, color graphics, etc.

5.12 SUMMARY

- In perspective projection the viewpoint or center of projection is at a finite distance from the viewplane or plane of projection.
- Parallel projection the angle made by the direction of projecting lines with the projection plane or view plane.
- Orthographic projections that show more than one side of an object are called axonometric orthographic projection. The most common axonometric projection can be considered as an isometric projection where each coordinate axis is intersected by the projection plane in the model coordinate system at an equal distance.
- A perspective transformation is determined by specifying a definite center of projection (COP) (also called viewpoint), and a plane of projection.
- A vanishing point is found by passing a line through the center of the projection, parallel to a set of parallel object edges. The point where the line pierces the projection plane is the vanishing point.
- A simple example of an object space algorithm is the back-face removal (also called the Back-Face cut) where no faces on the back of the object are displayed.
- The z-buffer or depth buffer is the simplest visible surface or hidden surface algorithm. it is an image space algorithm. The z-buffer is a simple extension of the frame buffer idea. A frame buffer is used to store the attributes of each pixel in the image space.

- Binary space partitioning is a generic process of recursively dividing a scene into two until the partitioning satisfies one or more requirements.
- The specific choice of partitioning plane and criterion for terminating the partitioning process varies depending on the purpose of the BSP tree.
- Binary space partitioning arose from the computer graphics need to rapidly draw three-dimensional scenes composed of polygons.
- A disadvantage of binary space partitioning is that generating a BSP tree can be time-consuming.
- BSP trees are often used by 3D video games, particularly first-person shooters and those with indoor environments.
- Multimedia refers to a mixture of interactive media or data types, predominantly text, graphics, audio and video simultaneously delivered by a computer.
- Multimedia is one of the most fascinating and fastest growing areas in the field of information technology. Text, pictures, animation, movies and sound– all these varied media are seamlessly blended, resulting in simple slide shows to dazzling, interactive presentations.
- An authoring system is a collection of software tools that help in various aspects of multimedia production.
- Text, containing words and symbols, is the most common form of communication.
- Hypertext is the organisation of information units into connected associations that a user can choose to make. An instance of such an association is called a Hyperlink.
- In the DVI-A system, the cables are used to transmit the DVI signal to display in the analog format for the LCDs and CRT monitors.
- In DVI-I, the cables are integrated and are used to transmit the digital source signal to a digital display or it can change the analog source to an analog display. It makes the DVI-I connection flexible and better than other DVI technologies.
- DVI technology is helpful to play back the full motion videos, multiple stereo sound tracks, live television shows, color graphics, etc.

5.13 KEY TERMS

- **Vanishing point:** It is found by passing a line through the center of the projection, parallel to a set of parallel object edges. The point where the line pierces the projection plane is the vanishing point.
- **Hidden Line and Surface Elimination:** A major consideration in the generation of realistic graphics display is identifying those parts of a scene which are visible from a selected viewing position.

- **Back-Face removal:** A simple example of an object space algorithm is Back-Face removal (also called Back-Face cut) where no faces on the back of the object are displayed.
- **Z-Buffer Method (Depth –Buffer):** The z-buffer or depth buffer is the simplest of the visible surface or hidden surface algorithms and it is an image space algorithm. The z-buffer is a simple extension of the frame buffer idea.
- **Multimedia:** It refers to a mixture of interactive media or data types, predominantly text, graphics, audio and video simultaneously delivered by a computer.

5.14 SELF-ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What is one-point perspective projection?
2. State about the computing vanishing points.
3. Define the term three-point perspective.
4. Write the limitations of Back-face removal algorithm.
5. Write the advantages and disadvantages of Z-Buffer Algorithm
6. What are the disadvantages of binary space partition?
7. Write the properties of Bezier curves.
8. Define the term cubic Bezier curves.
9. Define the term 'multimedia'.
10. What is the function of lingo software?
11. Name any two plug-in applications.
12. State the image and video standards.
13. How will you define graphic file formats?

Long-Answer Questions

1. Discuss the classification of geometric projection with the help of diagram.
2. Explain oblique projection by giving appropriate example.
3. Describe the perspective transformation and with the help of appropriate example.
4. What is the need for hidden-surface algorithms? How does the Z-buffer algorithm determine which surface are hidden?
5. Explain the Z-Buffer algorithm for hidden surface removal.
6. Describe the depth sort algorithm for hidden surface removal.

7. Explain the scan line algorithm for hidden surface removal.
8. Discuss about the A-Buffer visible surface algorithm.
9. Write an efficient routine to display two-dimensional, cubic Bezier curves, given a set of four control points in the xy plane.
10. Explain the Bezier surfaces with the help of diagram and example.
11. Discuss the various application areas of multimedia.
12. Describe all the hardware and software required for creating multimedia.
13. How is task planning done in multimedia? Explain with the help of an example.
14. Explain the significance of digital video and video compression with reference to multimedia.
15. Describe the various elements of the multimedia with the help of example.
16. What are different types of DVI? Describe each type.

5.15 FURTHER READING

Hearn, Donald and M. Pauline Baker. *Computer Graphics*. New Jersey: Prentice Hall.

Rogers, David F. *Procedural Elements for Computer Graphics*. New York: McGraw-Hill Higher Education.

Foley, James D., Andries van Dam, Steven K. Feiner and John F. Hughes. *Computer Graphics: Principles and Practice*. London: Addison-Wesley Professional.

Mukhopadhyay, Anirban and Arup Chattopadhyay. *Introduction to Computer Graphics and Multimedia*. New Delhi: Vikas Publishing House Pvt Ltd.