Table of Contents

Chapter 1 Introduction to Computer Graphics

1.1 Introduction 1
1.2 Image Processing as Picture Analysis 2
1.3 The Advantages of Interactive Graphics 3
1.4 Applications of Computer Graphics 4
1.5 Classification of Applications 4
1.6 Input Devices 5
1.6.1 Keyboard 5
1.6.2 Mouse 6
1.6.3 Trackball and Spaceball 7
1.6.4 Joysticks 7
1.6.5 Data Glove 8
1.6.6 Digitizer/Graphical Tablet 8
1.6.7 Image Scanners 9
1.6.8 Touch Panels 10
1.6.9 Light Pens 11
1.6.10 Voice Systems 11
1.7 Output Devices 12
1.7.1 Video Display Devices 12
1.7.1.1 Cathode-Ray-Tubes 12
1.7.1.2 Vector Scan/Random Scan Display 14
1.7.1.3 Raster Scan Display 15
1.7.1.4 Colour CRT Monitors 22
1.7.1.5 Direct-view Storage Tubes 24
1.7.1.6 Flat Panel Displays 25
1.7.1.7 Plasma Panel Display 25
1.7.1.8 Liquid Crystal Monitors 26
1.7.1.9 Important Characteristics of Video Display Devices 27
1.7.2 Hardcopy Devices 28
1.7.2.1 Important Characteristics of Hardcopy Devices 28

1.7.2.2 Printers 29
1.7.2.3 Plotters 34
1.7.2.4 Cameras 36
1.7.2.5 Comparison of Various Monochrome Hardcopy Devices 36
1.7.2.6 Comparison of Various Colour Hardcopy Devices 37
1.8 Coordinate Systems 37
1.8.1 Two Dimensional Cartesian Reference System 37
1.8.2 Three Dimensional Cartesian Reference System 38
1.8.3 Polar Coordinate System 39
1.9 Coordinate Representations 40
Solved Examples 41
Review Questions 42
University Questions 43

Chapter 2 Raster Graphics Algorithms for Drawing 2-D Primitives **45**

2.1 Introduction 45
2.2 Basic Concepts in Line Drawing 45
2.3 Line Drawing Algorithms 46
2.3.1 Digital Differential Analyzer 47
2.3.2 Bresenham's Line Algorithm 53
2.4 Antialiasing of Lines 59
2.5 Methods of Antialiasing 59
2.5.1 Increasing Resolution 59
2.5.2 Unweighted Area Sampling 60
2.5.3 Weighted Area Sampling 60
2.6 Thick Line Segments 61
2.7 Basic Concepts in Circle Drawing 63
2.8 Representation of a Circle 64
2.8.1 Polynomial Method 64
2.8.2 Trigonometric Method 64
2.9 Circle Drawing Algorithms 65
2.9.1 DDA Circle Drawing Algorithm 65
2.9.2 Bresenham's Circle Drawing Algorithm 67

2.9.3 Midpoint Circle Drawing Algorithm 71
2.10 Ellipse Drawing Algorithm 75
Solved Examples 82
Review Questions 84
University Questions 85

Chapter 3 Area Filling **88**

3.1 Introduction 88
3.2 Types of Polygons 88
3.3 Representation of Polygons 89
3.4 Entering Polygons 90
3.5 An Inside Test 91
3.6 Polygon Filling 93
3.6.1 Seed Fill 93
3.6.1.1 Boundary Fill Algorithm 93
3.6.1.2 Flood Fill Algorithm 95
3.6.2 Scan Line Algorithm 96
Solved Example 106
Review Questions 107
University Questions 108

Chapter 4 2-D Geometric Transformation **110**

4.1 Introduction 110
4.2 Two Dimensional Transformations 110
4.2.1 Translation 110
4.2.2 Rotation 111
4.2.3 Scaling 113
4.3 Homogeneous Coordinates 115
4.3.1 Homogeneous Coordinates for Translation 116
4.3.2 Homogeneous Coordinates for Rotation 116
4.3.3 Homogeneous Coordinates for Scaling 117
4.4 Composition of 2D Transformations 121
4.4.1 Rotation About an Arbitrary Point 121

4.5 Other Transformations 123
4.5.1 Reflection 123
4.5.2 Shear 125
4.5.2.1 X shear 125
4.5.2.2 Y shear 125
4.5.2.3 Shearing Relative to Other Reference Line 126
4.6 Inverse Transformations 130
Solved Examples 131
Review Questions 143
University Questions 144

Chapter 5 2-D Viewing and Clipping 147

5.1 Introduction 147
5.2 Viewing Transformation 147
5.2.1 Normalization Transformation 148
5.2.2 Workstation Transformation 149
5.3 2D Clipping 152
5.3.1 Point Clipping 153
5.3.2 Line Clipping 153
5.3.2.1 Sutherland and Cohen Subdivision Line Clipping Algorithm 153
5.3.2.2 Midpoint Subdivision Algorithm 162
5.3.2.3 Generalized Clipping with Cyrus-Beck Algorithm 167
5.3.2.4 Liang-Barsky Line Clipping Algorithm 173
5.4 Polygon Clipping 178
5.5 Sutherland - Hodgeman Polygon Clipping 179
5.6 Weiler-Atherton Algorithm 191
5.7 Generalized Clipping 192
5.8 Interior and Exterior Clipping 193
Solved Examples 193
Review Questions 195
University Questions 195

Chapter 6 3-D Concepts **198**

6.1 Introduction 198
6.2 Translation 198
6.3 Scaling 199
6.4 Rotation 200
6.5 Rotation about Arbitrary Axis 201
6.6 Reflection with Respect to Given Plane 208
6.6.1 Reflection with Respect to xy Plane 208
6.6.2 Reflection with Respect to Any Plane 208
Solved Examples 211
Review Questions 219
University Questions 219

Chapter 7 Three Dimensional Viewing, Projection and Clipping **221**

7.1 Introduction 221
7.2 Three Dimensional Viewing 221
7.3 Viewing Parameters 222
7.4 Transformation from World Coordinate to Viewing Coordinates 225
7.5 Projections 226
7.5.1 Parallel Projection 226
7.5.2 Perspective Projection 227
7.5.3 Types of Parallel Projections 227
7.5.3.1 Orthographic Projection 228
7.5.3.2 Oblique Projection 228
7.5.4 Types of Perspective Projections 229
7.5.5 Transformation Matrices for General Parallel Projection 231
7.5.5.1 On XY Plane 231
7.5.5.2 On Given View Plane 232
7.5.6 Transformation Matrix for Oblique Projection onto xy Plane 233
7.5.7 Transformation Matrix for Perspective Projection 234
7.6 Three Dimensional Clipping 235
7.7 Three - Dimensional Midpoint Subdivision Algorithm 239
Solved Examples 239

Review Questions 245
University Questions 246

Chapter 8 Hidden Surface Elimination Methods 248

8.1 Introduction 248
8.2 Techniques for Efficient Visible-Surface Algorithms 249
8.2.1 Coherence 249
8.2.2 Perspective Transformation 249
8.2.3 Extents and Bounding Volumes 250
8.2.4 Back-Face Culling 250
8.2.5 Spatial Partitioning 251
8.2.6 Hierarchy 251
8.3 Hidden Line Elimination Algorithms 251
8.3.1 Robert's Algorithm 251
8.3.2 Appel's Algorithm 252
8.3.3 Haloed Lines 253
8.4 Hidden Surface Elimination Algorithms 253
8.4.1 Painter's Algorithm (Depth Sort Algorithm) 253
8.4.2 Scan Line Algorithm 255
8.4.3 Z-Buffer Algorithm 256
8.4.4 Warnock's Algorithm (Area Subdivision Algorithm) 258
8.4.5 Back-Face Removal Algorithm 261
Review Questions 262
University Questions 263

Chapter 9 Curves 265

9.1 Introduction 265
9.2 Generation of Curves 265
9.2.1 Circular Arc Generation Using DDA Algorithm 265
9.2.2 Interpolation 267
9.3 Spline Representation 270
9.3.1 Geometric and Parametric Continuity 271
9.3.2 Spline Specifications 272
9.4 Bezier Curves 273

9.5 B-Spline Curves 279
9.6 Parametric Bicubic Surfaces 284
9.6.1 Hermite Surfaces 285
9.6.2 B-Spline Surfaces 286
9.6.3 Bezier Surface 286
Review Questions 286
University Questions 286

Chapter 10 Light Shading **288**

10.1 Introduction 288
10.2 Diffuse Illumination 288
10.3 Point-Source Illumination 290
10.4 Specular Reflection 291
10.4.1 The Phong Illumination Model 291
10.4.2 The Halfway Vector 293
10.5 Combined Diffuse and Specular Reflections 293
10.6 Shading Algorithms 294
10.6.1 Constant-Intensity Shading 294
10.6.2 Gouraud Shading 294
10.6.3 Phong Shading 297
10.6.4 Halftone Shading 299
10.6.5 Dithering Techniques 300
10.7 Transparency 301
10.8 Shadows 302
10.9 Ray-Tracing 303
10.9.1 Ray Surface Intersection Calculations 305
10.9.2 Reducing Object-Intersection Calculations 307
10.9.3 Antialiased Ray Tracing 309
10.9.3.1 Super Sampling 309
10.9.3.2 Adaptive Sampling 310
10.9.3.3 Stochastic Sampling / Distributed Ray Tracing 310
10.9.3.4 Advantages of Distributed Ray Tracing 310
10.10 Colour Models 312

10.10.1 Properties of Light 312
10.10.2 CIE Chromaticity Diagram 313
10.10.3 RGB Colour Model 315
10.10.4 CMY Colour Model 316
10.10.5 HSV Colour Model 317
10.10.6 HLS Colour Model 319
Review Questions 320
University Questions 321

Index **323**

1

Introduction to Computer Graphics

1.1 Introduction

The computer is an information processing machine. It is a tool for storing, manipulating and correlating data. There are many ways to communicate the processed information to the user. The computer graphics is one of the most effective and commonly used way to communicate the processed information to the user. It displays the information in the form of graphics objects such as pictures, charts, graphs and diagrams instead of simple text. Thus we can say that computer graphics makes it possible to express data in pictorial form. The picture or graphics object may be an engineering drawing, business graphs, architectural structures, a single frame from an animated movie or a machine parts illustrated for a service manual. It is the fundamental cohesive concept in computer graphics. Therefore, it is important to understand -

- How pictures or graphics objects are presented in computer graphics ?
- How pictures or graphics objects are prepared for presentation ?
- How previously prepared pictures or graphics objects are presented ?
- How interaction with the picture or graphics object is accomplished ?

In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called pixels. The pixel is the smallest addressable screen element. It is the smallest piece of the display screen which we can control. The control is achieved by setting the intensity and colour of the pixel which compose the screen. This is illustrated in Fig. 1.1.

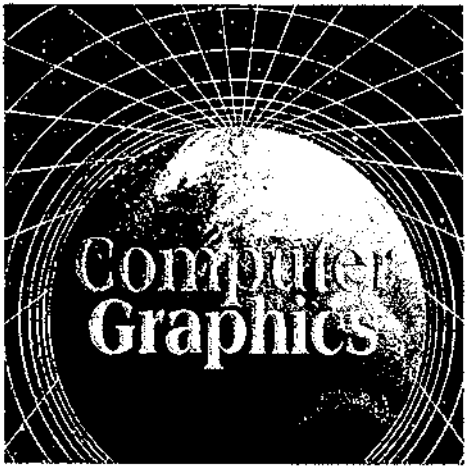


Fig. 1.1 Representation of picture

Each pixel on the graphics display does not represent mathematical point. Rather, it represents a region which theoretically can contain an infinite number of points. For example, if we want to display point P_1 whose coordinates are (4.2, 3.8) and point P_2

whose coordinates are (4.8, 3.1) then P_1 and P_2 are represented by only one pixel (4, 3), as shown in the Fig. 1.2. In general, a point is represented by the integer part of x and integer part of y, i.e., pixel (int (x), int (y)).

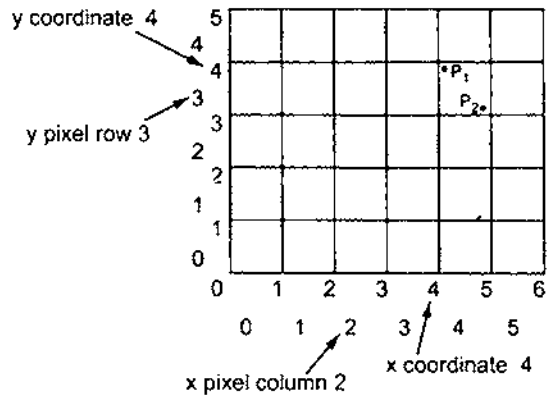


Fig. 1.2 Pixel display area of 6 × 5

The special procedures determine which pixel will provide the best approximation to the desired picture or graphics object. The process of determining the appropriate pixels for representing picture or graphics object is known as **rasterization**, and the process of representing continuous picture or graphics object as a collection of discrete pixels is called **scan conversion**.

The computer graphics allows rotation, translation, scaling and performing various projections on the picture before displaying it. It also allows to add effects such as hidden surface removal, shading or transparency to the picture before final representation. It provides user the control to modify contents, structure, and appearance of pictures or graphics objects using input devices such as a keyboard, mouse, or touch-sensitive panel on the screen. There is a close relationship between the input devices and display devices. Therefore, graphics devices includes both input devices and display devices.

1.2 Image Processing as Picture Analysis

The computer graphics is a collection, combination and representation of real or imaginary objects from their computer-based models. Thus we can say that computer graphics concerns the pictorial synthesis of real or imaginary objects. However, the related field image processing or sometimes called picture analysis concerns the analysis of scenes, or the reconstruction of models of 2D or 3D objects from their picture. This is exactly the reverse process. The image processing can be classified as

- Image enhancement
- Pattern detection and recognition
- Scene analysis and computer vision

The image enhancement deals with the improvement in the image quality by eliminating noise or by increasing image contrast. Pattern detection and recognition deal

with the detection and clarification of standard patterns and finding deviations from these patterns. The optical character recognition (OCR) technology is an practical example of pattern detection and recognition. Scene analysis and computer vision deals with the recognition and reconstruction of 3D model of scene from several 2D images.

The above three fields of image processing proved their importance in many area such as finger print detection and recognition, modeling of buildings, ships, automobiles etc., and so on.

We have discussed the two fields : computer graphics and image processing of computer processing of pictures. In the initial stages they were quite separate disciplines. But now a days they use some common features, and overlap between them is growing. They both use raster displays.

1.3 The Advantages of Interactive Graphics

Let us discuss the advantages of interactive graphics.

- Today, a high quality graphics displays of personal computer provide one of the most natural means of communicating with a computer.
- It provides tools for producing pictures not only of concrete, "real-world" objects but also of abstract, synthetic objects, such as mathematical surfaces in 4D and of data that have no inherent geometry, such as survey results.
- It has an ability to show moving pictures, and thus it is possible to produce animations with interactive graphics.
- With interactive graphics use can also control the animation by adjusting the speed, the portion of the total scene in view, the geometric relationship of the objects in the scene to one another, the amount of detail shown and so on.
- The interactive graphics provides tool called **motion dynamics**. With this tool user can move and tumble objects with respect to a stationary observer, or he can make objects stationary and the viewer moving around them. A typical example is walk throughs made by builder to show flat interior and building surroundings. In many case it is also possible to move both objects and viewer.
- The interactive graphics also provides facility called **update dynamics**. With update dynamics it is possible to change the shape, colour or other properties of the objects being viewed.
- With the recent development of digital signal processing (DSP) and audio synthesis chip the interactive graphics can now provide audio feedback alongwith the graphical feedbacks to make the simulated environment even more realistic.

In short, interactive graphics permits extensive, high-bandwidth user-computer interaction. It significantly enhances the ability to understand information, to perceive trends and to visualize real or imaginary objects either moving or stationary in a realistic environment. It also makes it possible to get high quality and more precise results and products with lower analysis and design cost.

1.4 Applications of Computer Graphics

The use of computer graphics is wide spread. It is used in various areas such as industry, business, government organisations, education, entertainment and most recently the home. Let us discuss the representative uses of computer graphics in brief.

- **User Interfaces :** User friendliness is one of the main factors underlying the success and popularity of any system. It is now a well established fact that graphical interfaces provide an attractive and easy interaction between users and computers. The built-in graphics provided with user interfaces use visual control items such as buttons, menus, icons, scroll bar etc, which allows user to interact with computer only by mouse-click. Typing is necessary only to input text to be stored and manipulated.
- **Plotting of graphics and chart :** In industry, business, government and educational organisations, computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in form of histograms, bars and pie-charts. These graphs and charts are very useful for decision making.
- **Office automation and Desktop Publishing :** The desktop publishing on personal computers allow the use of graphics for the creation and dissemination of information. Many organisations does the in-house creation and printing of documents. The desktop publishing allows user to create documents which contain text, tables, graphs and other forms of drawn or scanned images or pictures. This is one approach towards the office automation.
- **Computer-aided Drafting and Design :** The computer-aided drafting uses graphics to design components and systems electrical, mechanical, electromechanical and electronic devices such as automobile bodies, structures of building, airplane, ships, very large-scale integrated (VLSI) chips, optical systems and computer networks.
- **Simulation and Animation :** Use of graphics in simulation makes mathematic models and mechanical systems more realistic and easy to study. The interactive graphics supported by animation software proved their use in production of animated movies and cartoons films.
- **Art and Commerce :** There is a lot of development in the tools provided by computer graphics. This allows user to create artistic pictures which express messages and attract attentions. Such pictures are very useful in advertising.
- **Process Control :** By the use of computer now it is possible to control various processes in the industry from a remote control room. In such cases, process systems and processing parameters are shown on the computer with graphic symbols and identifications. This makes it easy for operator to monitor and control various processing parameters at a time.
- **Cartography :** Computer graphics is also used to represent geographic maps, weather maps, oceanographic charts, contour maps, population density maps and so on.

1.5 Classification of Applications

In the last section we have seen various uses of computer graphics. These uses can be classified as shown in the Fig. 1.3. As shown in the Fig. 1.3, the use of computer graphics can be classified according to dimensionality of the object to be drawn : 2D or 3D. It can also be classified according to kind of picture : Symbolic or Realistic. Many computer graphics

applications are classified by the type of interaction. The type of interaction determines the user's degree of control over the object and its image. In controllable interaction user can change the attributes of the images. Role of picture gives the another classification. Computer graphics is either used for representation or it can be an end product such as drawings. Pictorial representation gives the final classification of use computer graphics. It classifies the use of computer graphics to represent pictures such as line drawing, black and white, colour and so on.

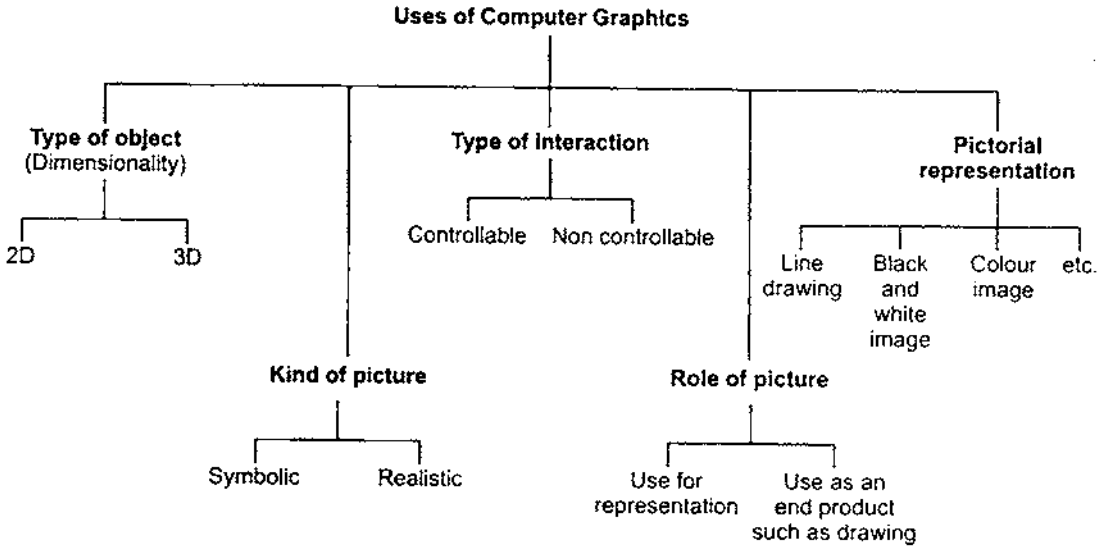


Fig. 1.3

1.6 Input Devices

Number of devices are available for data input in the graphics systems. These include keyboard, mouse, trackball, spaceball, joystick, digitizers, scanners and so on. Let us discuss them.

1.6.1 Keyboard

The keyboard is a primary input device for any graphics system. It is used for entering text and numbers, i.e. on graphics data associated with pictures such as labels x-y coordinates etc.

Keyboards are available in various sizes, shapes and styles. Fig. 1.4 shows standard keyboard. It consists of

- Alphanumeric key
- Function keys
- Modifier keys
- Cursor movement keys
- Numeric keypad

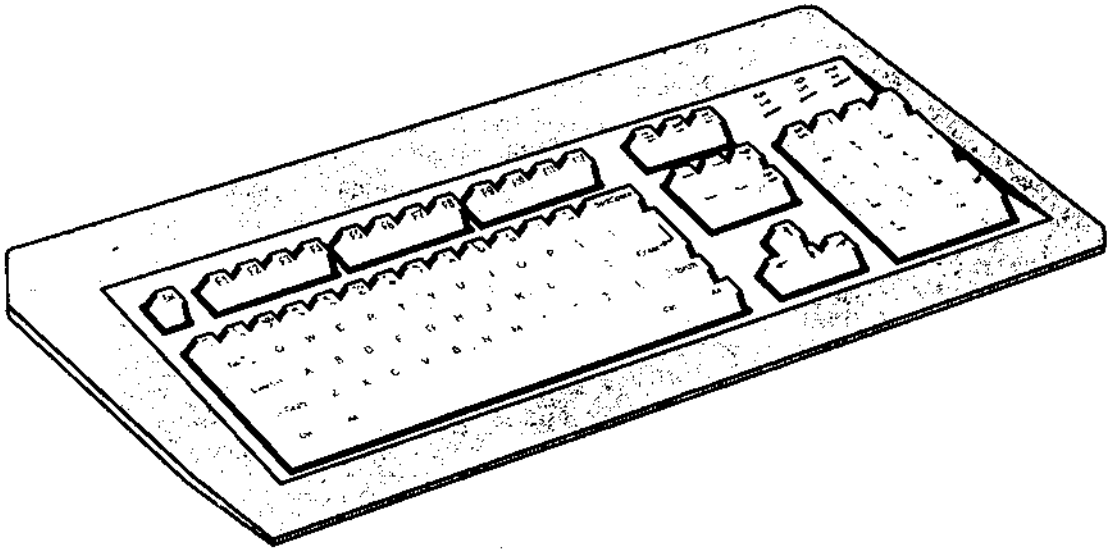


Fig. 1.4

When we press a key on the keyboard, keyboard controller places a code corresponding to key pressed into a part of its memory, called **keyboard buffer**. This code is called **scan code**. The keyboard controller informs CPU of the computer about the key press with the help of an interrupt signal. The CPU then reads the scan code from the keyboard buffer, as shown in the Fig. 1.5.

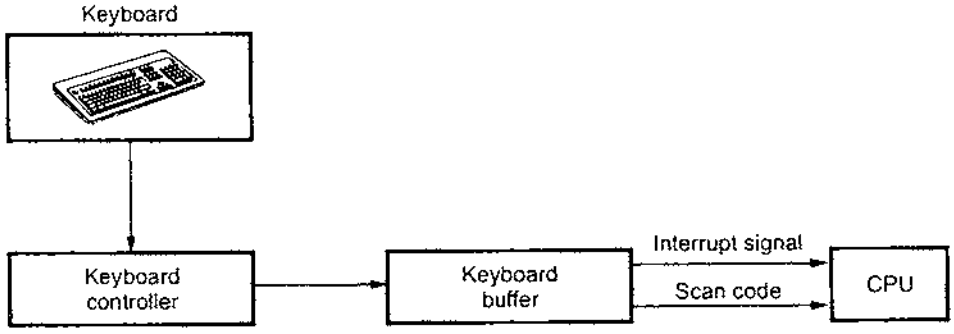


Fig. 1.5 Getting the scan code from keyboard

1.6.2 Mouse

A mouse is a palm-sized box used to position the screen cursor. It consists of ball on the bottom connected to wheels or rollers to provide the amount and direction of movement. One, two or three buttons are usually included on the top of the mouse for signaling the execution of some operation. Now-a-days mouse consists of one more wheel on the top to scroll the screen pages.

(a) Mouse

(b) Mouse with scrolling wheel

Fig. 1.6

1.6.3 Trackball and Spaceball

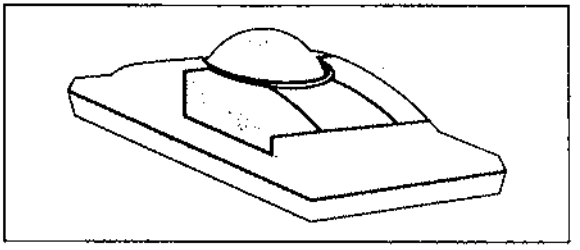


Fig. 1.7 Trackball

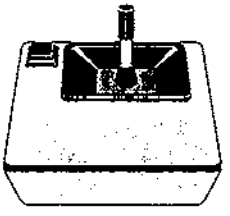
As the name implies, a trackball is a ball that can be rotated with the fingers or palm of the hand to produce screen cursor movement. The potentiometers attached to the trackball are used to measure the amount and direction of rotation.

The trackball is a two dimensional positioning device whereas spaceball provides six degree of freedom. It does not actually move. It consists of strain guages which measure the amount of pressure applied to the spaceball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. It is usually used in three-dimensional positioning and selecting operations in virtual-reality systems.

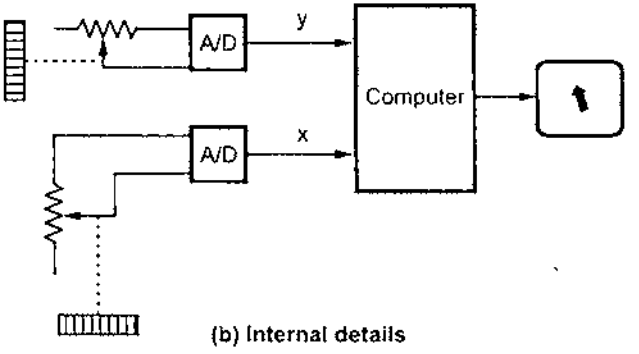
1.6.4 Joysticks

A joystick has a small, vertical lever (called the stick) mounted on the base and used to steer the screen cursor around. It consists of two potentiometers attached to a single lever. Moving the lever changes the settings on the potentiometers. The left or right movement is indicated by one potentiometer and forward or back movement is indicated by other potentiometer. Thus with a joystick both x and y-coordinate positions can be simultaneously altered by the motion of a single lever. This is illustrated in Fig. 1.8.

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



(a) Joystick



(b) Internal details

Fig. 1.8

1.6.5 Data Glove

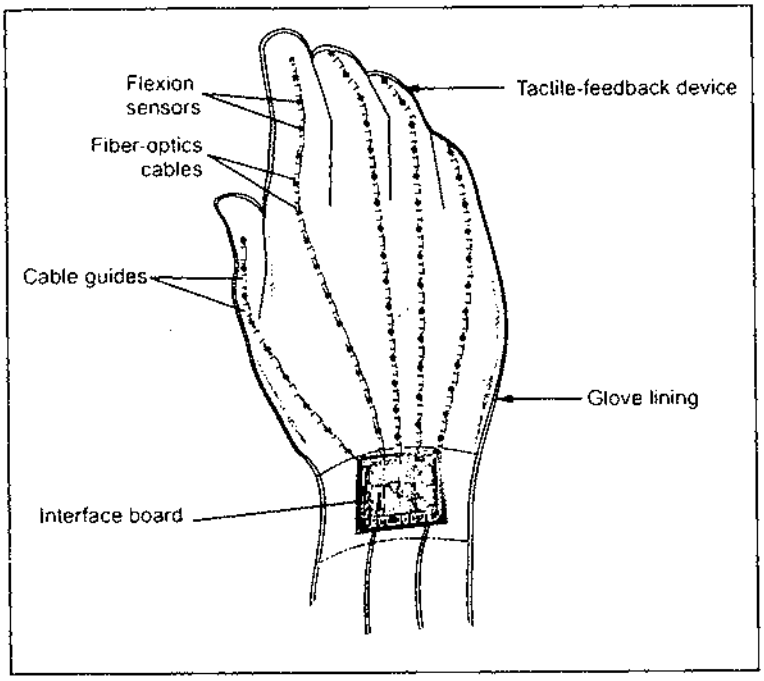


Fig. 1.9

The data glove is used to grasp a virtual object. The Fig. 1.9 shows the data glove. It is constructed with a series of sensors that detect hand and finger motions. Each sensor is a short length of fiberoptic cable, with a light-emitting diode (LED) at one end and a phototransistor at the other end. The surface of a cable is roughened in the area where it is to be sensitive to bending. When the cable is flexed, some of the LED's light is lost, so less light is received by the phototransistor.

The input from the glove can be used to position or manipulate objects in a virtual scene. Thus by wearing the dataglove, a user can grasp, move and rotate objects and then release them.

1 6.6 Digitizer/Graphical Tablet

For applications such as tracing we need a device called a **digitizer** or a **graphical tablet**. It consists of a flat surface, ranging in size from about 6 by 6 inches up to 48 by 72 inches or more, which can detect the position of a movable stylus. Fig. 1.10 shows a small tablet with penlike stylus.

Different graphics tablets use different techniques for measuring position, but they all resolve the position into a horizontal and a vertical direction, which correspond to the axes of the display. Most graphics tablets use an electrical sensing mechanism to determine the position of the stylus. In one such arrangement, a grid of wire on 1/4 to 1/2 inch centers is embedded in the tablet surface. Electromagnetic signals generated by electrical pulses applied in sequence to the wires in the grid induce an electrical signal in a wire coil in the stylus. The strength of the signal induced by each pulse

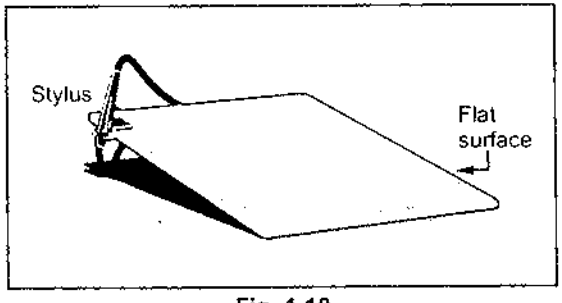


Fig. 1.10

is used to determine the position of the stylus. The signal strength is also used to determine roughly how far the stylus or cursor is from the graphical tablet.

Every time user may not wish to enter stylus position into the computer. In such cases user can lift the stylus or make the tablet off by pressing a switch provided on the stylus.

Other graphical tablet technologies use sound (sonic) coupling and resistive coupling. The sonic tablet uses sound waves to couple the stylus to microphones positioned on the periphery of the digitizing area. Sound brust are created by an electrical spark at the tip of the stylus. The time between when the spark occurs and when its sound arrives at each microphone is proportional to the distance from the stylus to each mi. rophone. The sonic tablets mainly used in 3D positioning the devices. The resistive tablet uses a battery powered stylus that emits high-frequency radio signals. The tablet is a piece of glass coated with a thin layer of conducting material in which an electrical potential is induced by the radio signals. The strength of the signals at the edges of the tablet is inversely proportional to the distance to the stylus and can thus be used to calculate the stylus position.

1.6.7 Image Scanners

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

Fig. 1.11 shows the working of photoscanner. As shown in the Fig. 1.11, the photograph is mounted on a rotating drum. A finely collimated light beam is directed at the photo, and the amount of light reflected is measured by a photocell. As the drum rotates, the light source slowly moves from one end to the other, thus doing a raster scan of the entire photograph.

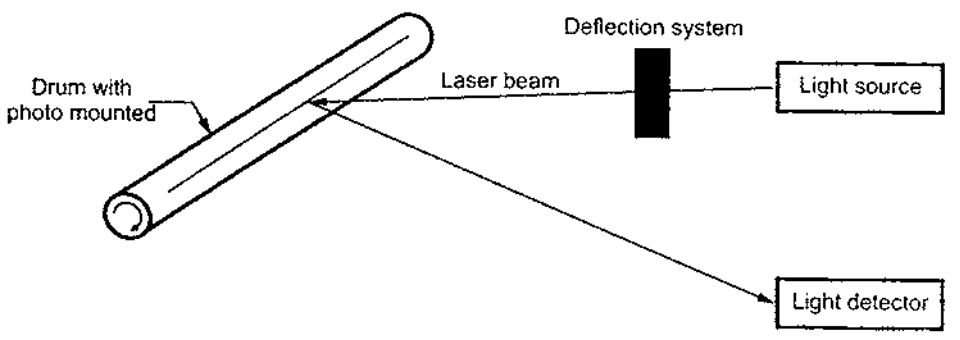


Fig. 1.11 Photoscanner

For coloured photographs, multiple passes are made, using filters in the front of the photocell to separate out various colours.

Other type of scanners are electro-optical devices that use arrays of light sensitive charge coupled devices (CCDs) to turn light reflected from, or transmitted through, artwork, photographs, slides etc. into a usable digital file composed of pixel information.

The optical resolution and colour depth are the two important specifications of the scanner. The photoscanners have resolution upto 2000 units per inch. Resolution of the CCD array is 200 to 1000 units per inch which is less than the photoscanners. The colour depth is expressed in bits. It specifies the number of colours scanner can capture.

According to construction the scanners also can be classified as :

Flatbed scanners, also called desktop scanners, are the most versatile and commonly used scanners. In fact, this article focuses on the technology as it relates to flatbed scanners.

Sheet-fed scanners are similar to flatbed scanners except the document is moved and the scan head is immobile. A sheet-fed scanner looks a lot like a small portable printer.

Handheld scanners use the same basic technology as a flatbed scanner, but relay on the user to move them instead of a motorized belt. This type of scanner typically does not provide good image quality. However, it can be useful for capturing an image quickly.

Drum scanners are used by the publishing industry to capture incredibly detailed images. They use a technology called a **photomultiplier tube** (PMT). In PMT, the document to be scanned is mounted on a glass cylinder. Located at the center of the cylinder is a sensor that splits light bounced from the document into three beams. Each beam is sent through a colour filter into a photomultiplier tube where the light is changed into an electrical signal.

1.6.8 Touch Panels

As the name implies, touch panels allow displayed objects or screen positions to be selected with the touch of a finger. The touch panels are the transparent devices which are fitted on the screen. They consist of touch sensing mechanism. Touch input can be recorded using optical, electrical or acoustical methods.

Optical touch panels use a line of infrared light emitting diodes along one vertical edge and along one horizontal edge of the screen. The opposite vertical and horizontal edges contain light detectors. These detectors are used to record which beams are interrupted when the panel is touched. The two crossing beams that are interrupted indicate the horizontal and vertical coordinates of the screen position selected.

In electrical touch panels, two transparent plates are used. These plates are separated by a small distance. One plate is coated with conducting material and other plate is coated with resistive material. When outer plate is touched, it is forced to contact with the inner plate. This contact creates a voltage drop across the resistive plate that is used to determine the coordinate values of the selected screen position.

An acoustic touch panels use high frequency sound waves in horizontal and vertical directions across a glass plate. Touching the screen causes partial reflection of each wave from finger to the emitter. The screen coordinates of point of contact are then calculated by measuring time between the transmission of each wave and its reflection to the emitter.

1.6.9 Light Pens

Light pen is a pencil shaped device used to select positions by detecting the light coming from points on the CRT screen. It consists of a photoelectric cell housed in a pencil like case as shown in the Fig. 1.12.

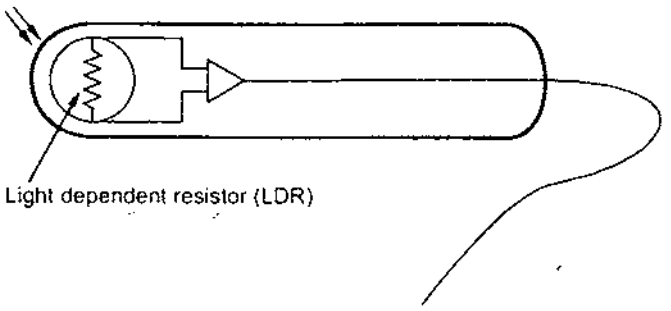


Fig. 1.12

The light pen may be pointed at the screen. An optical system focuses light on to the photo cell in the field view of the pen. The pen will send a pulse whenever the phosphor below it is illuminated. This output of the photo-cell is then amplified and carried over a shielded cable to light pen interface. A 'detect' by the light pen can either be used to cause an immediate interrupt to the computer through an interface or be used to set a flip-flop which is cleared when read by the computer. Thus, the system can identify the part of the graphics to which the pen is pointing. It records the co-ordinate position of the electron beam.

Light pen is an event driven device. After displaying each point, one can test the light pen flip-flop. Thus exact location of the spot to which the light pen is pointing can be detected. For using light pen for positioning, a tracking program is run on the computer. The display processor also has the capability to disable the light pen during the refresh cycle. This ensures the inputs not desired by the operator are ignored. To facilitate the operation of the light pen, a finger operated switch is provided to control the light reaching the photo cell.

The response time of the pen is also important. For slow displays transistor type photo cells such as photo diodes are used. These are small, inexpensive and suitable for hand held operation. The response time is about one microsecond. For the highspeed displays photo-multiplier tube is used. It is bulky and uses fiber-optic cable.

Now a days improved light pens are available. These consist of a matrix of fibre optic sensors.

1.6.10 Voice Systems

Speech recognizers are used in some graphics systems as input devices to accept voice commands. Such a voice-system input can be used to initiate graphics operations or to enter data. These systems operate by matching an input with a predefined dictionary of words and phrases.

The voice recognizers are classified accordings to whether or not they must be trained to recognize the waveforms of a particular speaker, and whether they can recognize connected

speech as opposed to single words or phrases. The speaker independent recognizers have very limited vocabularies. Usually, they include only the digits and 50 to 100 words.

The Fig. 1.13 shows the typical voice recognition system. In such systems microphone is also included to minimize input of other backgrond sounds.

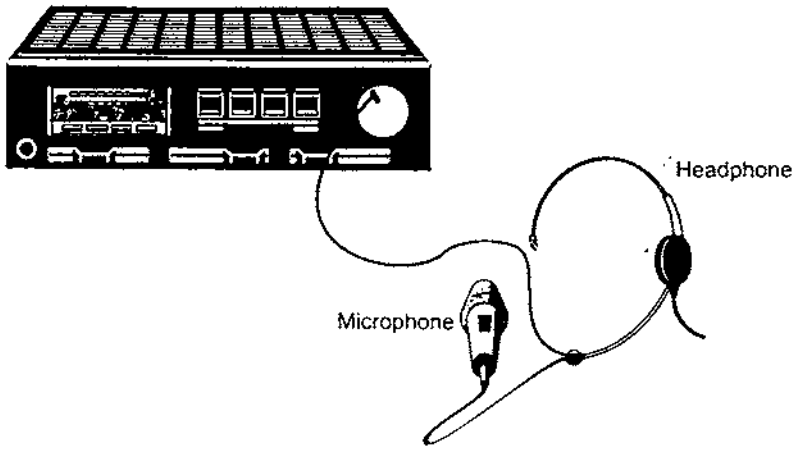


Fig. 1.13

The one advantage of voice system over other devices is that in voice systems the attention of the operator does not have to be switched from one device to another to enter a command.

1.7 Output Devices

The output devices can be classified as display devices and hardcopy devices. Let us see some of them.

1.7.1 Video Display Devices

The most commonly used output device in a graphics system is a video monitor. The operation of most video monitors is based on the standard **cathode-ray-tube** (CRT) design. Let us see the basics of the CRT.

1.7.1.1 Cathode-Ray-Tubes

A CRT is an evacuated glass tube. An electron gun at the rear of the tube produces a beam of electrons which is directed towards the front of the tube (screen). The inner side of the screen is coated with phosphor substance which gives off light when it is stroked by electrons. This is illustrated in Fig. 1.14. It is possible to control the point at which the electron beam strikes the screen, and therefore the position of the dot upon the screen, by deflecting the electron beam. The Fig. 1.15 shows the electrostatic deflection of the electron beam in a CRT.

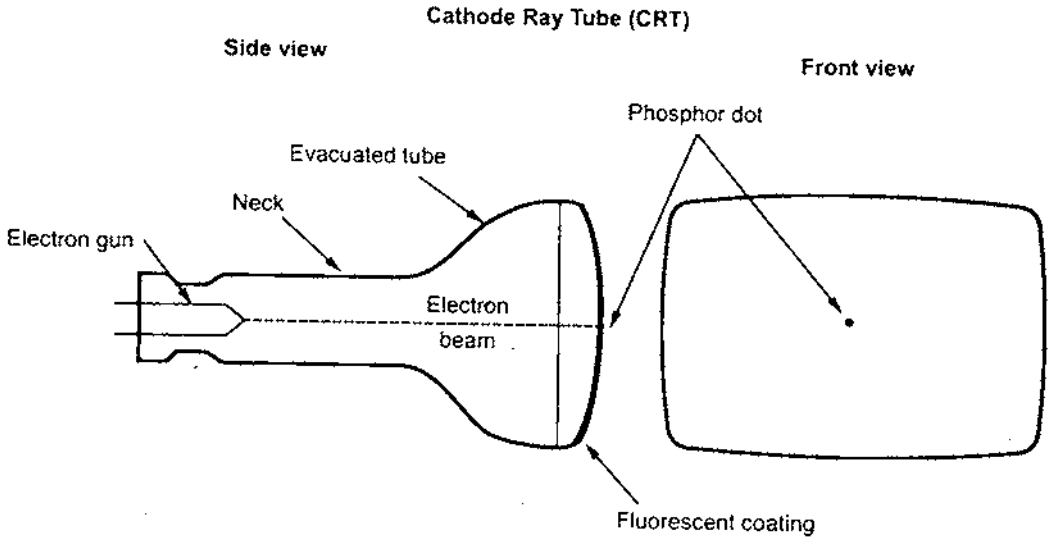


Fig. 1.14 Simplified representation of CRT

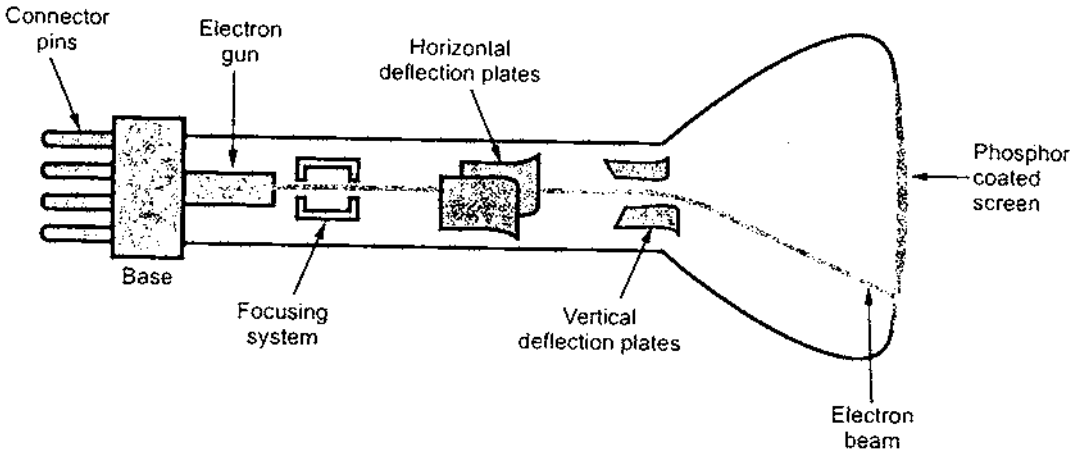


Fig. 1.15 Cathode Ray Tube

The deflection system of the cathode-ray-tube consists of two pairs of parallel plates, referred to as the vertical and horizontal deflection plates. The voltage applied to vertical plates controls the vertical deflection of the electron beam and voltage applied to the horizontal deflection plates controls the horizontal deflection of the electron beam. There are two techniques used for producing images on the CRT screen : Vector scan/random scan and Raster scan.

1.7.1.2 Vector Scan/Random Scan Display

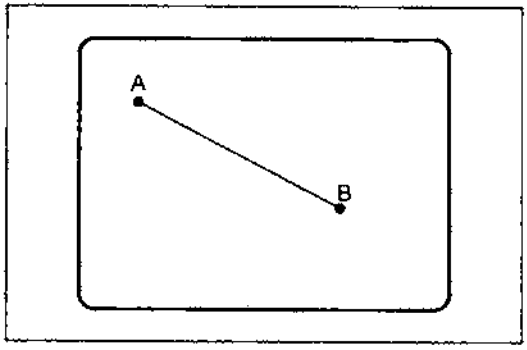


Fig. 1.16 Vector scan CRT

As shown in Fig. 1.16, vector scan CRT display directly traces out only the desired lines on CRT i.e. If we want a line connecting point A with point B on the vector graphics display, we simply drive the beam deflection circuitry, which will cause beam to go directly from point A to B. If we want to move the beam from point A to point B without showing a line between points, we can blank the beam as we move it. To move the beam across the CRT, the information about both, magnitude and direction is required. This information is generated with the help of vector graphics generator.

The Fig. 1.17 shows the typical vector display architecture. It consists of display controller, Central Processing Unit (CPU), display buffer memory and a CRT. A display controller is connected as an I/O peripheral to the central processing unit (CPU). The display buffer memory stores the computer produced display list or display program. The program contains point and line plotting commands with (x, y) or (x, y, z) end point coordinates, as well as character plotting commands. The display controller interprets commands for plotting points, lines and characters and sends digital and point coordinates to a vector generator. The vector generator then converts the digital coordinate values to analog voltages for beam-deflection circuits that displace an electron beam writing on the CRT's phosphor coating.

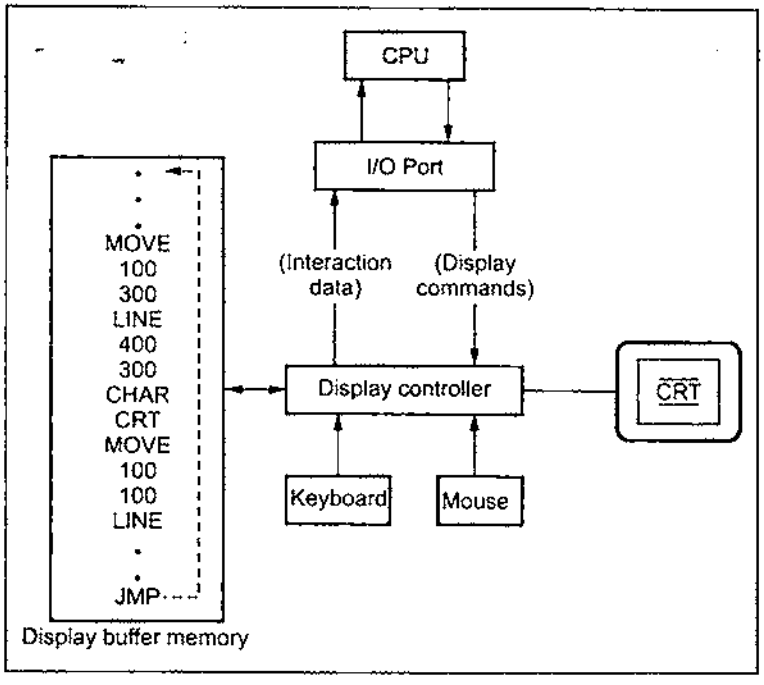


Fig. 1.17 Architecture of a vector display

In vector displays beam is deflected from end point to end point, hence this technique is also called **random scan**. We know as beam, strikes phosphor it emits light. But phosphor light decays after few milliseconds and therefore it is necessary to repeat through the display list to refresh the phosphor at least 30 times per second to avoid flicker. As display buffer is used to store display list and it is used for refreshing, the display buffer memory is also called **refresh buffer**.

1.7.1.3 Raster Scan Display

The Fig. 1.18 shows the architecture of a raster display. It consists of display controller, central processing unit (CPU), video controller, refresh buffer, keyboard, mouse and the CRT.

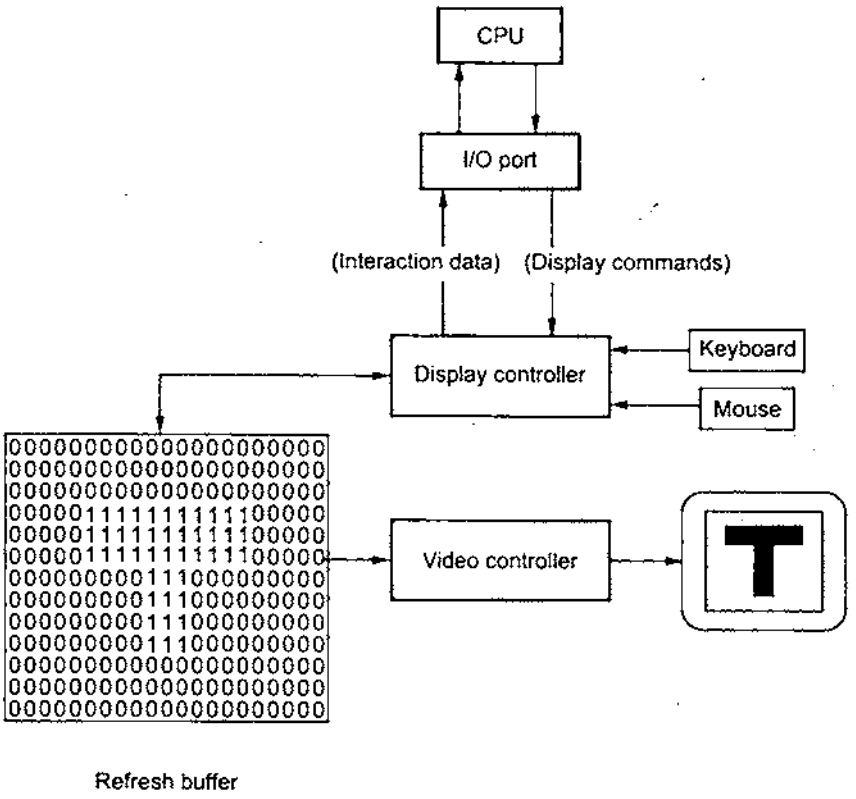


Fig. 1.18 Architecture of a raster display

As shown in the Fig. 1.18, the display image is stored in the form of 1s and 0s in the refresh buffer. The video controller reads this refresh buffer and produces the actual image on the screen. It does this by scanning one scan line at a time, from top to bottom and then back to the top, as shown in the Fig. 1.18.

Raster scan is the most common method of displaying images on the CRT screen. In this method, the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern shown in the Fig. 1.19

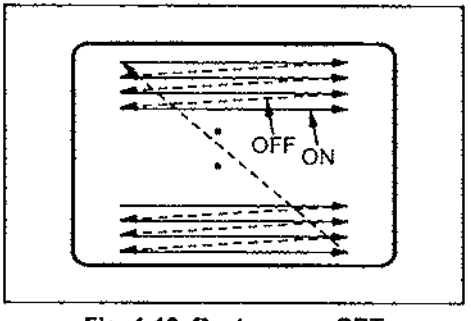


Fig. 1.19 Raster scan CRT

Here, the beam is swept back and forth from the left to the right across the screen. When the beam is moved from the left to the right, it is ON. The beam is OFF, when it is moved from the right to the left as shown by dotted line in Fig. 1.19

When the beam reaches the bottom of the screen, it is made OFF and rapidly retraced back to the top left to start again. A display produced in this way is called **raster scan display**. Raster scanning process is similar to reading different lines on the page of a book. After completion of scanning of one line, the electron beam flies back to the start of next line and process repeats. In the raster scan display, the screen image is maintained by repeatedly scanning the same image. This process is known as **refreshing of** screen.

In raster scan displays a special area of memory is dedicated to graphics only. This memory area is called **frame buffer**. It holds the set of intensity values for all the screen points. The stored intensity values are retrieved from frame buffer and displayed on the screen one row (scan line) at a time. Each screen point is referred to as a **pixel** or **pel** (shortened forms of picture element). Each pixel on the screen can be specified by its row and column number. Thus by specifying row and column number we can specify the **pixel position** on the screen.

Intensity range for pixel positions depends on the capability of the raster system. It can be a simple black and white system or colour system. In a simple black and white system, each pixel position is either on or off, so only one bit per pixel is needed to control the intensity of the pixel positions. Additional bits are required when colour and intensity variations can be displayed. Upto 24 bits per pixel are included in high quality display systems, which can require several megabytes of storage space for the frame buffer. On a black and white system with one bit per pixel, the frame buffer is commonly called a **bitmap**. For systems with multiple bits per pixel, the frame buffer is often referred to as a **pixmap**.

Vector Scan Display	Raster Scan Display
1. In vector scan display the beam is moved between the end points of the graphics primitives.	1. In raster scan display the beam is moved all over the screen one scan line at a time, from top to bottom and then back to top.
2. Vector display flickers when the number of primitives in the buffer becomes too large.	2. In raster display, the refresh process is independent of the complexity of the image.
3. Scan conversion is not required.	3. Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixels in the frame buffer.
4. Scan conversion hardware is not required.	4. Because each primitive must be scan-converted, real time dynamics is far more computational and requires separate scan conversion hardware.
5. Vector display draws a continuous and smooth lines.	5. Raster display can display mathematically smooth lines, polygons, and boundaries of curved primitives only by approximating them with pixels on the raster grid.
6. Cost is more.	6. Cost is low.
7. Vector display only draws lines and characters.	7. Raster display has ability to display areas filled with solid colours or patterns.

Table 1.1

Frame Buffer Organization

In raster scan displays a special area of memory is dedicated to graphics only. This memory area is called frame buffer. It holds the set of intensity values for all the screen points. The stored intensity values are retrieved from frame buffer and displayed on the screen one row (scan line) at a time.

Usually, frame buffer is implemented using rotating random access semiconductor memory. However, frame buffer also can be implemented using shift registers. Conceptually, shift register is operated as first-in, first-out (FIFO) fashion, i.e. similar to stack. We know that, when stack is full and if we want to add new data bit then first data bit is pushed out from the bottom and then the new data bit is added at the top. Here, the data pushed out of the stack can be interpreted as the intensity of a pixel on a scan line.

Fig. 1.20 shows the implementation of frame buffer using shift register. As shown in the Fig. 1.20, one shift register is required per pixel on a scan line and the length of shift register in bits is equal to number of scan lines. Here, there are 8 pixels per scan line and there are in all 5 scan lines. Therefore, 8 shift registers, each of 5 bit length are used to implement frame buffer. The synchronization between the output of the shift register and the video scan rate is maintained data corresponding to particular scan line is displayed correctly.

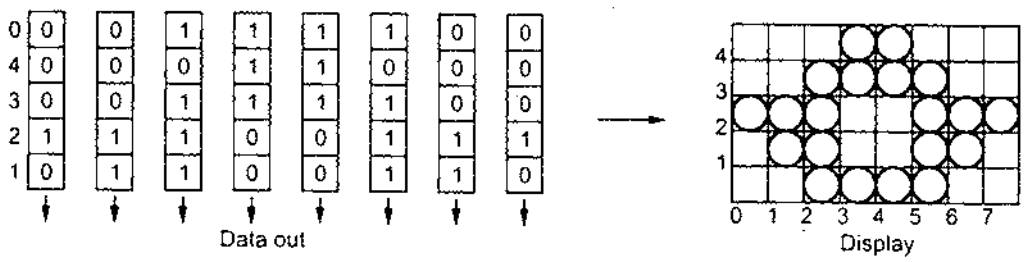


Fig. 1.20 Frame buffer using eight 5-bit shift registers

Both rotating memory and shift register frame buffer implementations have low levels of interactivity. The interactivity in rotating memory is limited due to disk access time and it is reduced in shift register implementations because changes can only be made as bits are being added to the register.

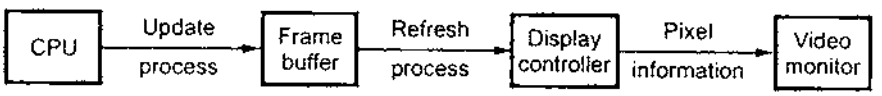
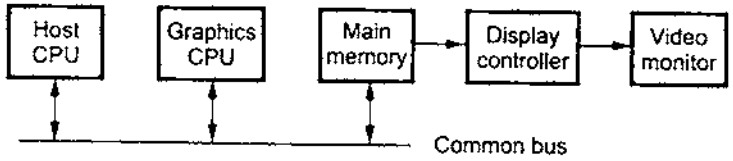


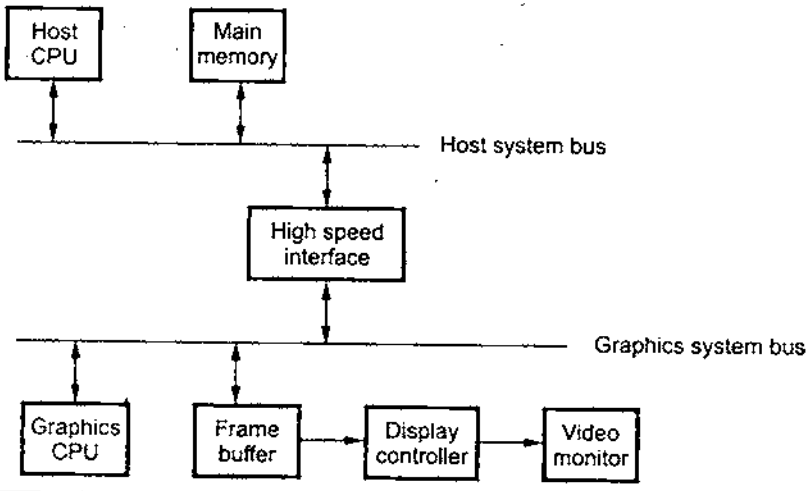
Fig. 1.21 Frame buffer graphics system

Fig. 1.21 shows the frame buffer graphics system. It consists of CPU, frame buffer, display controller and video monitor. An application program running in the computer updates the frame buffer as per the picture information. The display controller cycles through the frame buffer in scan line order (top to bottom) and passes the corresponding information to the video monitor to refresh the display. The frame buffer can be part of

computer memory itself or it can be implemented with separate memory as shown in the Fig. 1.22.



(a) Frame buffer as a part of computer memory



(b) Separate frame buffer

Fig. 1.22 Frame buffer architectures

Generally, separate graphics processor is used to improve the performance of graphics system. The graphics processor manipulates the frame buffer as per commands issued by main processor.

The performance of the graphics system is also affected by sharing of single memory done by two processors. The performance of the graphics system thus can be improved by having separate frame buffer memory as shown in the Fig. 1.22 (b).

Display File and its Structure

We know that in raster scan displays image information is stored in the frame buffer. It includes information of all pixels. On the other hand, the vector refresh displays store only the commands necessary for drawing the line segments. Here, input to the vector generator is stored instead of the output. The file used to store the commands necessary for drawing the line segments is called **display file.**

In vector refresh display system, display processor uses the information in the display file to draw lines with the help of vector generating algorithms. This is illustrated in Fig. 1.23. Therefore, we can say that display files provides an interface between the image specification process and the image display process. It also describes image in a compact

format. The concept of display file may be applied to devices other than refresh displays. Such files are called pseudo display files, or metafiles.

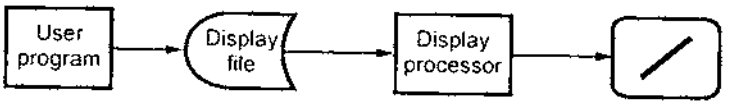


Fig. 1.23 Vector refresh display system

The Fig. 1.23 shows the structure of display file. We know that it contains series of commands. Each display file command contains two fields, an **operation code (opcode)** and **operands**. Opcode identifies the command such as draw line, move cursor etc., and operand provides the coordinates of a point to process the command.

One way to store opcode and operands of series of commands is to use three separate arrays. One for operation code, one for operand 1, i.e. x coordinate and one for operand2, i.e. y coordinate. This is illustrated in Fig. 1.24. The display file stores all commands to be needed to create a specified image. It is necessary to assign meaning to the possible operation codes before we can proceed to interpret them. Let us consider three commands MOVE, LINE and PLOT, and assign opcodes to these commands as shown in table 1.2.

Command	Opcode
MOVE	1
LINE	2
PLOT	3

Table 1.2

Once the opcodes are defined we can write commands needed to draw image into the display file. The following algorithm gives the steps required to insert command in the display file.

Algorithm

1. Read opcode, x and y coordinates.
2. Search for empty space in the display file. Let i be the empty row.
3. DF_OP [i] ← Opcode;
   DF_x [i] ← x ;
   DF_y [i] ← y ;
4. Stop

The table 1.3 shows the structure of display file with five commands in it, and Fig. 1.24 shows how these commands are interpreted and plotted.

DF_OP	DF_x	DF_y
Opcode	Operand 1 x	Operand 2 y
1	30	30
2	50	80
2	70	30
1	40	55
2	60	55

Table 1.3 Display file structure

Command 1 : 1, 30, 30

It moves current cursor position to point (30, 30)

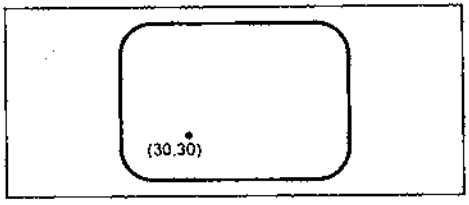


Fig. 1.24 (a)

Command 2 : 2, 50, 80

It draws the line between point (50, 80) and the current cursor position (30, 30)

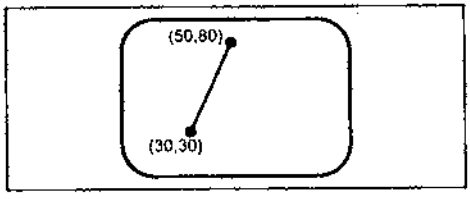


Fig. 1.24 (b)

Command 3 : 2, 70, 30

It draws the line between point (70, 30) and the current cursor position (50, 80)

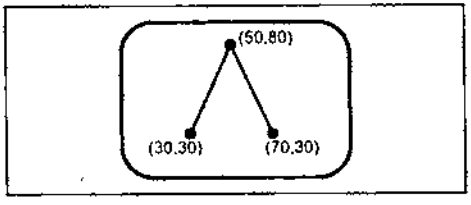


Fig. 1.24 (c)

Command 4 : 1, 40, 55

It moves current cursor position to point (40, 55)

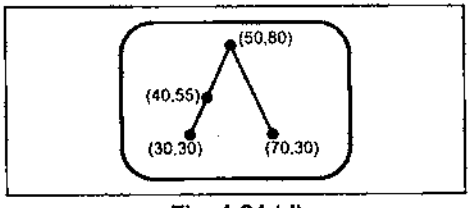


Fig. 1.24 (d)

Command 5 : 2, 60, 55

It draws the line between point (60, 55) and the current cursor position (40, 55)

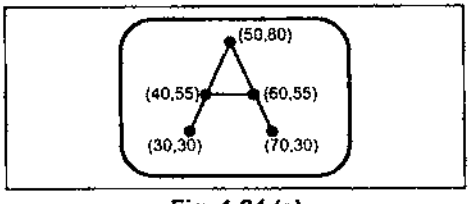


Fig. 1.24 (e)

Display File Interpreter

We have seen that display file contains information necessary to construct the picture. This information is in the form of commands. The program which converts these commands into actual picture is called display file interpreter. It is an interface between graphics representation in the display file and the display device, as shown in the Fig. 1.25.

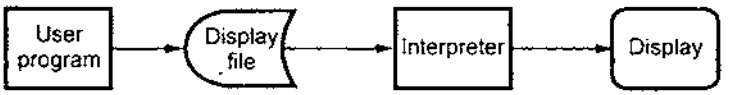


Fig. 1.25 Display file and interpreter

As shown the Fig. 1.25, the display process is divided into two steps : first the image is stored in the display file structure and then it is interpreted by an appropriate interpreter to get the actual image.

Instead of this, if we store actual image for particular display device it may not run on other displays. To achieve the device independence the image is stored in the raw format, i.e. in the display file format and then it is interpreted by an appropriate interpreter to run on required display.

Another advantage of using interpreter is that saving raw image takes much less storage than saving the picture itself.

Display Controller

In some graphics systems a separate computer is used to interpret the commands in the display file. Such computer is known as display controller. Display controller access display file and it cycles through each command in the display file once during every refresh cycle Fig. 1.26 shows the vector scan system with display controller.

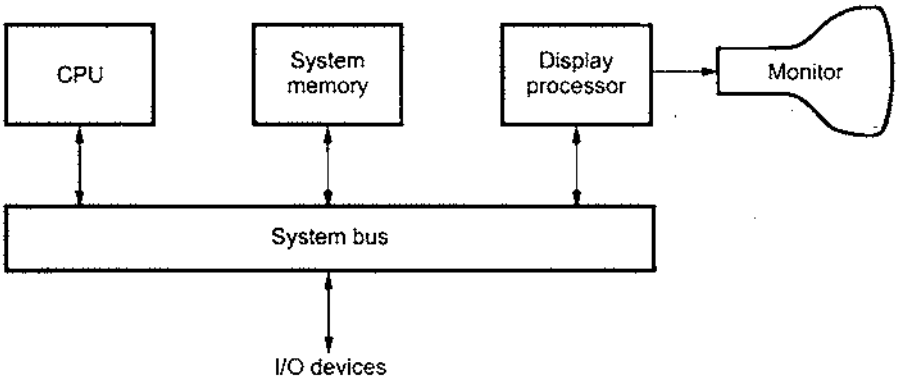


Fig. 1.26 Vector scan system

In the raster scan display systems, the purpose of display controller is to free the CPU from the graphics routine task. Here, display controller is provided with separate memory area as shown in the Fig. 1.26. The main task of display controller is to digitize a picture definition given in an application program into a set of pixel-intensity values for storage in the frame buffer. This digitization process is known as **scan conversion.**

Display controller are also designed to perform a number of additional operations. These operations include

- Generating various line styles (dashed, dotted, or solid)
- Display colour areas
- Performing certain transformations and
- Manipulations on displayed objects

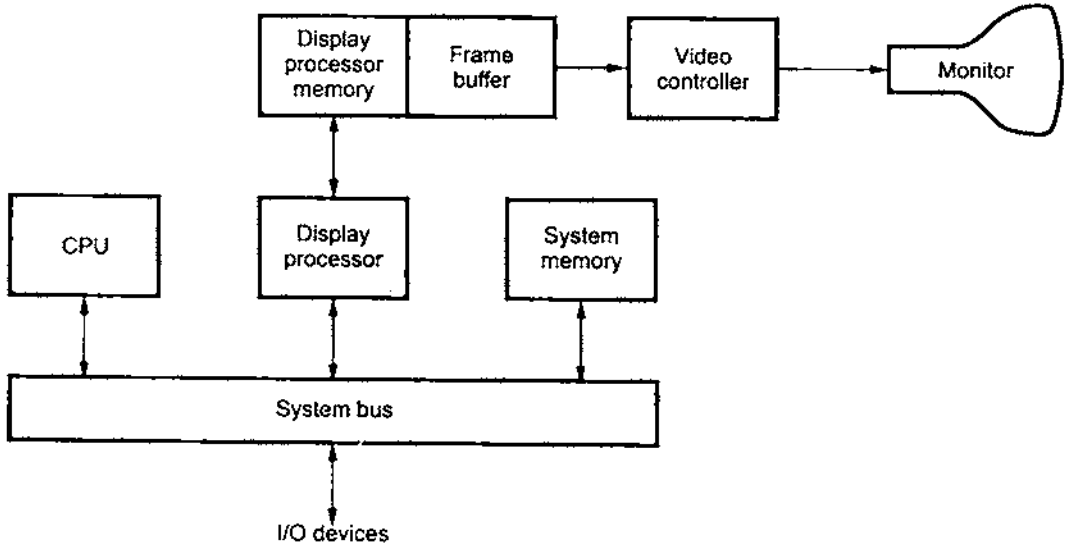


Fig. 1.27 Raster scan system with a display processor

1.7.1.4 Colour CRT Monitors

A CRT monitor displays colour pictures by using a combination of phosphors that emit different-coloured light. It generates a range of colours by combining the emitted light from the different phosphors. There are two basic techniques used for producing colour displays :

- Beam-penetration technique and
- Shadow-mask technique

Beam-penetration Technique

This technique is used with random-scan monitors. In this technique, the inside of CRT screen is coated with two layers of phosphor, usually red and green. The displayed colour depends on how far the electron beam penetrates into the phosphor layers. The outer layer is of red phosphor and inner layer is of green phosphor. A beam of slow electrons excites only the outer red layer. A beam of very fast electrons penetrates through the red layer and excites the inner green layer. At intermediate beam speeds, combinations of red and green light are emitted and two additional colours, orange and yellow displayed. The beam acceleration voltage controls the speed of the electrons and hence the screen colour at any point on the screen.

Merits and Demerits

- It is an inexpensive technique to produce colour in random scan monitors.
- It can display only four colours.
- The quality of picture produced by this technique is not as good as compared to other techniques.

Shadow Mask Technique

The shadow mask technique produces a much wider range of colours than the beam penetration technique. Hence this technique is commonly used in raster-scan displays including colour TV. In a shadow mask technique, CRT has three phosphor colour dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. The Fig. 1.28 shows the shadow mask CRT. It has three electron guns, one for each colour dot, and a shadow mask grid just behind the phosphor coated screen.

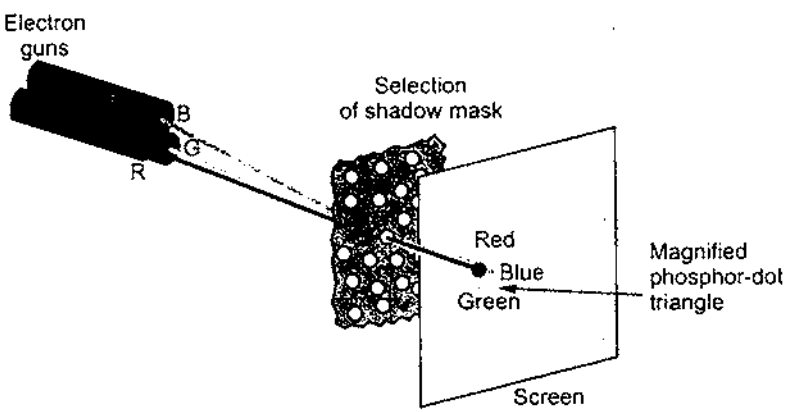


Fig. 1.28

The shadow mask grid consists of series of holes aligned with the phosphor dot pattern. As shown in the Fig. 1.28, three electron beams are deflected and focused as a group onto the shadow mask and when they pass through a hole in the shadow mask, they excite a dot triangle. A dot triangle consists of three small phosphor dots of red, green and blue colour. These phosphor dots are arranged so that each electron beam can activate only its

corresponding colour dot when it passes through the shadow mask. A dot triangle when activated appears as a small dot on the screen which has colour of combination of three small dots in the dot triangle. By varying the intensity of the three electron beams we can obtain different colours in the shadow mask CRT.

1.7.1.5 Direct-view Storage Tubes

We know that, in raster scan display we do refreshing of the screen to maintain a screen image. The direct-view storage tubes give the alternative method of maintaining the screen image. A direct-view storage tube (DVST) uses the storage grid which stores the picture information as a charge distribution just behind the phosphor-coated screen.

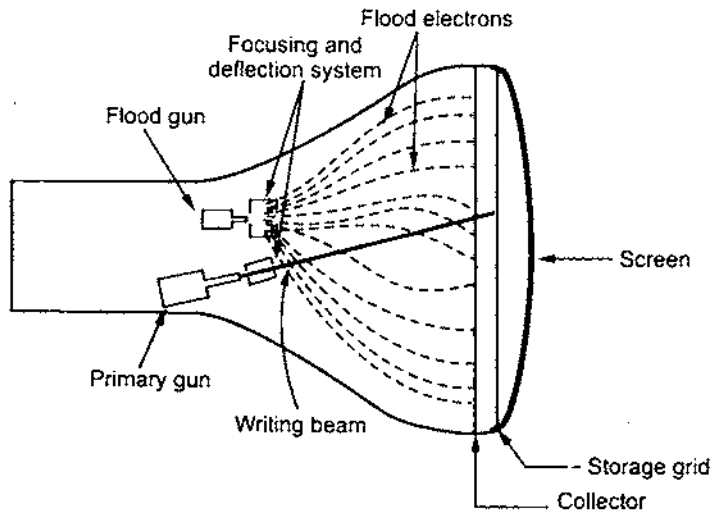


Fig. 1.29 Arrangement of the DVST

The Fig. 1.29 shows the general arrangement of the DVST. It consists of two electron guns: a primary gun and a flood gun.

A primary gun stores the picture pattern and the flood gun maintains the picture display.

A primary gun produces high speed electrons which strike on the storage grid to draw the picture pattern. As electron beam strikes on the storage grid with high speed, it knocks out electrons from the storage grid keeping the net positive charge. The knocked out electrons are attracted towards the collector. The net positive charge on the storage grid is nothing but the picture pattern. The continuous low speed electrons from flood gun pass through the control grid and are attracted to the positive charged areas of the storage grid. The low speed electrons then penetrate the storage grid and strike the phosphor coating without affecting the positive charge pattern on the storage grid. During this process the collector just behind the storage grid smooths out the flow of flood electrons.

Advantages of DVST

1. Refreshing of CRT is not required.
2. Because no refreshing is required, very complex pictures can be displayed at very high resolution without flicker.
3. It has flat screen.

Disadvantages of DVST

1. They do not display colours and are available with single level of line intensity.
2. Erasing requires removal of charge on the storage grid. Thus erasing and redrawing process takes several seconds.
3. Selective or part erasing of screen is not possible.
4. Erasing of screen produces unpleasant flash over the entire screen surface which prevents its use of dynamic graphics applications.
5. It has poor contrast as a result of the comparatively low accelerating potential applied to the flood electrons.
6. The performance of DVST is some what inferior to the refresh CRT.

1.7.1.6 Flat Panel Displays

The term flat-panel display refers to a class of video devices that have reduced volume, weight, and power requirements compared to a CRT. The important feature of flat-panel display is that they are thinner than CRTs. There are two types of flat panel displays : emissive displays and nonemissive displays.

Emissive displays : They convert electrical energy into light energy. Plasma panels, thin-film electro luminescent displays, and light emitting diodes are examples of emissive displays.

Nonemissive displays : They use optical effects to convert sunlight or light from some other source into graphics patterns. Liquid crystal display is an example of nonemissive flat panel display.

1.7.1.7 Plasma Panel Display

Plasma panel display writes images on the display surface point by point, each point remains bright after it has been intensified. This makes the plasma panel functionally very similar to the DVST eventhough its construction is markedly different.

The Fig. 1.30 shows the construction of plasma panel display. It consists of two plates of glass with thin, closely spaced gold electrodes. The gold electrodes are attached to the inner faces and covered with a dielectric material. These are attached as a series of vertical conducting ribbons on one glass plate, and a set of horizontal ribbons to the other glass plate. The space between two glass plates is filled with neon-based gas and sealed. By applying voltages between the electrodes the gas within the panel is made to behave as if it were divided into tiny cells, each one independent of its neighbours. These independent cells are made to glow by placing a firing voltage of about 120 volts across it by means of the electrodes. The glow can be sustained by maintaining a high frequency alternating voltage of about 90 volts across the cell. Due to this refreshing is not required.

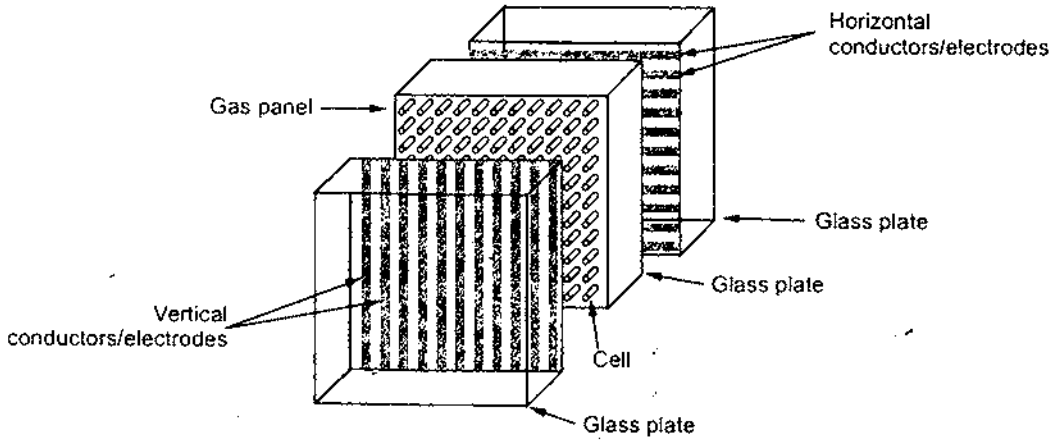


Fig. 1.30 Construction of plasma panel display

Advantages

1. Refreshing is not required.
2. Produces a very steady image, totally free of flicker.
3. Less bulky than a CRT.
4. Allows selective writing and selective erasing, at speed of about 20 μsec per cell.
5. It has the flat screen and is transparent, so the displayed image can be superimposed with pictures from slides or other media projected through the rear panel.

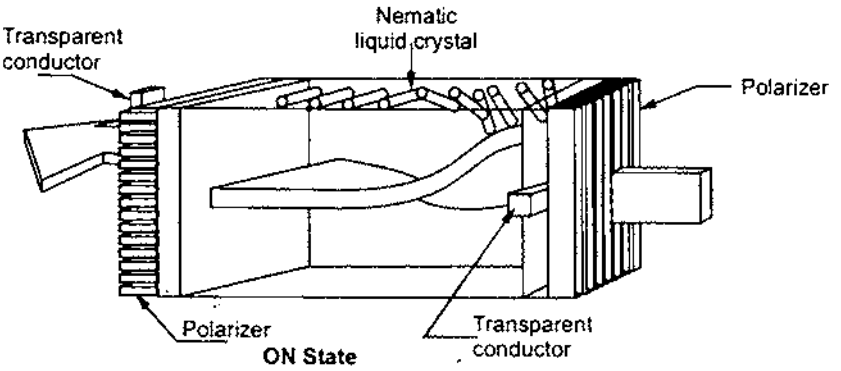
Disadvantages

1. Relatively poor resolution of about 60 dots per inch.
2. It requires complex addressing and wiring.
3. Costlier than the CRTs.

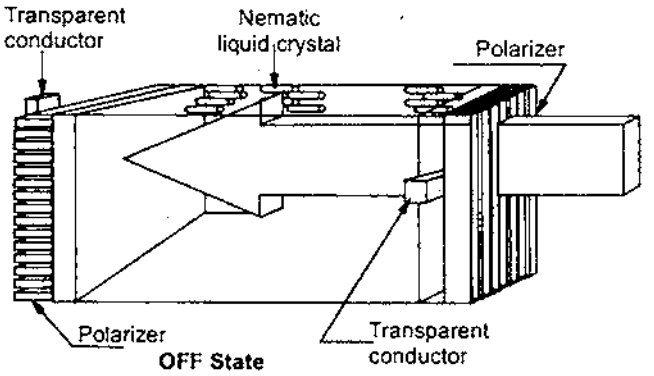
1.7.1.8 Liquid Crystal Monitors

The term liquid crystal refers to the fact that these compounds have a crystalline arrangement of molecules, yet they flow like a liquid. Flat panel displays commonly use nematic (thread like) liquid-crystal compounds that tend to keep the long axes of the rod-shaped molecules aligned.

Two glass plates, each containing a light polarizer at right angles to the other plate sandwich the liquid-crystal material. Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate. The intersection of two conductors defines a pixel position. In the ON state, polarized light passing through material is twisted so that it will pass through the opposite polarizer. It is then reflected back to the viewer. To turn OFF the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that the light is not twisted as shown in the Fig. 1.31. This type of flat panel device is referred to as a passive matrix LCD.



(a) Field effect display 'ON State'



(b) Field effect display 'OFF State'

Fig. 1.31

1.7.1.9 Important Characteristics of Video Display Devices

Persistence : The major difference between phosphors is their persistence. It decides how long they continue to emit light after the electron beam is removed. Persistence is defined as the time it takes the emitted light from the screen to decay to one-tenth of its original intensity. Lower persistence phosphors require higher refreshing rates to maintain a picture on the screen without flicker. However it is useful for displaying animations. On the other hand higher persistence phosphors are useful for displaying static and highly complex pictures.

Resolution : Resolution indicates the maximum number of points that can be displayed without overlap on the CRT. It is defined as the number of points per centimeter that can be plotted horizontally and vertically.

Resolution depends on the type of phosphor, the intensity to be displayed and the focusing and deflection systems used in the CRT.

Aspect Ratio : It is the ratio of vertical points to horizontal points to produce equal length lines in both directions on the screen. An aspect ratio of 4/5 means that a vertical line plotted with four points has the same length as a horizontal line plotted with five points.

1.7.2 Hardcopy Devices

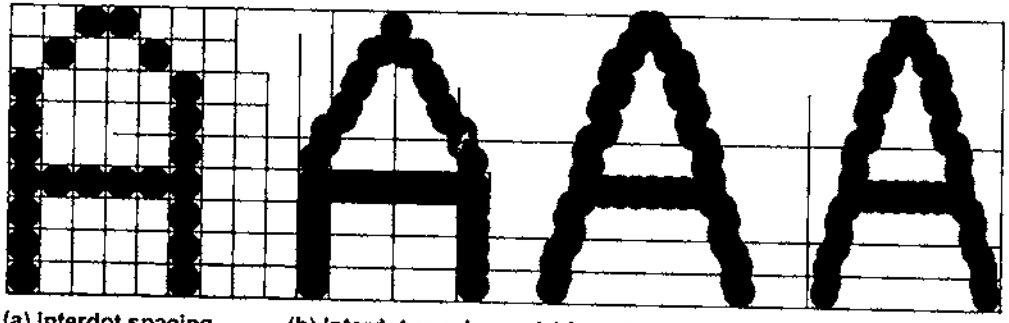
We can obtain hard-copy output for our image in several formats using printer or plotter. Therefore, printers and plotters are also called hard-copy devices. The quality of pictures obtained from a hard-copy device depends on dot size and the number of dots per inch, or lines per inch, that can be printed, i.e. it depends on the resolution of printer or plotter. Before going to see working principle of various plotters and printers we see the important characteristics of hardcopy devices.

1.7.2.1 Important Characteristics of Hardcopy Devices

Dot Size : It is the diameter of a single dot on the device's output. It is also referred to as **spot size**.

Addressability : It is the number of individual dots (not necessarily distinguishable) per inch that can be created. If the address of current dot is (x, y), then address of next dot in the horizontal direction is given as (x+1, y). Similarly, the address of next dot in vertical direction is (x, y+1).

Interdot distance : It is the reciprocal of addressability. If addressability is large the interdot distance is less. The interdot distance should be less to get smooth shapes, as shown in the Fig. 1.32.



(a) Interdot spacing equal to dot size (b) Interdot spacing one half dot size (c) Interdot spacing one-thrid dot size (d) Interdot spacing one-quarter dot size

Fig. 1.32

Resolution : It is the number of distinguishable lines per inch that a device can create. It depends on a dot size and the cross-sectional intensity distribution of a spot. A spot with sharply delineated edges yields higher resolution than does one with edge that trail off, as shown in the Fig. 1.33.

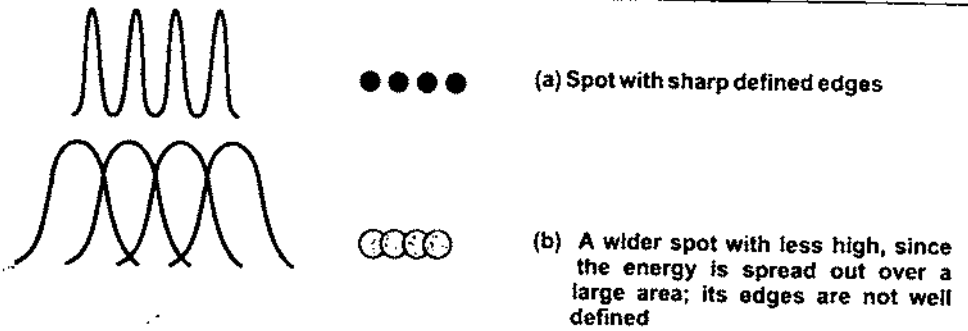


Fig. 1.33

1.7.2.2 Printers

Printers can be classified according to their printing methodology : **Impact printers** and **Non-impact printers**. Impact printers press formed character faces against an inked ribbon onto the paper. A line printer and dot matrix printer are the examples of an impact printers. Non impact printers and plotters use laser techniques, ink-jet sprays, xerographic processes, electrostatic methods, and electrothermal methods to get images onto the paper. An ink-jet printer and laser printer are the examples of non-impact printers.

Line Printers

A line printer prints a complete line at a time. The printing speed of line printer vary from 150 lines to 2500 lines per minute with 96 to 100 characters on one line. The line printers are divided into two categories : Drum printers and chain printer.

Drum Printers

A drum printers consists of a cylindrical drum. One complete set of characters is embossed on all the print positions on a line, as shown in the Fig. 1.34. The character to be printed is adjusted by rotating drum.

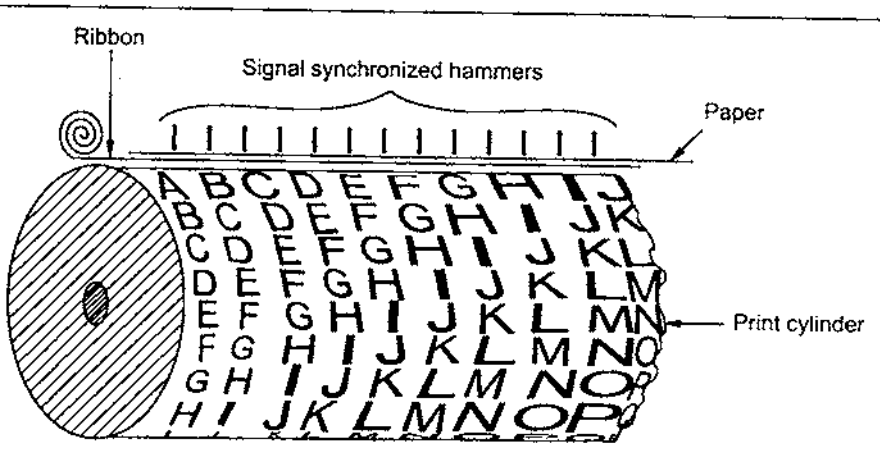


Fig. 1.34 Cylinder of a drum printer

The codes of all characters to be printed on line are transmitted from the memory of the computer to a printer memory, commonly known as **printer buffer**. This printer buffer can store 132 characters. A print drum is rotated with high speed and when printer buffer information matches with the drum character, character is printed by striking the hammer. Thus to print one line drum has to rotate one full rotation. A carbon ribbon and paper are in between the hammer and the drum therefore when hammer strikes the paper an impression is made on the backside of the paper by the ribbon mounted behind the paper. In drum printers to get good impression of the line on paper it is necessary to synchronize the movements of drum and the hammer.

Chain Printers

In these printers chain with embossed character set is used, instead of drum. Here, the character to be printed is adjusted by rotating chain. To print line, computer loads the code of all characters to be printed on line into print buffer. The chain rotated and when character specified in the print buffer appears in front of hammer, hammer strikes the carbon ribbon. A carbon ribbon is placed between the chain, paper and hammer. In this printer to get good printing quality the movement of hammer and chain must be synchronized.

Dot Matrix Printers

Dot matrix printers are also called **serial printers** as they print one character at a time, with printing head moving across a line. In dot matrix printer the print head consists of a 9×7 array of pins. As per the character definition pin are moved forward to form a character and they hit the carbon ribbon in front of the paper thereby printing that character, as shown in Fig. 1.35.

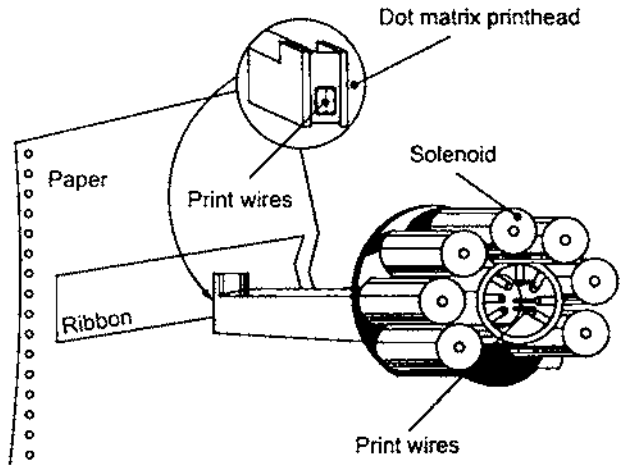
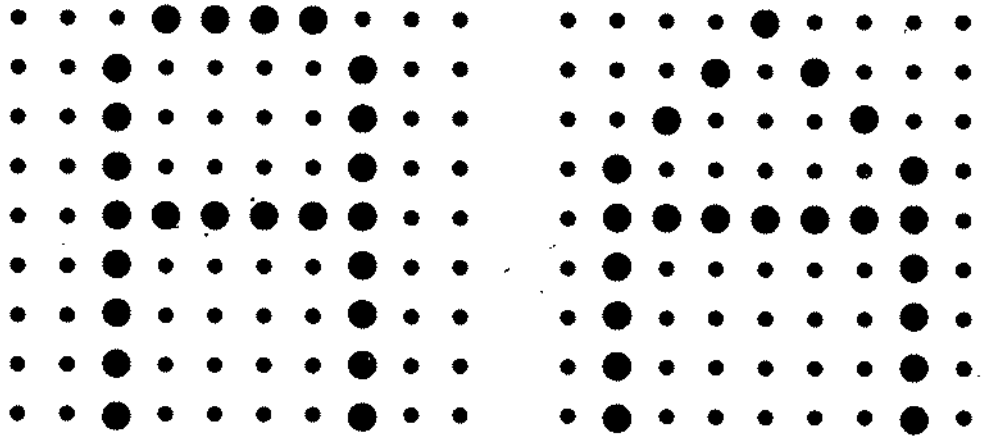


Fig. 1.35

In these printers character definition can be changed to get different font as shown in the Fig. 1.36.



(a) Dot pattern for A (b) Dot pattern for A

Fig. 1.36

An other advantage of dot matrix printers is that they can print alphabets other than English, such as Devangari, Tamil etc.

Comparison between line printer and dot matrix printer

	Line printer	Dot matrix printer
1)	Prints one line at a time.	Prints a character at a time.
2)	Characters are embossed on the drum or chain.	Characters are formed by combination of dots.
3)	Characters can not be printed with different fonts.	Characters can be printed with various fonts.
4)	Better printing quality.	Poor printing quality as characters are formed by combination of dots.
5)	Better printing speed.	Poor printing speed.
6)	Heavy duty printers.	Light duty printers.

Ink Jet Printer

An ink-jet printer places extremely small droplets of ink onto paper to create an image. If we ever look at a piece of paper that has come out of an ink-jet printer, we know that : the dots are extremely small (usually between 50 and 60 microns in diameter), so small that they are thinner than the diameter of a human hair (70 microns). The dots are positioned very precisely, with resolutions of up to 1440 × 720 dots per inch (dpi). The dots can have different colours combined together to create photo-quality images.

Ink jet printers print directly on paper by spraying ink through tiny nozzles as shown in the Fig. 1.37.

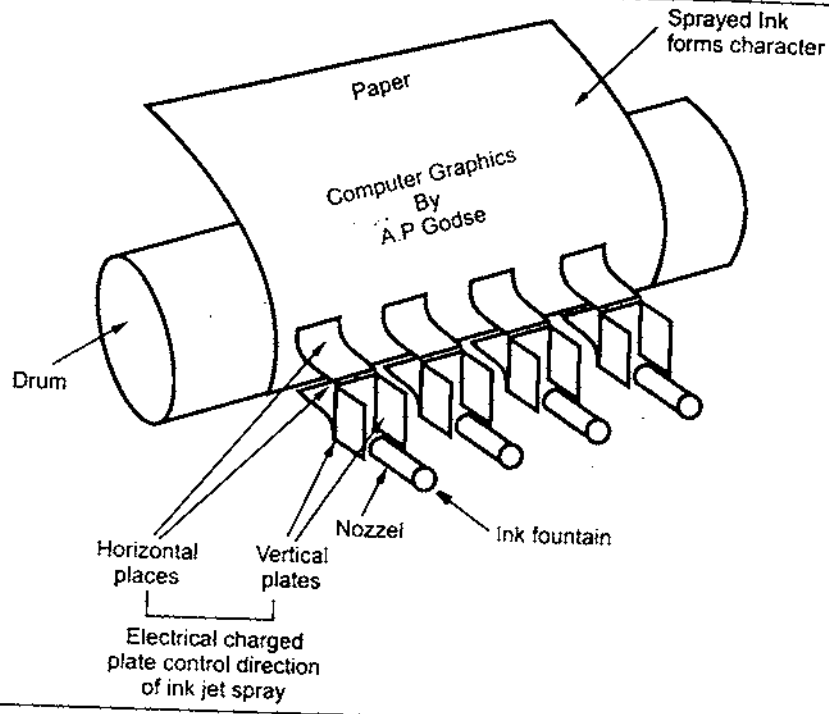


Fig. 1.37 Ink jet printer

As shown in the Fig. 1.37, the ink is deflected by an electric field with the help of horizontal and vertical charged plates to produce dot matrix patterns.

Features of ink-jet printer

1. They can print from two to four pages per minute
2. Resolution is about 360 dots per inch, therefore better printing quality is achieved.
3. The operating cost is quite low, the only part that needs replacement is the ink cartridge .
4. Colour ink jet printers have four ink nozzles with colours cyan, magenta, yellow and black, because it is possible to combine these colours to create any colour in the visible spectrum.

Laser Printer

The line, dot matrix, and ink jet printers need a head movement on a ribbon to print characters. This mechanical movement is relatively slow due to the high inertia of mechanical elements. In laser printers these mechanical movements are avoided. In these printers, an electronically controlled laser beam traces out the desired character to be printed on a photoconductive drum. The exposed areas of the drum gets charged, which attracts an oppositely charged ink from the ink toner on to the exposed areas. This image is

then transferred to the paper which comes in contact with the drum with pressure and heat, as shown in the Fig. 1.38. The charge on the drum decides the darkness of the print. When charge is more, more ink is attracted and we get a dark print.

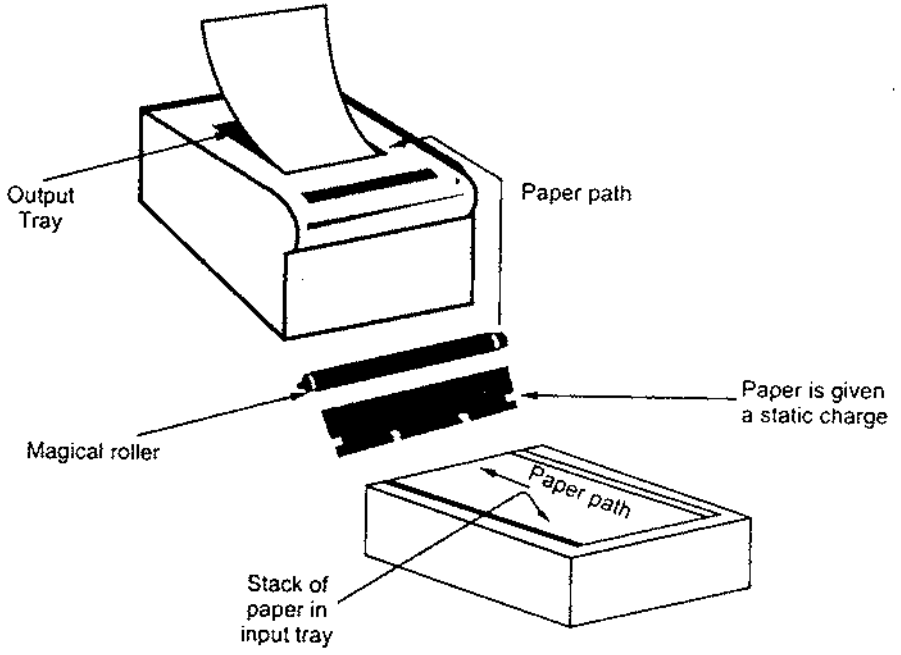


Fig. 1.38 Laser-printer

A colour laser printer works like a single colour laser printer, except that the process is repeated four times with four different ink colours : Cyan, magenta, yellow and black. Laser printers have high resolution from 600 dots per inch upto 1200 dots per inch. These printers print 4 to 16 page of text per minute. The high quality and speed of laser printers make them ideal for office environment.

Advantages of Laser printer

The main advantages of laser printers are speed, precision and economy. A laser can move very quickly, so it can "write" with much greater speed than an ink-jet. Because the laser beam has an unvarying diameter, it can draw more precisely, without spilling any excess ink. Laser printers tend to be more expensive than ink-jet printers, but it doesn't cost as much to keep them running. Its toner power is cheap and lasts for longer time.

Thermal Transfer Printer

In thermal transfer printer, wax paper and plain paper are drawn together over the strip of heating nibs. The heating nibs are selectively heated to cause the pigment transfer. In case of colour thermal transfer printers, the wax paper is placed on a roll of alternating, cyan, magenta, yellow and black strips, each of a length equal to the paper size. It is possible to create one colour hardcopy with less than 1 minute. This is possible because the material used to manufacture nib heats and cools very rapidly. Morden thermal transfer printers

accept a video signal and digital bitmap input, making them convenient for creating hardcopy of video images.

1.7.2.3 Plotters

Pen plotter

Pen plotter is an example of a hard-copy output device that does not use the raster-scan approach. The plotter uses random scan approach, in which a pen is steered over a piece of paper according to motion instruction issued by the computer. Then pen can be lowered on to the paper, so that it leaves a trace, or it can be raised in order to reposition the pen without drawing.

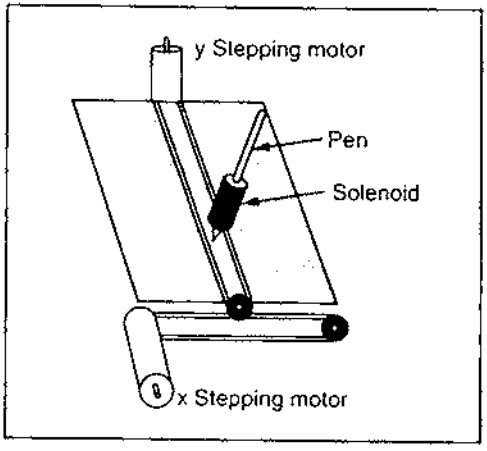


Fig. 1.39

Generally, plotters use two motors to move a pen in x and y directions. Some plotters move the paper in one direction and pen in an orthogonal direction. The colour plotters are constructed using multiple pens or alternatively a single pen holder and a mechanical pen loader that can select one of several pens from a stable of pens.

For drawing the desired shape, some plotters move the pen in a series of small, incremental motions in one of the eight directions, while others use a servomechanism to move the pen in a smooth path. Many plotters have controllers built into them that perform computations necessary to approximate geometrical shapes.

Flat bed plotter

In flat bed plotter pen moves in x and y direction on a sheet of paper spread out on the table and held down by electrostatic charge, by vaccum, or simply by being stretched tightly, as shown in the Fig. 1.40.

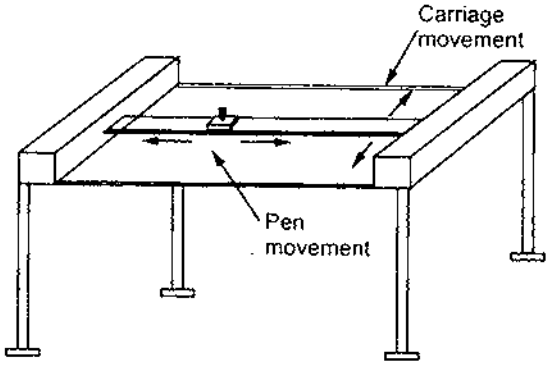


Fig. 1.40

As shown in the Fig. 1.40, carriage moves longitudinally over the table. On the carriage pen is mounted and it is moved latitudinally along the carriage. The pen is lowered to draw lines and it is lifted up during only movement. Flat bed plotters are available in sizes from 12 by 18 inches to 6 by 10 feet and larger.

Drum Plotter

Drum plotters move the paper along one axis and pen along the other axis, as shown in the Fig. 1.41. Normally, the paper is stretched tightly across a drum and pins on the drum engage the prepunched holes in the paper to prevent slipping.

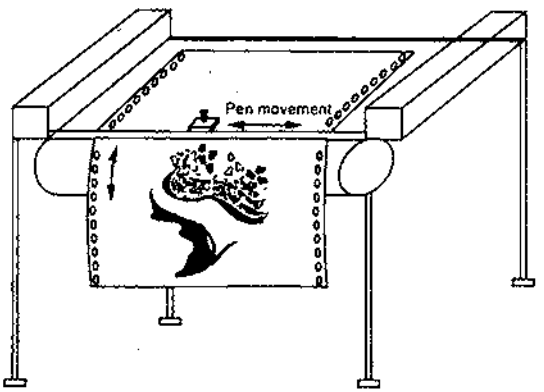


Fig. 1.41 Drum plotter

Drum plotters provide facility to move paper forward and backward, as per requirement.

Electrostatic Plotter

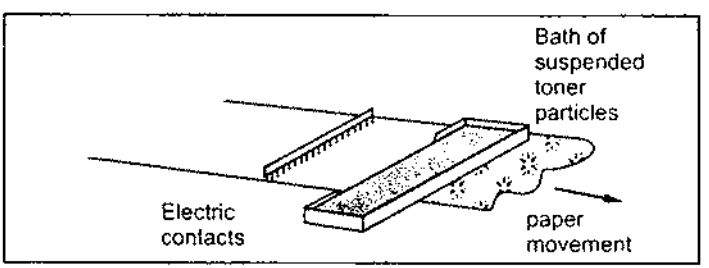


Fig. 1.42

Electrostatic plotter places a negative charge on those parts of a white paper that are to be black, then moves positively charged black toner across the paper, as shown in Fig. 1.42. The positively charged black toner attracts towards negative charge on paper and adhere there.

The electric contacts are used to deposite negative charge on the paper. Normally these contacts are constructed with a comb like structure and placed on the paper. Each contact is either on to impart a negative charge or off to impart no charge. Each dot on an electrostatic plot is either black or white; gray levels must be created with dither patterns. In dithering technique we can create an apparent increase in the number of available gray levels. This is achieved by incorporating multiple pixels positions to draw each intensity value. When we view a very small area from a sufficiently large viewing distance, our eyes average fine details with in the small area and record only the overall intensity of the area.

Important Features of Electrostatic Plotters

- Electrostatic plotters are faster than pen plotters and very high quality printers.
- Recent electrostatic plotters include a scan conversion capability.
- Colour electrostatic plotters are available. They make multiple passes over the paper to plot colour pictures or use multiple heads to deposit all the colours in a single pass.

1.7.2.4 Cameras

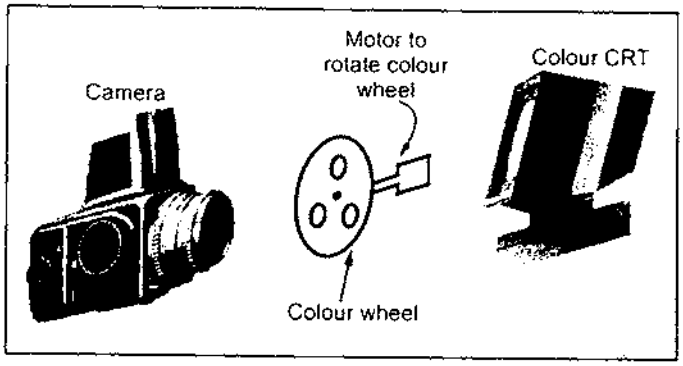


Fig. 1.43 Colour photograph recording using colour filters

Camera is also considered as a hardcopy device. It photographs an image displayed on a television or CPU monitor. It records the image on the colour film using film recorder. The film recorder captures the image in the form of a raster video signal, a bitmap or vector-style instructions. The resolution of image is depend on the screen resolution. The film recorder uses a raster scan technique to record the image displayed on the CRT. The film recorders use colour filters to record colour images, as shown in the Fig. 1.43.

Once the image is recorded in the film recorder, it is exposed to light of a specific colour to get it printed on a paper. The recently developed Cycolour technique use the paper, which is embeded with millions of microcapsules filled with one of the three coloured dyes-cyan, magenta, or yellow. These capsules harden selectively when exposed to light of a specific colour. For example, when exposed to red light, chemicals in the cyan-filled capsules cause that capsule to harden. With hardened cyan filled capsules when this paper passed through pressure rollers and pressed against a sheet of paper only unhardened capsules (magenta and yellow) break. The breaking of unhardened capsules (magenta and yellow) cause mixing of colours between them and the mixed colour (red) is transferred to the plain paper.

1.7.2.5 Comparison of Various Monochrome Hardcopy Devices

Parameters	Dot Matrix Printer	Ink Jet Printer	Thermal Printer	Laser Printer	Pen Plotter	Electro-static Plotter	Photo
Intensity levels per dot	2	2	2	2	2	2	Many
Addressability (points per inch)	upto 250	upto 360	upto 200	upto 1500	Atleast 1000	upto 400	upto 800
Dot size (thousandths of inch)	10-18	8-16	7-10	5	6-15	8	6-18

Relative purchase cost	Very low	Low	Low-Medium	High	Medium	Medium-High	Medium
Relative printing cost	Very low	Medium	Medium-High	Medium	Low	Medium-High	High
Image quality	Poor	Better	Better	Best	Good	Better	Best
Speed	Low	Medium	Medium	High	Very low	High	Very low

1.7.2.6 Comparison of Various Colour Hardcopy Devices

Parameters	Dot Matrix Printer	Ink Jet Printer	Thermal Printer	Laser Printer	Pen Plotter	Electro-static Plotter	Photo
Intensity levels per dot	8	8-many	8-many	8	upto16	8	Many
Addressability (points per inch)	upto 250	upto 360	upto 200	upto 1500	Atleast 1000	upto 400	upto 800
Dot size (thousandths of inch)	10-18	8-16	7-10	5	6-15	8	6-18
Relative purchase cost	Very low	Low	Medium	High	Medium	Medium-High	Medium-High
Relative printing cost	Very low	Medium	High	Medium	Low	High	High
Image quality	Poor	Better	Better	Best	Good	Better	Best
Speed	Low	Medium	Medium	High	Very low	Medium-High	Very low

1.8 Coordinate Systems

Most of the graphics packages use cartesian coordinate systems. However, in some applications, non-cartesion coordinate systems such as spherical, cylindrical, or other symmetries are useful. In this section, we first see standard cartesian coordinate systems and then we consider a commonly used non cartesian system, polar coordinate system.

1.8.1 Two Dimensional Cartesian Reference System

There are two possible orientations for a cartesian screen reference system, as shown in the Fig. 1.44. The Fig. 1.44 (a) shows the standard coordinate orientation with the coordinate origin at the lower-left corner of the screen. In some systems, particularly in personal computers, coordinate orientation is as shown in Fig. 1.44 (b). In this system, the coordinate origin is at the upper left corner.

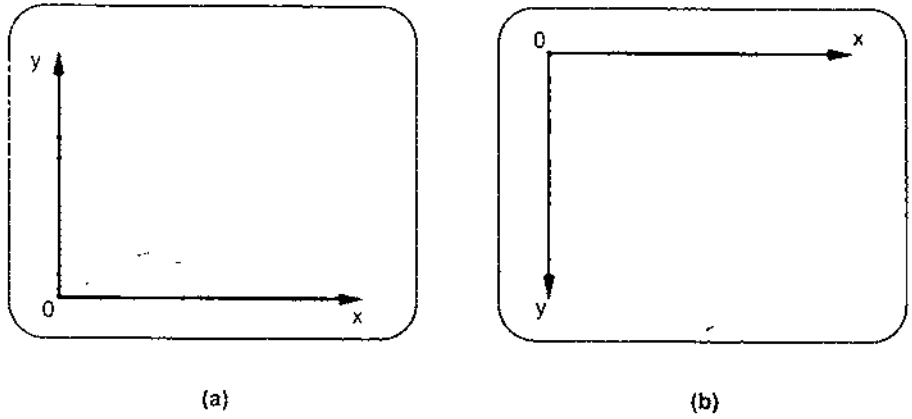


Fig. 1.44 Screen cartesian reference system

1.8.2 Three Dimensional Cartesian Reference System

There are two types of three dimensional reference system according to the orientation for the coordinate axes : **Right handed system** and **left handed system**. The right handed system uses the right hand thumb to point the positive z direction when we imagine the fingers curling from positive x axis to the positive y axis (through 90°) grasping the z axis, as shown in the Fig. 1.45.

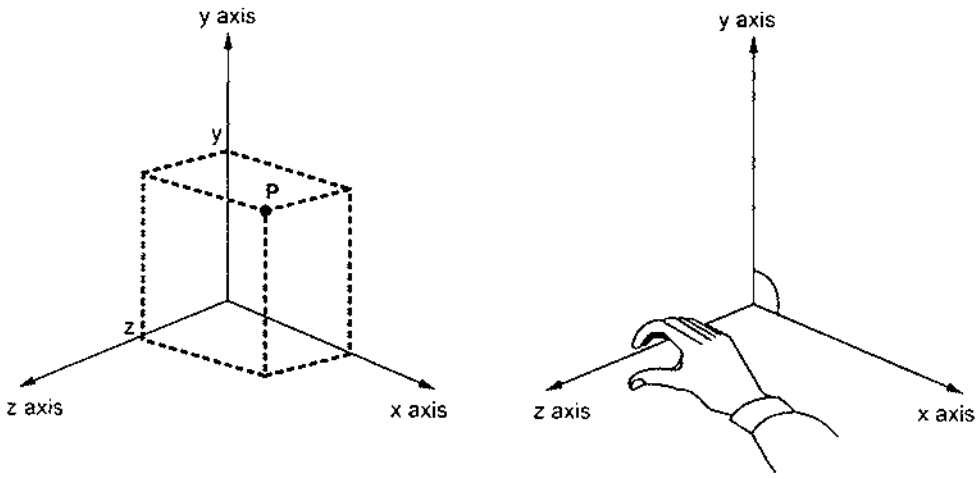


Fig. 1.45

In left handed cartesian coordinate system, the left hand thumb is used to point the positive z direction when we imagine the fingers of the left hand curl from the positive x axis to the positive y axis (through 90°) to grasp the z axis, as shown in the Fig. 1.46.

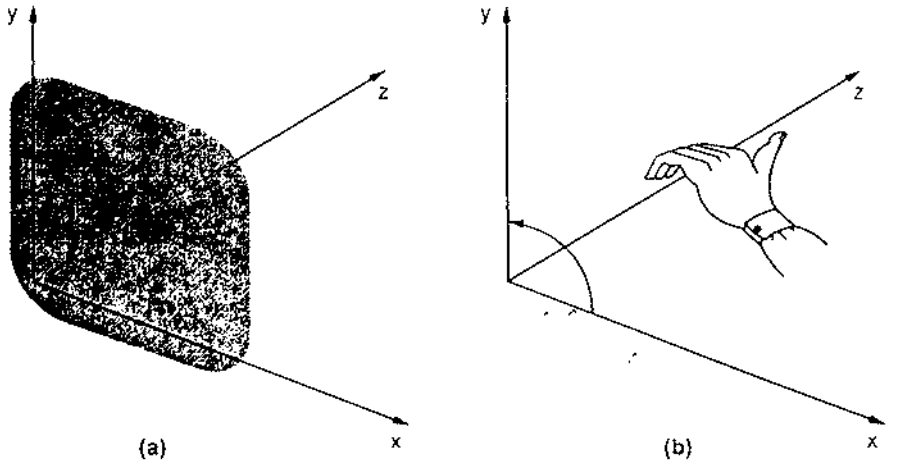


Fig. 1.46

1.8.3 Polar Coordinate System

It is a most commonly used non-cartesian coordinate system. In polar coordinate system a position is specified with a radial distance r from the coordinate origin, and an angular displacement θ from the horizontal, as shown in the Fig. 1.47.

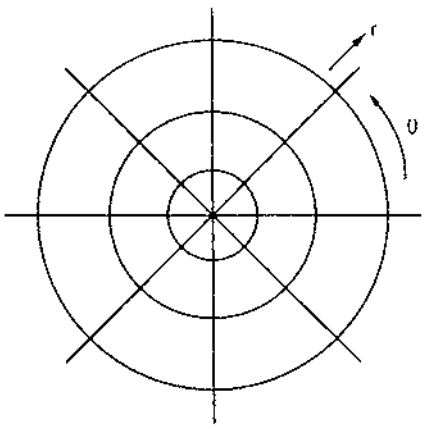


Fig. 1.47

Counter clockwise displacements are considered as positive angular displacements and clockwise displacements are consider as negative angular displacements. The angle θ is measured in degrees.

The Fig. 1.48 shows the relation between polar and cartesian coordinates. It is as follows :

$$x = r\cos\theta, \quad y = r\sin\theta$$

The inverse transformation from cartesian to polar coordinates is

$$r = \sqrt{x^2 + y^2},\ \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

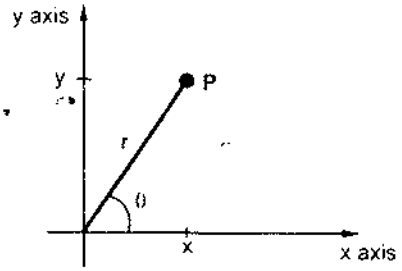


Fig. 1.48 Relation between polar and cartesian coordinates

1.9 Coordinate Representations

In general, graphics packages are designed to be used with cartesian coordinate specifications. If coordinate values for a picture are specified in some other reference frame, they must be converted to cartesian co-ordinates before they can be input to the graphics packages. Furthermore, we can construct the shape of individual objects, such as trees, buildings or furniture, in a scene within a separate coordinate reference frames called **modeling coordinates**, or **local coordinates** or **master coordinates** require coordinate conversion.

The objects represented in the modeling coordinates are first placed into appropriate positions within the scene using a reference frame called **world coordinates**. Then the world coordinate description of the scene is transferred to one or more output device reference frames for display. These display coordinates are referred to as **device coordinates** or **screen coordinates**, in case of video monitor. Generally, in a graphic system the world coordinate positions are first converted into **normalized device coordinates**, in the range from 0 to 1, before final conversion to specific device coordinates. (Refer section 5.2.1 for detail information on normalized coordinates). This conversion makes the system independent of the various devices that might be used at a particular workstation. The Fig. 1.49 illustrates the sequence of coordinate transformations from modeling coordinates to device coordinates for a two-dimensional application.

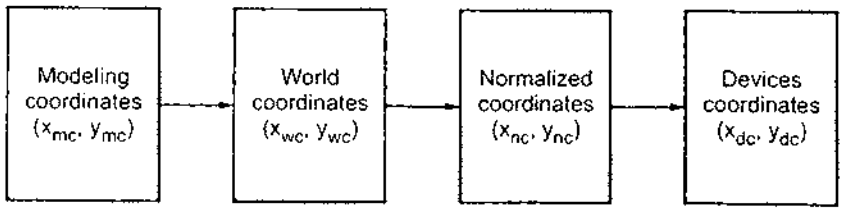


Fig. 1.49

Solved Examples

Ex. 1.1: *A video monitor has a display area measuring 12 inch by 9.6 inch. If the resolution is 1280 by 1024 and the aspect ratio is 1. What is the diameter of each screen point ?*

(Dec.-2001)

Sol. : An aspect ratio of 1 means that a vertical line plotted and horizontal line plotted with equal number of points have the same length. Therefore the diameter of each screen point can be given as

$$d = \frac{\text{horizontal display length}}{\text{horizontal resolution}}$$

$$= \frac{\text{vertical display length}}{\text{vertical resolution}}$$

$$= \frac{12}{1280} = \frac{9.6}{1024} = 9.375 \times 10^{-3} \text{ inch}$$

Ex. 1.2 : *How long it will take to load a 640 by 480 frame buffer with 12 bits per pixel if 10^5 bits can be transferred per second ? How long it will take to load a 24-bits per pixel frame buffer with a resolution of 1280 by 1024 using the same transfer rate ?* **(Dec-2001)**

Sol. : i) Total number of bits required to load the frame buffer with a resolution of 640 × 480 and with 12-bits per pixel can be given as

$$B = 640 \times 480 \times 12 = 3.6864 \times 10^6$$

Transfer rate is 10^5 bits/sec,

Therefore, time required to load the frame buffer is

$$T = \frac{3.6864 \times 10^6}{10^5} = 36.864 \text{ seconds}$$

ii) Total number of bits required to load the frame buffer with a resolution of 1280 × 1024 and with 24-bits per pixel can be given as

$$B = 1280 \times 1024 \times 24 = 31.45728 \times 10^6$$

Transfer rate is 10^5 bits/sec,

Therefore, time required to load the frame buffer is

$$T = \frac{31.45728 \times 10^6}{10^5}$$

$$= 314.5728 \text{ seconds}$$

Ex. 1.3 : *What is the fraction of the total refresh time per frame spent in retrace of the electron beam for a non-interlaced raster system with a resolution of 1280 by 1024, a refresh rate of 60 Hz, a horizontal retrace time of 5μsec and a vertical retrace time of 500 microseconds (μ sec) ?* **(Dec-99)**

Sol. : Total horizontal retrace time = $1024 \times 5 \times 10^{-6}$

Total vertical retrace time = 500×10^{-6}

∴ Total retrace time = $1024 \times 5 \times 10^{-6} + 500 \times 10^{-6}$

∴ The fraction of the total refresh time per frame spent in retrace of the electron beam can be given as

$$T = \frac{\text{Total retrace time}}{\text{refresh time}} = \frac{1024 \times 5 \times 10^{-6} + 500 \times 10^{-6}}{1 / 60}$$

$$= 0.3372$$

Ex. 1.4 : *For an electrostatic plotter 18-inch-wide paper, a resolution of 200 units to the inch in each direction and a paper speed of 3 inches per second, how many bits per second must be provided to allow the paper to move at full speed ?* **(Dec-99)**

Sol. : Total dots in the horizontal direction = 18 × 200 = 3600

Total dots in the 3 inch length of paper in one column = 3 × 200 = 600

∴ Total number of dot information required per second = 3600 × 600

$$= 2.16 \times 10^6$$

If we assume 8-bits are required for each dot, then total number of bits per second required to plot are given as

$$B = 2.16 \times 10^6 \times 8$$

$$= 17.28 \times 10^6$$

Review Questions

1. Discuss on topic image processing as picture analysis.
2. List the advantages of interactive graphics.
3. Explain the representative uses of computer graphics.
4. Explain the classification of use of computer graphics.
5. What do you mean by rasterization ?
6. Define scan conversion.
7. Write a short note on
   a) Keyboard b) Mouse
   c) Trackball and spaceball d) Joystick
   e) Digitizer f) Light pen
   g) Touch panels h) Scanner
8. Write a short note on
   a) Cathode -Ray Tubes
   b) Vector scan display
   c) Raster scan display
   d) Beam penetration technique
   e) Shadow mask technique
9. Explain the working of direct-view storage tubes.
10. List the advantages and disadvantages of DVST.

11. Write a short note on
    a) Flat panel display
    b) Plasma panel display
12. List the important characteristics of video display devices .
13. Explain the important characteristics of hard copy devices.
14. Define dot size, addressability, interdot distance and resolution.
15. Give the classification of printers.
16. Explain various types of printers.
17. Give the difference between line printer and dot matric printer.
18. List the features of ink-jet printer.
19. Write a short note on laser printer.
20. Explain the principle of thermal transfer printer.
21. Write a short note on pen plotter
22. Write a short note on electrostatic plotter.
23. Write a short note on cameras.
24. Give the comparison between various monochrome and colour hardcopy devices.
25. Explain display file and its structure.
26. What is the role of display file interpreter ?
27. Explain the function of display processor in raster scan and vector scan displays.
28. What is a frame buffer? Explain the organization of frame buffer.
29. Write a short note on coordinate systems.
30. Explain the two dimensional cartesian reference system.
31. Explain the three dimensional cartesian reference system.
32. Explain the polar coordinate system.
33. Write a note an coordinate representations.

University Questions

1. Write detailed note on bit planes and frame buffer organisations. (Dec-96, Dec-97, May-2000, Dec-2000)
2. Write detailed note on DVST display devices (May-97, Dec-2000)
3. Write detailed note on bit plane organisation. (May-97, May-2001)
4. Write detail note on storage type CRTs and refresh type CRTs. (May-98)
5. Compare storage against refresh type CRT display. List out the important properties of phosphor being used in CRTs. (Dec-98)
6. Write short note on plasma panel display (Dec-98)
7. Write short note on Graphical output devices. (May-99)
8. What is refresh buffer? Identify the contents and organization of the refresh buffer for the case of raster display and vector display. (Dec-99)

9. Compare and contrast the operating characteristics of raster refresh systems, plasma panels and LCDs. **(Dec-99)**
10. Identify the appropriate applications for each of the display technologies cited in past (as above) **(Dec-99)**
11. What is the fraction of the total refresh time per frame spent in retrace of the electron beam for a non-interlaced raster system with a resolution of 1280 by 1024, a refresh rate of 60 Hz, a horizontal retrace time of 5μsec and a vertical retrace time of 500 microseconds (μ sec) ? **(Dec-99)**

    Ans. : [$(1024 \times 5 \times 10^{-6} + 500 \times 10^{-6}) / (1/60) = 0.3372$]
12. For an electrostatic plotter 18-inch-wide paper, a resolution of 200 units to the inch in each direction and a paper speed of 3 inches per second, how many bits per second must be provided to allow the paper to move at full speed ? **(Dec-99)**

    Ans. : [$18 \times 200 \times 3 \times 8$ (bits per pixel) = 86400]
13. The light pen is an aging technology with a limited use. Justify this contention. **(Dec-99)**
14. Compare refresh type and storage type CRT display. **(May-2000)**
15. Explain the block diagram of raster display system with display processor . Also explain how the monitor functions in raster display. **(May-2001)**
16. Write a short note on bit plane and frame buffer organization. **(Dec-2001)**
17. Compare and contrast the operating characteristics of different display technologies. Identify the appropriate applications for each. **(May-2002)**
18. Write a short note on random scan displays. **(May-2002)**
19. Write a short note on computer graphics applications **(May-2002)**
20. Write a short note on display processors. **(May-2002, May-2003)**
21. Illustrate plasma panel display. Give its advantages and disadvantages. **(May-2003)**

❑❑❑

2 Raster Graphics Algorithms for Drawing 2-D Primitives

2.1 Introduction

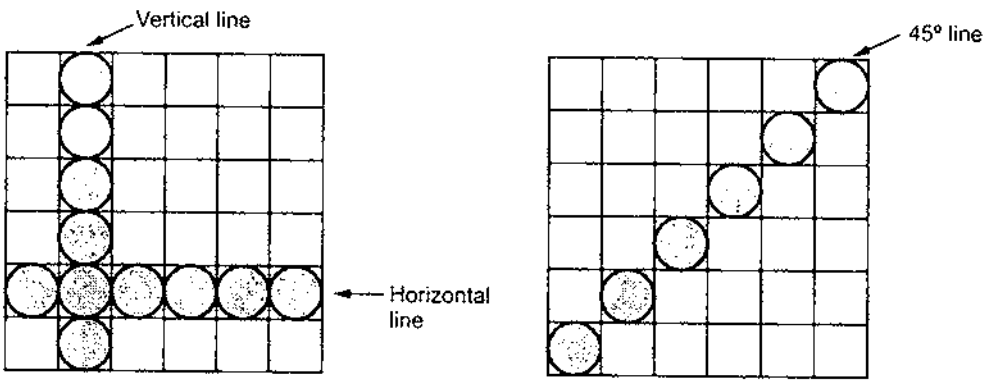
In the previous chapter we have seen that the raster scan graphics devices require special procedures for displaying graphics objects such as line, circle, polygons, curves and even characters. In this chapter we will examine the procedures for line, circle and character generation.

2.2 Basic Concepts in Line Drawing

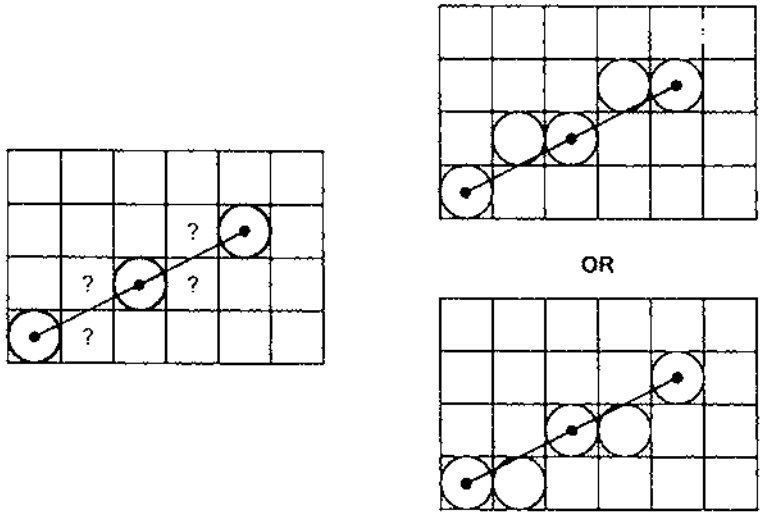
Before discussing specific line drawing algorithms it is useful to note the general requirements for such algorithms. These requirements specify the desired characteristics of line.

- The line should appear as a straight line and it should start and end accurately.
- The line should be displayed with constant brightness along its length independent of its length and orientation.
- The line should be drawn rapidly.

Let us see the different lines drawn in Fig. 2.1.



(a) Vertical and Horizontal lines (b) 45° line



(c) Line with other orientation

Fig. 2.1

As shown in Fig. 2.1 (a), horizontal and vertical lines are straight and have same width. The 45° line is straight but its width is not constant. On the other hand, the line with any other orientation is neither straight nor has same width. Such cases are due to the finite resolution of display and we have to accept approximate pixels in such situations, shown in Fig. 2.1 (c).

The brightness of the line is dependent on the orientation of the line. We can observe that the effective spacing between pixels for the 45° line is greater than for the vertical and horizontal lines. This will make the vertical and horizontal lines appear brighter than the 45° line. Complex calculations are required to provide equal brightness along lines of varying length and orientation. Therefore, to draw line rapidly some compromises are made such as

- Calculate only an approximate line length
- Reduce the calculations using simple integer arithmetic
- Implement the result in hardware or firmware

2.3 Line Drawing Algorithms

Considering the assumptions made in the previous section most line drawing algorithms use incremental methods. In these methods line starts with the starting point. Then a fix increment is added to the current point to get the next point on the line. This is continued till the end of line. Let us see the incremental algorithm.

Incremental Algorithm

1. CurrPosition = Start

   Step = Increment
2. if (| CurrPosition – End | < Accuracy) then go to step 5

```
{ This checks whether the current position is reached upto approximate
end point. If yes, line drawing is completed. }
```

if (CurrPosition < End) then go to step 3

```
[Here start < End]
```

if (CurrPosition > End) then go to step 4

```
[Here start > End]
```

3. CurrPosition = CurrPosition + Step

   go to step 2

4. CurrPosition = CurrPosition – Step

   go to step 2

5. Stop.

In the following sections we discuss the line rasterizing algorithms based on the incremental algorithm.

2.3.1 Digital Differential Analyzer

We know that the slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \qquad \ldots (2.1)$$

The above differential equation can be used to obtain a rasterized straight line. For any given x interval Δx along a line, we can compute the corresponding y interval Δy from equation 2.1 as

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x \qquad \ldots (2.2)$$

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

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y \qquad \ldots (2.3)$$

Once the intervals are known the values for next x and next y on the straight line can be obtained as follows

$$x_{i+1} = x_i + \Delta x = x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \qquad \ldots (2.4)$$

and

$$y_{i+1} = y_i + \Delta y = y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \qquad \ldots (2.5)$$

The equations 2.4 and 2.5 represent a recursion relation for successive values of x and y along the required line. Such a way of rasterizing a line is called a **digital differential analyzer** (DDA). For simple DDA either Δx or Δy, whichever is larger, is chosen as one raster unit, i.e.

if $|\Delta x| \geq |\Delta y|$ then

else $\Delta x = 1$

$\Delta y = 1$

With this simplification, if $\Delta x = 1$ then

we have $y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1}$ and

$x_{i+1} = x_i + 1$

If $\Delta y = 1$ then

we have

$y_{i+1} = y_i + 1$ and

$x_{i+1} = x_i + \frac{x_2 - x_1}{y_2 - y_1}$

Let us see the digital differential analyzer (DDA) routine for rasterizing a line

DDA Line Algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
   [if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. if $(\Delta x \geq \Delta y)$ then
   length = Δx
   else
   length = Δy
   end if
4. $\Delta x = (x_2 - x_1) / \text{length}$
   $\Delta y = (y_2 - y_1) / \text{length}$
   [This makes either Δx or Δy equal to 1 because length is either $|x_2 - x_1|$ or $|y_2 - y_1|$. Therefore, the incremental value for either x or y is one.]
5. $x = x_1 + 0.5 * \text{Sign}(\Delta x)$
   $y = y_1 + 0.5 * \text{Sign}(\Delta y)$
   [Here, Sign function makes the algorithm work in all quadrant. It returns - 1, 0, 1 depending on whether its argument is < 0, = 0, > 0 respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.]
6. i = 1 [Begins the loop, in this loop points are plotted]
   While (i ≤ length)
   {
   Plot (Integer (x), Integer (y))
   $x = x + \Delta x$
   $y = y + \Delta y$
   i = i + 1

7. Stop

Let us see few examples to illustrate this algorithm.

Ex. 2.1 : *Consider the line from (0, 0) to (4, 6). Use the simple DDA algorithm to rasterize this line*

Sol. : Evaluating steps 1 to 5 in the DDA algorithm we have

$$x_1 = 0 \qquad y_1 = 0$$

$$x_2 = 4 \qquad y_2 = 6$$

$$\therefore \quad \text{Length} = |y_2 - y_1| = 6$$

$$\therefore \quad \Delta x = |x_2 - x_1| / \text{length} = \frac{4}{6}$$

and

$$\Delta y = |y_2 - y_1| / \text{length} = 6 / 6 = 1$$

Initial value for

$$x = 0 + 0.5 * \text{Sign}\left(\frac{4}{6}\right) = 0.5$$

$$y = 0 + 0.5 * \text{Sign}(1) = 0.5$$

Tabulating the results of each iteration in the step 6 we get,

i	Plot	x	y
		0.5	0.5
1	(0, 0)		
		1.167	1.5
2	(1, 1)		
		1.833	2.5
3	(1, 2)		
		2.5	3.5
4	(2, 3)		
		3.167	4.5
5	(3, 4)		
		3.833	5.5
6	(3, 5)		
		4.5	6.5

Table 2.1

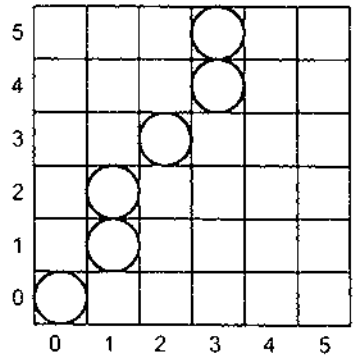


Fig. 2.2 Result for a simple DDA

The results are plotted as shown in the Fig. 2.2. It shows that the rasterized line lies to both sides of the actual line, i.e. the algorithm is **orientation dependent**.

Ex. 2.2 : *Consider the line from (0, 0) to (– 6, – 6). Use the simple DDA algorithm to rasterize this line.*

Sol. : Evaluating steps 1 to 5 in the DDA algorithm we have

$$x_1 = 0 \qquad y_1 = 0$$

$$x_2 = -6 \qquad y_2 = -6$$

$$\therefore \text{ Length} = |x_2 - x_1| = |y_2 - y_1| = 6$$

$$\therefore \qquad \Delta x = \Delta y = -1$$

Initial values for

$$x = 0 + 0.5 * \text{Sign}(-1) = -0.5$$

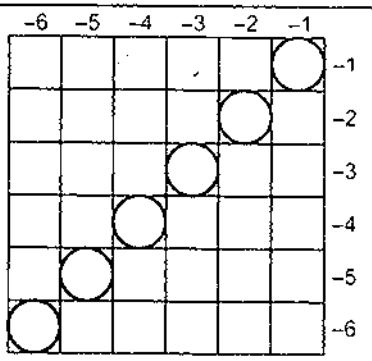
$$y = 0 + 0.5 * \text{Sign}(-1) = -0.5$$

Tabulating the results of each iteration in the step 6 we get,

i	Plot	x	y
		– 0.5	– 0.5
1	(– 1, – 1)		
		– 1.5	– 1.5
2	(– 2, – 2)		
		– 2.5	– 2.5
3	(– 3, – 3)		
		– 3.5	– 3.5
4	(– 4, – 4)		

		– 4.5	– 4.5
5	(– 5, – 5)		
		– 5.5	– 5.5
6	(– 6, – 6)		
		– 6.5	– 6.5

Table 2.2



★ –ve pixel values are with reference to pixel at the center of screen

Fig. 2.3 Result for a simple DDA

The results are plotted as shown in the Fig. 2.3. It shows that the rasterized line lies on the actual line and it is 45° line.

'C' code for DDA Line Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float x,y,x1,y1,x2,y2,dx,dy,length;
int i,gd,gm;
clrscr();

/* Read two end points of line
--------------------------------- */
printf("Enter the value of x1 :\t");
scanf("%f",&x1);
```

```
printf("Enter the value of y1 :\t");
scanf("%f",&y1);
printf("Enter the value of x2 :\t");
scanf("%f",&x2);
printf("Enter the value of y2 :\t");
scanf("%f",&y2);

/* Initialise graphics mode
---------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

dx=abs(x2-x1);
dy=abs(y2-y1);

if (dx >= dy)
      {
      length = dx;
      }
      else
      {
      length = dy;
      }
dx = (x2-x1)/length;
dy = (y2-y1)/length;

x = x1 + 0.5;  /* Factor 0.5 is added to round the values */
y = y1 + 0.5;  /* Factor 0.5 is added to round the values */

i = 1;         /* Initialise loop counter */
while(i <= length)
   {
    putpixel(x,y,15);
    x = x + dx;
    y = y + dy;
    i = i + 1;
    delay(100); /* Delay is purposely inserted to see
                   observe the line drawing process */
```

```
    }
getch();
closegraph();
}
```

Advantages of DDA Algorithm

1. It is the simplest algorithm and it does not require special skills for implementation.
2. It is a faster method for calculating pixel positions than the direct use of equation y = mx + b. It eliminates the multiplication in the equation by making use of raster characteristics, so that appropriate increments are applied in the x or y direction to find the pixel positions along the line path.

Disadvantages of DDA Algorithm

1. Floating point arithmetic in DDA algorithm is still time-consuming.
2. The algorithm is orientation dependent. Hence end point accuracy is poor.

2.3.2 Bresenham's Line Algorithm

Bresenham's line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the computer can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2. Therefore, it is an efficient method for scan-converting straight lines.

The basic principle of Bresenham's line algorithm is to select the optimum raster locations to represent a straight line. To accomplish this the algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called **decision variable** or the **error**. This is illustrated in the Fig. 2.4.

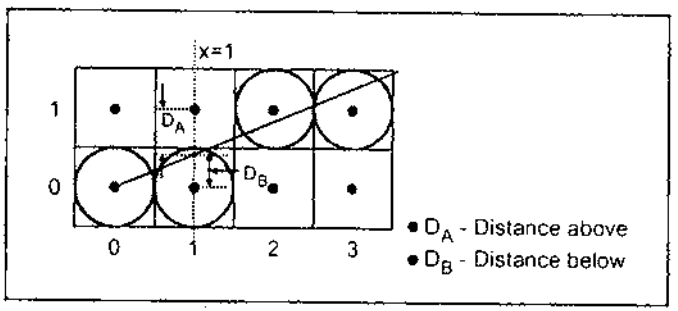


Fig. 2.4

As shown in the Fig. 2.4, the line does not pass through all raster points (pixels). It passes through raster point (0, 0) and subsequently crosses three pixels. It is seen that the intercept of line with the line x = 1 is closer to the line y = 0, i.e. pixel (1, 0) than to the line y = 1 i.e. pixel (1, 1). Hence, in this case, the raster point at (1, 0) better represents the path of the line than that at (1, 1). The intercept of the line with the line x = 2 is close to the line y = 1, i.e. pixel (2, 1) than to the line y = 0, i.e. pixel (2, 0). Hence, the raster point at (2, 1) better represents the path of the line, as shown in the Fig. 2.4

In mathematical terms error or decision variable is defined as

$$e = D_B - D_A \quad \text{or} \quad D_A - D_B$$

Let us define $e = D_B - D_A$. Now if $e > 0$, then it implies that $D_B > D_A$, i.e., the pixel above the line is closer to the true line. If $D_B < D_A$ (i.e. $e < 0$) then we can say that the pixel below the line is closer to the true line. Thus by checking only the sign of error term it is possible to determine the better pixel to represent the line path.

The error term is initially set as

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

$$\text{where} \quad \Delta y = y_2 - y_1, \text{ and } \Delta x = x_2 - x_1$$

Then according to value of e following actions are taken.

```
while ( e ≥ 0)
    {
        y = y + 1
        e = e - 2 * Δx
    }
x = x + 1
e = e + 2 * Δy
```

When $e \geq 0$, error is initialized with $e = e - 2\Delta x$. This is continued till error is negative. In each iteration y is incremented by 1. When $e < 0$, error is initialized to $e = e + 2\Delta y$. In both the cases x is incremented by 1. Let us see the Bresenham's line drawing algorithm.

Bresenham's Line Algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
   [if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. {Initialize starting point]
   $x = x_1$
   $y = y_1$
4. $e = 2 * \Delta y - \Delta x$
   [Initialize value of decision variable or error to compensate for nonzero intercepts]
5. i = 1 [Initialize counter]
6. Plot (x, y)
7. while ($e \geq 0$)
   {
   $y = y + 1$
   $e = e - 2 * \Delta x$
   }
   $x = x + 1$
   $e = e + 2 * \Delta y$
8. $i = i + 1$
9. if ($i \leq \Delta x$) then go to step 6.
10. Stop

Ex. 2.3 : *Consider the line from (5, 5) to (13, 9). Use the Bresenham's algorithm to rasterize the line.*

Sol. : Evaluating steps 1 through 4 in the Bresenham's algorithm we have,

$$\Delta x = |13 - 5| = 8$$
$$\Delta y = |9 - 5| = 4$$
$$x = 5$$
$$y = 5$$
$$e = 2 * \Delta y - \Delta x = 2 * 4 - 8$$
$$= 0$$

Tabulating the results of each iteration in the step 5 through 10.

i	Plot	x	y	e
		5	5	0
1	(5, 5)	6	6	– 8
2	(6, 6)	7	6	0
3	(7, 6)	8	7	– 8
4	(8, 7)	9	7	0
5	(9, 7)	10	8	– 8
6	(10, 8)	11	8	0
7	(11, 8)	12	9	– 8
8	(12, 9)	13	9	0
9	(13, 9)	14	10	– 8

Table 2.3

The results are plotted as shown in Fig. 2.5.

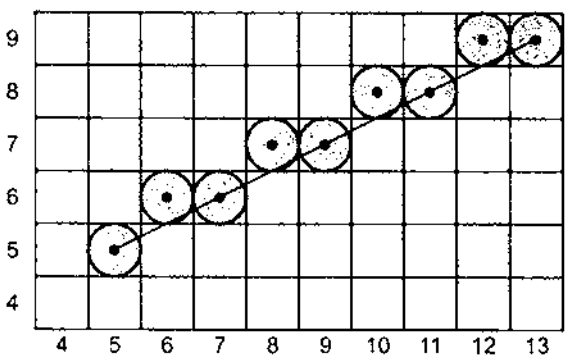


Fig. 2.5

'C' code for Bresenham's Line Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float x,y,x1,y1,x2,y2,dx,dy,e;
int i,gd,gm;
clrscr();

/* Read two end points of line
--------------------------------- */
printf("Enter the value of x1 :\t");
scanf("%f",&x1);
printf("Enter the value of y1 :\t");
scanf("%f",&y1);
printf("Enter the value of x2 :\t");
scanf("%f",&x2);
printf("Enter the value of y2 :\t");
scanf("%f",&y2);

/* Initialise graphics mode
--------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

dx=abs(x2-x1);
dy=abs(y2-y1);

/* Initialise starting point
----------------------------*/
x = x1;
y = y1;

/* Initialise decision variable
------------------------------- */
e = 2 * dy-dx;
```

2dx-dy

```
i = 1;     /* Initialise loop counter */

do
{
putpixel(x,y,15);
while(e >= 0)
    {
     y = y + 1;
     e = e - 2 * dx;
    }
x = x + 1;
e = e + 2 * dy;
i = i + 1;
}
while( i <= dx);
getch();
closegraph();
}
```

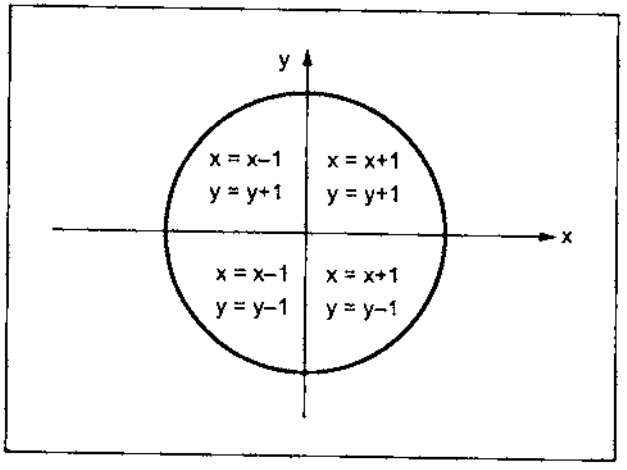


Fig. 2.6 Conditions for generalized Bresenham's algorithm

The Bresenham's algorithm only works for the first octant. The generalized Bresenham's algorithm requires modification for lines lying in the other octants. Such algorithm can be easily developed by considering the quadrant in which the line lies and it slope. When the absolute magnitude of the slope of the line is greater than 1, y is incremented by one and Bresenham's decision variable or error is used to determine when to increment x. The x and y incremental values depend on the quadrant in which the line exists. This is illustrated in Fig. 2.6.

Generalized Bresenham's Algorithm

1. Read the line end point (x_1, y_1) and (x_2, y_2) such that they are not equal.
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. Initialize starting point

   $x = x_1$

   $y = y_1$

4. $s_1 = \text{Sign}(x_2 - x_1)$

   $s_2 = \text{Sign}(y_2 - y_1)$

   [Sign function returns - 1, 0, 1 depending on whether its argument is <0, = 0, > 0 respectively]

5. if $\Delta y > \Delta x$ then

   Exchange Δx and Δy

   Ex_change = 1

   else

   Ex_change = 0

   end if

   [Interchange Δx and Δy depending on the slope of the line and set Ex_change flag accordingly]

6. $e = 2 * \Delta y - \Delta x$

   [Initialize value of decision variable or error to compensate for nonzero intercept].

7. i = 1 [Initialize counter]

8. Plot (x, y)

9. while ($e \geq 0$)

   { if (Ex_change = 1) then

   $x = x + s_1$

   else

   $y = y + s_2$

   end if

   $e = e - 2 * \Delta x$

   }

10. if Ex_change = 1 then

    $y = y + s_2$

    else

    $x = x + s_1$

    end if

    $e = e + 2 * \Delta y$

11. i = i + 1

12. if ($i \leq \Delta x$) then go to step 8

13. Stop

2.4 Antialiasing of Lines

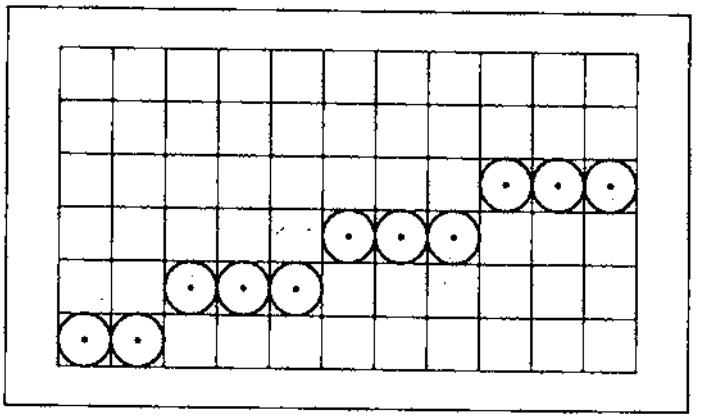


Fig. 2.7 Aliasing effect

In the line drawing algorithms, we have seen that all rasterized locations do not match with the true line and we have to select the optimum raster locations to represent a straight line. This problem is severe in low resolution screens. In such screens line appears like a stair-step, as shown in the Fig. 2.7. This effect is known as **aliasing**. It is dominant for lines having slopes less than 20° or greater than 70°.

The aliasing effect can be reduced by adjusting intensities of the pixels along the line. The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called **antialiasing**.

2.5 Methods of Antialiasing

2.5.1 Increasing Resolution

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.

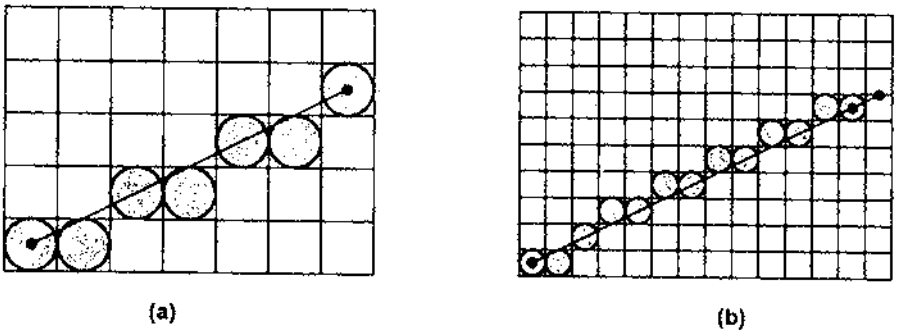


Fig. 2.8 Effect on aliasing with increase in resolution

As shown in the Fig. 2.8, line looks better in twice resolution, but this improvement comes at the price of quadrupling the cost of memory, bandwidth of memory and scan-conversion time. Thus increasing resolution is an expensive method for reducing aliasing effect.

2.5.2 Unweighted Area Sampling

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. In antialiasing, instead of picking closest pixel, both pixels are highlighted. However, their intensity values may differ.

In unweighted area sampling, the intensity of pixel is proportional to the amount of line area occupied by the pixel. This technique produces noticeably better results than does setting pixels either to full intensity or to zero intensity.

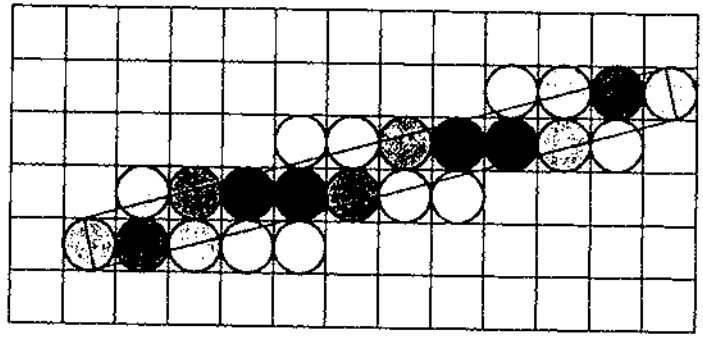


Fig. 2.9 Unweighted area sampling

2.5.3 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 amount 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 unequally 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 Fig. 2.10.

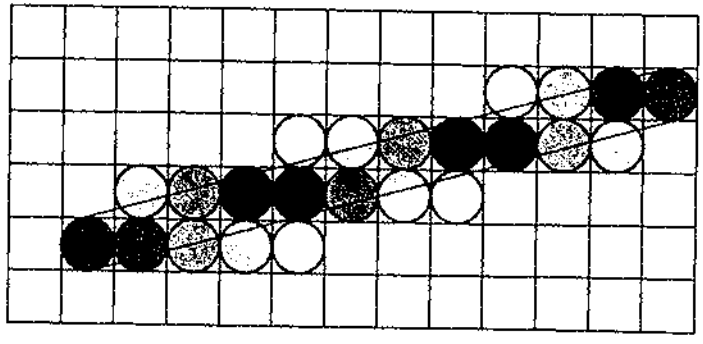


Fig. 2.10 Weighted area sampling

2.6 Thick Line Segments

So far we have discussed line drawing algorithms where thickness of line is one pixel. In raster displays, it is possible to draw lines with thickness greater than one pixel. To produce a thick line, we have to run two line drawing algorithms in parallel to find the pixels along the line edges, and while stepping along the line we have to turn on all the pixels which lie between the boundaries. This is illustrated in Fig. 2.11.

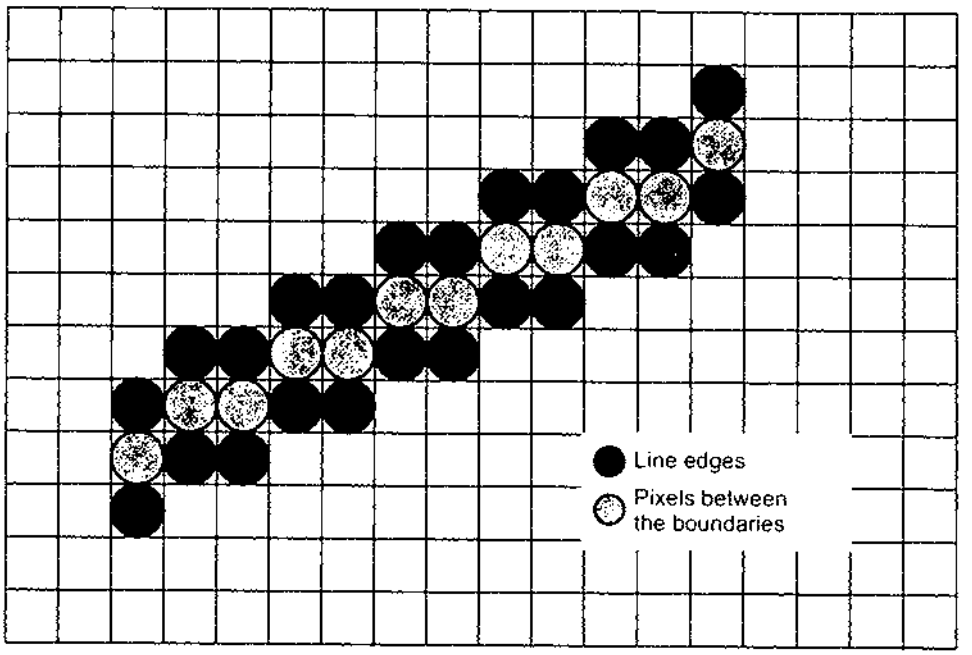


Fig. 2.11 Thick line

Let us consider line from point (x_1, y_1) to (x_2, y_2) having thickness w, then we have a top boundary between the points $(x_1, y_1 + w_y)$ and $(x_2, y_2 + w_y)$ and a lower boundary between $(x_1, y_1 - w_y)$ and $(x_2, y_2 - w_y)$ where w_y is given by

$$w_y = \frac{(w-1)}{2} \frac{\left[(x_2 - x_1)^2 + (y_2 - y_1)^2\right]^{1/2}}{|x_2 - x_1|}$$

Here, w_y is the amount by which the boundary lines are moved from the line center, as shown in the Fig. 2.12. The factor$(w - 1)$ in the above equation exist because the line boundary itself has a thickness of one pixel. We further divide the factor $(w - 1)$ by 2 because half the thickness will be used to offset the top boundary, and the other half to move the bottom boundary.

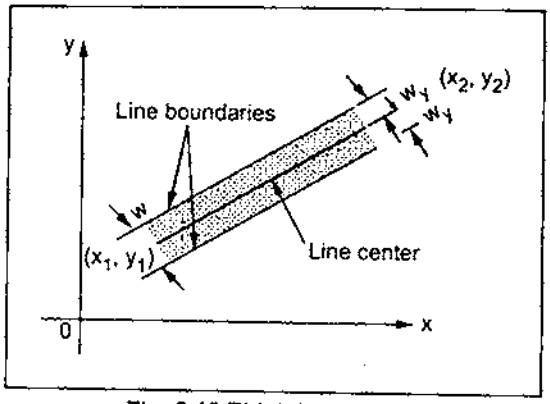


Fig. 2.12 Thick line details

We can use equation for w_y for lines having slope less than 1. For sharp slope lines, i.e. lines having slope greater than 1 lines are handled similarly with the x and y roles reversed. In this case w_x is given as

$$w_x = \frac{(w-1)}{2} \frac{\left[(x_2 - x_1)^2 + (y_2 - y_1)^2\right]^{1/2}}{|y_2 - y_1|}$$

Thus, left and right boundaries are $(x_1 - w_x, y_1)$ to $(x_2 - w_x, y_2)$ and $(x_1 + w_x, y_1)$ to $(x_2 + w_x, y_2)$, respectively.

'C' code for Thick Line Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
int gd = DETECT, gm  ;
float wy, wx, x1, y1, x2, y2;
int i, thickness;
initgraph(&gd, &gm, " ");

/* Read two end points of the line
------------------------------------ */
printf("Enter the co-ordinates for the line:\n");
printf("X1: ");
scanf("%f", &x1);
printf("Y1: ");
scanf("%f", &y1);
printf("X2: ");
scanf("%f", &x2);
printf("Y2: ");
scanf("%f", &y2);
```

```
/* Enter the thickness of the line
-----------------------------------*/
printf("Enter the required thickness: ");
scanf("%d", &thickness);
cleardevice();
line (x1, y1, x2, y2);
if ((y2 - y1) / (x2 - x1) < 1)
{
wy =(thickness-1)*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2*fabs(x2-x1));
 for(i = 0; i < wy; i++)
 {
 line(x1, y1 - i, x2, y2 - i);
 line(x1, y1 + i, x2, y2 + i);
 }
}
else
{
wx = (thickness-1)*sqrt(pow((x2-x1),2)+pow((y2-y1),2))/(2*fabs(y2-y1));
 for(i = 0; i < wx; i++)
 {
 line(x1 - i, y1, x2 - i, y2);
 line(x1 + i, y1, x2 + i, y2);
 }
}
printf("This is the line of thickness %d units.\n", thickness);
getch();
}
```

2.7 Basic Concepts in Circle Drawing

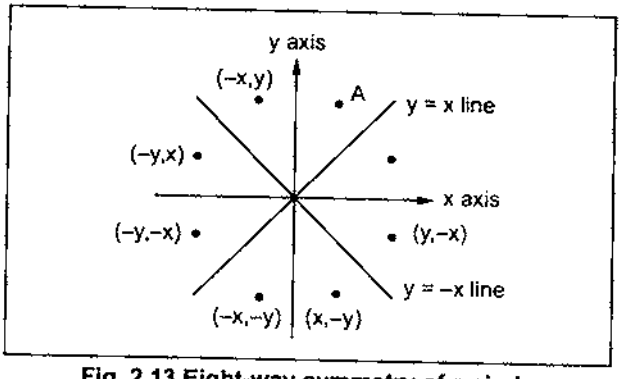


Fig. 2.13 Eight-way symmetry of a circle

A circle is a symmetrical figure. It has eight-way symmetry as shown in the Fig. 2.13. Thus, any circle generating algorithm can take advantage of the circle symmetry to plot eight points by calculating the coordinates of any one point. For example, if point A in the Fig. 2.13 is calculated with a circle algorithm, seven more points could be found just by reflection.

2.8 Representation of a Circle

There are two standard methods of mathematically representing a circle centered at the origin.

2.8.1 Polynomial Method

In this method circle is represented by a polynomial equation.

$$x^2 + y^2 = r^2$$

where x : the x coordinate
y : the y coordinate
r : radius of the circle

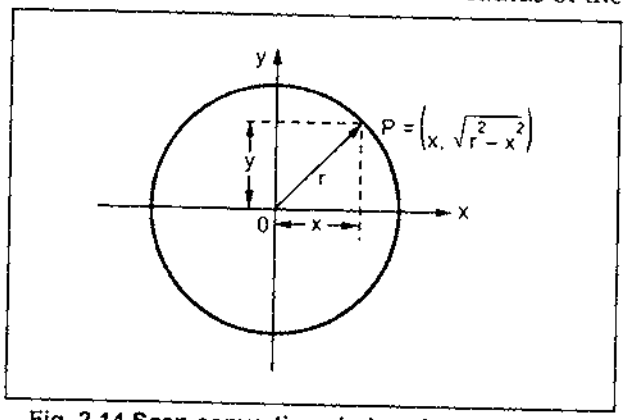


Fig. 2.14 Scan converting circle using polynomial method

Here, polynomial equation can be used to find y coordinate for the known x coordinate. Therefore, the scan converting circle using polynomial method is achieved by stepping x from 0 to $r\sqrt{2}$, and each y coordinate is found by evaluating $\sqrt{r^2 - x^2}$ for each step of x. This generates the 1/8 portion (90° to 45°) of the circle. Remaining part of the circle can be generated just by reflection.

The polynomial method of circle generation is an inefficient method. In this method for each point both x and r must be squared and x^2 must be subtracted from r^2; then the square root of the result must be found out.

2.8.2 Trigonometric Method

In this method, the circle is represented by use of trigonometric functions

$$x = r\cos\theta \quad \text{and} \quad y = r\sin\theta$$

where θ : current angle
r : radius of the circle
x : the x coordinate
y : the y coordinate

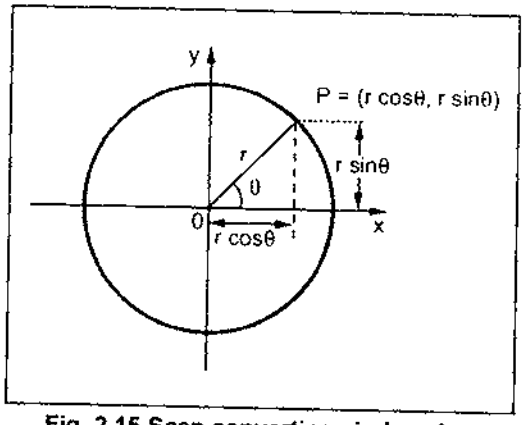


Fig. 2.15 Scan converting circle using trigonometric method

The scan converting circle using trigonometric method is achieved by stepping θ from 0 to $\pi/4$ radians, and each value of x and y is calculated. However, this method is more inefficient than the polynomial method because the computation of the values of sin θ and cos θ is even more time-consuming than the calculations required in the polynomial method.

2.9 Circle Drawing Algorithms

In this section we are going to discuss two efficient circle drawing algorithms :

- DDA algorithm and
- Bresenham's algorithm
- Midpoint algorithm

2.9.1 DDA Circle Drawing Algorithm

We know that, the equation of circle, with origin as the center of the circle is given as

$$x^2 + y^2 = r^2$$

The digital differential analyser algorithm can be used to draw the circle by defining circle as a differential equation. It is as given below

$$2\,x\,dx + 2\,y\,dy = 0 \quad \text{where } r \text{ is constant}$$

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

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

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

From above equation, we can construct the circle by using incremental x value, $\Delta x = \varepsilon y$ and incremental y value, $\Delta y = -\varepsilon x$, where ε is calculated from the radius of the circle as given below

$$2^{n-1} \le r < 2^n \quad r : \text{radius of the circle}$$

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

For example, if r = 50 then n = 6 so that $32 \le 50 < 64$

$$\therefore \quad \varepsilon = 2^{-6}$$

$$= 0.0156$$

Applying these incremental steps we have,

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

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

The points plotted using above equations give the spiral instead of the circle. To get the circle we have to make one correction in the equation; we have to replace x_n by x_{n+1} in the equation of y_{n+1}.

Therefore, now we have

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

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

Algorithm

1. Read the radius (r), of the circle and calculate value of ε
2. start_x = 0

   start_y = r
3. x_1 = start_x

   y_1 = start_y

4. do

   { $x_2 = x_1 + \varepsilon y_1$

   $y_2 = y_1 - \varepsilon x_2$

   [x_2 represents x_{n+1} and x_1 represents x_n]

   Plot (int (x_2) , int (y_2))

   $x_1 = x_2$;

   $y_1 = y_2$;

   [Reinitialize the current point]

   } while (y_1 – start_y) < ε or (start_x – x_1) > ε

   [check if the current point is the starting point or not. If current point is not starting point repeat step 4 ; otherwise stop]

5. Stop.

'C' code for DDA Circle Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float x1,y1,x2,y2,startx,starty,epsilon;
int gd,gm,i,val;
int r;
clrscr();

/* Read two end points of line
---------------------------------- */
printf("Enter the radius of a circle :");
scanf("%d",&r);

/* Initialise graphics mode
---------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/* Initialise starting point
------------------------------- */
x1=r*cos(0);
```

```
    y1=r*sin(0);
    startx = x1;
    starty = y1;

    /*Calculations for epsilon
    ----------------------------*/
    i=0;
    do
    {
    val = pow(2,i);
    i++;
    }while(val<r);
    epsilon = 1/pow(2,i-1);

    do
       {
        x2= x1 + y1*epsilon;
        y2 = y1 - epsilon*x2;
        putpixel(200+x2,200+y2,15);

    /* Reinitialise the current point
    ---------------------------------- */
        x1=x2;
        y1=y2;
        delay(1000);    /* Delay is purposely inserted to see
                           observe the line drawing process */
       }
     while( (y1 - starty ) < epsilon || (startx - x1) > epsilon);
    getch();
    closegraph();
    }
```

2.9.2 Bresenham's Circle Drawing Algorithm

The Bresenham's circle drawing algorithm considers the eight-way symmetry of the circle to generate it. It plots $1/8^{th}$ part of the circle, i.e. from 90° to 45°, as shown in the Fig. 2.16. As circle is drawn from 90° to 45°, the x moves in positive direction and y moves in the negative direction.

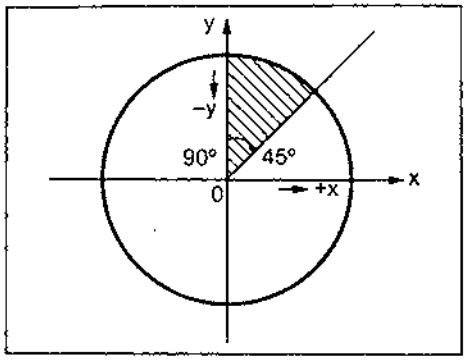


Fig. 2.16 1/8 part of circle

To achieve best approximation to the true circle we have to select those pixels in the raster that fall the least distance from the true circle. Refer Fig. 2.17. Let us observe the 90° to 45° portion of the circle. It can be noticed that if points are generated from 90° to 45°, each new point closest to the true circle can be found by applying either of the two options :

- Increment in positive x direction by one unit or
- Increment in positive x direction and negative y direction both by one unit

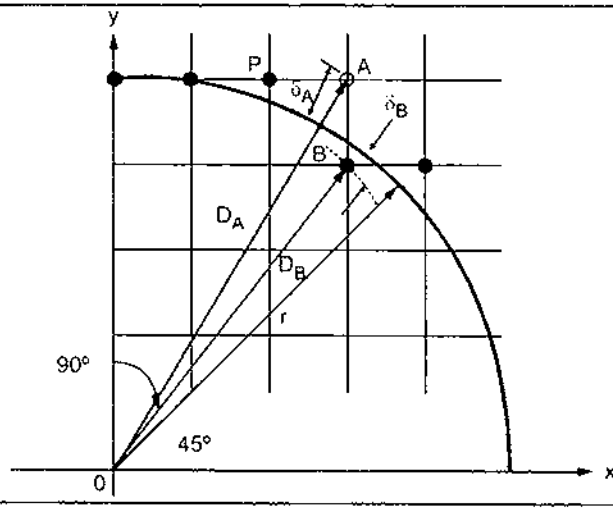


Fig. 2.17 Scan conversion with Bresenham's algorithm

Let us assume point P in Fig. 2.17 as a last scan converted pixel. Now we have two options either to choose pixel A or pixel B. The closer pixel amongst these two can be determined as follows

The distances of pixels A and B from the origin are given as

$$D_A = \sqrt{(x_{i+1})^2 + (y_i)^2} \quad \text{and}$$

$$D_B = \sqrt{(x_{i+1})^2 + (y_i - 1)^2}$$

Now, the distances of pixels A and B from the true circle are given as

$$\delta_A = D_A - r \text{ and } \delta_B = D_B - r$$

However, to avoid square root term in derivation of decision variable, i.e. to simplify the computation and to make algorithm more efficient the δ_A and δ_B are defined as

$$\delta_A = D_A^2 - r^2 \quad \text{and}$$

$$\delta_B = D_B^2 - r^2$$

From Fig. 2.17, we can observe that δ_A is always positive and δ_B always negative. Therefore, we can define **decision variable** d_i as

$$d_i = \delta_A + \delta_B$$

and we can say that, if $d_i < 0$, i.e., $\delta_A < \delta_B$ then only x is incremented; otherwise x is incremented in positive direction and y is incremented in negative direction. In other words we can write,

For $d_i < 0$, $x_{i+1} = x_i + 1$ and

For $d_i \geq 0$, $x_{i+1} = x_i + 1$ and $y_{i+1} = y_i - 1$

The equation for d_i at starting point, i.e. at x = 0 and y = r can be simplified as follows

$$\begin{aligned} d_i &= \delta_A + \delta_B \\ &= (x_i + 1)^2 + (y_i)^2 - r^2 + (x_i + 1)^2 + (y_i - 1)^2 - r^2 \\ &= (0 + 1)^2 + (r)^2 - r^2 + (0 + 1)^2 + (r - 1)^2 - r^2 \\ &= 1 + r^2 - r^2 + 1 + r^2 - 2r + 1 - r^2 \\ &= 3 - 2r \end{aligned}$$

Similarly, the equations for d_{i+1} for both the cases are given as

For $d_i < 0$, $d_{i+1} = d_i + 4\,x_i + 6$ and

For $d_i \leq 0$, $d_{i+1} = d_i + 4\,(x_i - y_i) + 10$

Algorithm to plot 1/8 of the circle

1. Read the radius (r) of the circle.
2. d = 3 – 2 r

   [Initialize the decision variable]
3. x = 0, y = r

   [Initialize starting point]
4. do

```
{
    plot (x, y)
    if (d < 0) then
    {
        d = d + 4x + 6
    }
    else
    { d = d + 4 (x – y) + 10
      y = y – 1
    }
    x = x + 1
} while (x < y)
```

5. Stop

The remaining part of circle can be drawn by reflecting point about y axis, x axis and about origin as shown in Fig. 2.18.

Therefore, by adding seven more plot commands after the plot command in the step 4 of the algorithm, the circle can be plotted. The remaining seven plot commands are :

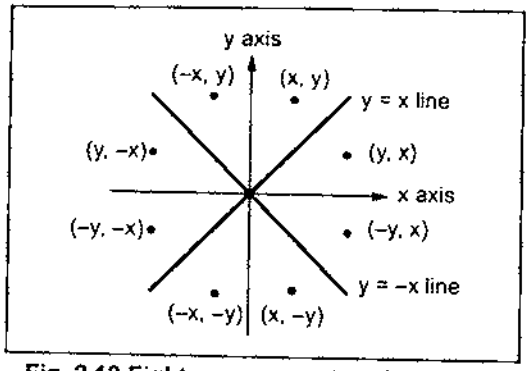


Fig. 2.18 Eight-way symmetry of the circle

plot (y, x)
plot (y, – x)
plot (x, – y)
plot (– x, – y)
plot (– y, – x)
plot (– y, x) and
plot (– x, y)

'C' code for Bresenham's Circle Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float d;
int gd,gm,x,y;
int r;
clrscr();

/* Read the radius of the circle
-----------------------------------------*/
printf("Enter the radius of a circle :");
scanf("%d",&r);

/* Initialise graphics mode
-------------------------------*/
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/* Initialise starting points
-------------------------------*/
x = 0;
y = r;
```

```
/* initialise the decision variable
----------------------------------------*/
d = 3 - 2 * r;

do
{       putpixel(200+x,200+y,15);
        putpixel(200+y,200+x,15);
        putpixel(200+y,200-x,15);
        putpixel(200+x,200-y,15);
        putpixel(200-x,200-y,15);
        putpixel(200-y,200-x,15);
        putpixel(200-y,200+x,15);
        putpixel(200-x,200+y,15);
        if (d <= 0)
        {
        d = d + 4*x + 6;
        }
        else
        {
        d = d + 4*(x-y) + 10;
        y = y - 1;
        }
        x = x + 1;
        delay(1000);  /* Delay is purposely inserted to see
                         observe the line drawing process */
        }
        while(x < y);
getch();
closegraph();
}
```

2.9.3 Midpoint Circle Drawing Algorithm

The midpoint circle drawing algorithm also uses the eight-way symmetry of the circle to generate it. It plots 1/8 part of the circle, i.e. from 90° to 45°, as shown in the Fig. 2.19. As circle is drawn from 90 to 45°, the x moves in the positive direction and y moves in the negative direction. To draw a 1/8 part of a circle we take unit steps in the positive x direction and make use of decision parameter to determine which of the two possible y positions is closer to the circle path at each step. The Fig. 2.19 shows the two possible y positions (y_i and $y_i + 1$) at sampling position $x_i + 1$. Therefore, we have to determine whether the pixel at position

$(x_i + 1, y_i)$ or at position $(x_i + 1, y_i - 1)$ is closer to the circle. For this purpose decision parameter is used. It uses the circle function ($f_{circle}(x, y) = x^2 + y^2 - r^2$) evaluated at the midpoint between these two pixels.

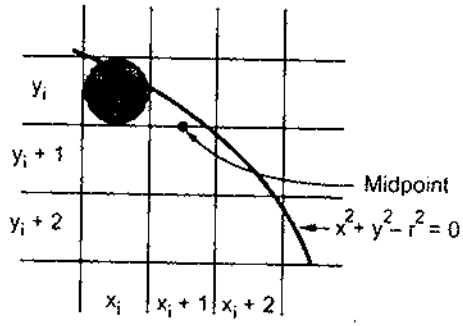


Fig. 2.19 Decision parameter to select correct pixel in circle generation algorithm

$$d_i = f_{circle}(x_i + 1, y_i - \frac{1}{2})$$

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

$$= (x_i + 1)^2 + y_i^2 - y_i + \frac{1}{4} - r^2 \quad \ldots (2.6)$$

If $d_i < 0$, this midpoint is inside the circle and the pixel on the scan line y_i is closer to the circle boundary. If $d_i \geq 0$, the midposition is outside or on the circle boundary, and $y_i - 1$ is closer to the circle boundary. The incremental calculation can be determined to obtain the successive decision parameters. We can obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{i+1} + 1 = x_i + 2$.

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

$$= [(x_i + 1) + 1]^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2$$

$$= (x_i + 1)^2 + 2(x_i + 1) + 1 + y_{i+1}^2 - (y_{i+1}) + \frac{1}{4} - r^2 \quad \ldots (2.7)$$

Looking at equations 2.6 and 2.7 we can write

$$d_{i+1} = d_i + 2(x_i + 1) + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i) + 1$$

where y_{i+1} is either y_i or y_{i-1}, depending on the sign of d_i.

If d_i is negative, $y_{i+1} = y_i$

$\therefore$ $d_{i+1} = d_i + 2(x_i + 1) + 1$

$$= d_i + 2x_{i+1} + 1 \quad \ldots (2.8)$$

If d_i is positive, $y_{i+1} = y_{i-1}$

$\therefore$ $d_{i+1} = d_i + 2(x_i + 1) + 1 - 2y_{i+1}$ ... (2.9)

The terms $2x_{i+1}$ and $-2y_{i+1}$ in equations (2.8) and (2.9) can be incrementally calculated as

$$2x_{i+1} = 2x_i + 2$$
$$2y_{i+1} = 2y_i - 2$$

The initial value of decision parameter can be obtained by evaluating circle function at the start position $(x_0, y_0) = (0, r)$.

$$d_0 = f_{circle}\left((0+1)^2 + \left(r - \frac{1}{2}\right)^2 - r^2\right)$$
$$= 1 + \left(r - \frac{1}{2}\right)^2 - r^2$$
$$= 1.25 - r$$

Algorithm

1. Read the radius (r) of the circle
2. Initialize starting position as
   x = 0
   y = r
3. Calculate initial value of decision parameter as
   P = 1.25 – r
4. do
```
    {   plot (x, y)
        if (d < 0)
            {   x = x + 1
                y = y
                d = d + 2x + 1
            }
        else
            {   x = x + 1
                y = y – 1
                d = d + 2x + 2y + 1
    }
        while (x < y)
```
5. Determine symmetry points
6. Stop.

'C' code for Midpoint Circle Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

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

```
#include<math.h>
main()
{
float  p;
int i,gd,gm,x,y;
int r;

/* initialise graphics
------------------------ */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/* Read the radius
----------------------- */
printf("Enter the radius of the circle :");
scanf("%d",&r);

x=0;
y=r;
p = 1.25 - r;
do
{
      putpixel(200+x,200+y,15);
      putpixel(200+y,200+x,15);
      putpixel(200+x,200-y,15);
      putpixel(200+y,200-x,15);
      putpixel(200-x,200-y,15);
      putpixel(200-x,200+y,15);
      putpixel(200-y,200+x,15);
      putpixel(200-y,200-x,15);

      if (p < 0)
      {
      x = x+1;
      y = y;
      p = p + 2*x + 1;
      }
      else
```

```
        {
        x= x+1;
        y= y-1;
        p = p + 2*(x-y) + 1;
        }
        delay(10000);
        }
        while(x < y);
getch();
closegraph();
}
```

2.10 Ellipse Drawing Algorithm

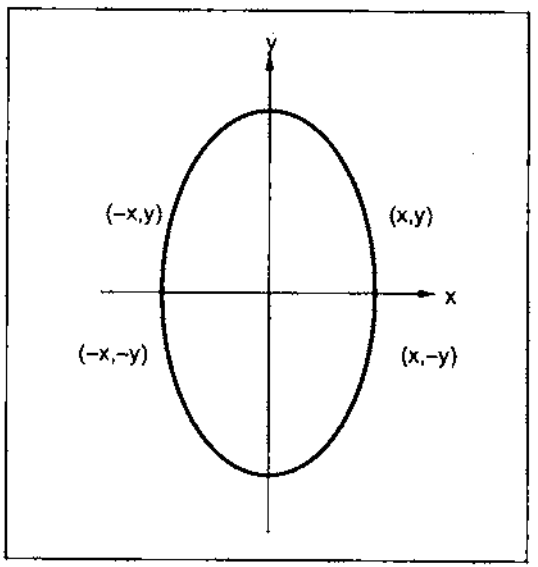


Fig. 2.20 Four way symmetry of ellipse

The midpoint ellipse drawing algorithm uses the four way symmetry of the ellipse to generate it. The Fig. 2.20 shows the four-way symmetry of ellipse. This approach is similar to that used in displaying a raster circle. Here, the quadrant of the ellipse is divided into two regions. The Fig. 2.21 shows the division of the first quadrant according to the slope of an ellipse with $r_x < r_y$. As ellipse is drawn from 90° to 0°, the x moves in the positive direction and y moves in the negative direction, and ellipse passes through two regions. It is important to note that while processing first quadrant we have to take steps in the x direction where the slope of the curve has a magnitude less than 1 (for region 1) and to take steps in the y direction where the slope has a magnitude greater than 1 (for region 2).

Like circle function, the ellipse function $f_{ellipse\,(x,\,y)}\left(r_y^2\ x^2 + r_x^2\ y^2 - r_x^2\ r_y^2\right)$ serves as the decision parameter in the midpoint algorithm. At each sampling position, the next pixel along the ellipse path is selected according to the sign of the ellipse function evaluated at midpoint between the two candidate pixels ($x_i + 1$, y_i or $x_i + 1$, $y_i - 1$ for region 1 and x_i, $y_i - 1$ or $x_i + 1$, $y_i - 1$ for region 2).

Starting at $(0, r_y)$ we have to take unit steps in the x direction until we reach the boundary between region 1 and region 2. Then we have to switch to unit steps in the y direction over the remainder of the curve in the first quadrant. To check for boundary point between region 1 and region 2 we have to test the value of the slope of the curve at each step. The slope of the ellipse at each step is given as

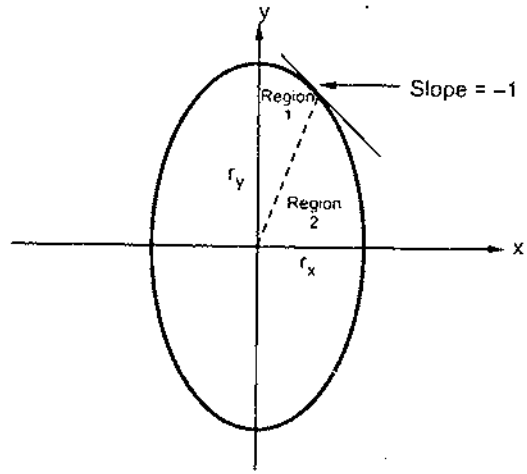


Fig. 2.21 Ellipse processing regions

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

At the boundary point between region 1 and region 2, $dy/dx = -1$ and

$$2r_y^2 x = 2r_x^2 y$$

Therefore, when

$$2r_y^2 x \geq 2r_x^2 y$$

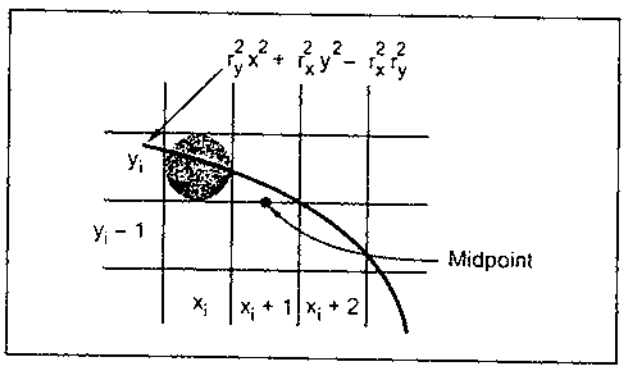


Fig. 2.22

We have to switch to unit steps in the y direction over the remainder of the curve in the first quadrant. The Fig. 2.22 shows the midpoint between the two candidate pixels at sampling position $x_i + 1$ in the first region. The next position along the ellipse path can be evaluated by decision parameter at this midpoint.

$$d_{1i} = f_{ellipse}\left(x_i + 1, y_i - \frac{1}{2}\right)$$

$$= r_y^2 (x_i + 1)^2 + r_x^2\left(y_i - \frac{1}{2}\right)^2 - r_x^2\, r_y^2$$

If $d_{1i} < 0$, the midpoint is inside the ellipse and the pixel on scan line y_i is closer to the ellipse boundary. If $d_{1i} \geq 0$, the midpoint is outside or on the ellipse boundary, and the pixel on the scan line $y_i - 1$ is closer to the ellipse boundary.

The incremental calculation of decision parameter of region 1 can be given as

$$d_{1i+1} = f_{ellipse}\left(x_{i+1}+1, y_{i+1}-\frac{1}{2}\right)$$

$$= r_y^2\,[(x_i+1)+1]^2 + r_x^2\left(y_{i+1}-\frac{1}{2}\right)^2 - r_x^2\,r_y^2$$

Substituting value of d_{1i} in above expression we get,

$$d_{1i+1} = d_{1i} + 2r_y^2\,(x_i+1) + r_y^2 + r_x^2\left[\left(y_{i+1}-\frac{1}{2}\right)^2 - \left(y_i-\frac{1}{2}\right)^2\right]$$

where y_{i+1} is either y_i or $y_i - 1$, depending on the sign of d_{1i}.

If d_{1i} is negative, i.e. $d_{1i} < 0$, $y_{i+1} = y_i$

$$\therefore \quad d_{1i+1} = d_{1i} + 2r_y^2\,x_{i+1} + r_y^2$$

If d_{1i} is positive or zero, i.e. $d_{1i} \geq 0$, $y_{i+1} = y_i - 1$

$$\therefore \quad d_{1i+1} = d_{1i} + 2r_y^2\,x_{i+1} + r_y^2 - 2r_x^2\,y_{i+1}$$

The terms $2r_y^2 x$ and $2r_x^2 y$ can be incrementally calculated as

$$2r_y^2\,x_{i+1} = 2r_y^2 x_i + 2r_y^2 \text{ and}$$

$$2r_x^2 y_{i+1} = 2r_x^2 y_i - 2r_x^2$$

In region 1, the initial value of the decision parameter can be obtained by evaluating the ellipse function at the start position $(x_0, y_0) = (0, r_y)$.

$$d_{10} = f_{ellipse}\left(1, r_y - \frac{1}{2}\right)$$

$$= r_y^2 + r_x^2\left(r_y - \frac{1}{2}\right)^2 - r_x^2\,r_y^2$$

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

For region 2, we sample at unit steps in the negative y direction, and the midpoint is now taken between horizontal pixels, at each step, as shown in the Fig. 2.22. For this region, the decision parameter is evaluated as

$$d_{2i} = f_{ellipse}\left(x_i + \frac{1}{2}, y_i - 1\right)$$

$$= r_y^2\left(x_i + \frac{1}{2}\right)^2 + r_x^2\,(y_i - 1)^2 - r_x^2\,r_y^2$$

If $d_{2i} > 0$, the midpoint is outside the ellipse boundary, and we select the pixel at x_i.

If $d_{2i} \leq 0$, the midpoint is inside or on the ellipse boudnary, and we select pixel position x_{i+1}. The incremental decision parameters for region 2 can be given as

$$d_{2i+1} = f_{ellipse}\left(x_{i+1} + \frac{1}{2}, y_{i+1} - 1\right)$$

$$= r_y^2\left(x_{i+1} + \frac{1}{2}\right)^2 + r_x^2\,[(y_i - 1) - 1]^2 - r_x^2\, r_y^2$$

Substituting value of expression d_{2i} in above expression we get,

$$d_{2i+1} = d_{2i} - 2r_x^2\,(y_i - 1) + r_x^2 + r_y^2\left[\left(x_{i+1} + \frac{1}{2}\right)^2 - \left(x_i + \frac{1}{2}\right)^2 2\right]$$

where x_{i+1} set either to x_i or to $x_i + 1$, depending on the sign of d_{2i}.

In region 2, the intial value of the decision parameter can be obtained by evaluating the ellipse function at the last position in the region 1.

$$\therefore \quad d_{20} = f_{ellipse}\left(x_0 + \frac{1}{2}, y_0 - 1\right)$$

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

Algorithm

1. Read radii r_x and r_y.
2. Initialise starting point as

   $x = 0$

   $y = r_y$
3. Calculate the initial value of decision parameter in region 1 as

   $d_1 = r_y^2 - r_x^2\, r_y + \frac{1}{4} r_x^2$
4. Initialize dx and dy as

   $dx = 2\, r_y^2 x$

   $dy = 2\, r_x^2 y$
5. do

```
{   plot (x, y)
    if (d1 < 0)
      {
        x = x + 1
        y = y
        dx = dx + 2 ry^2
        d1 = d1 + dx + ry^2
        [d1 = d1 + 2 ry^2 x + 2 ry^2 + ry^2]
      }
```

```
        else
            {
                x = x + 1
                y = y - 1
                dx = dx + 2 r_y^2
                dy = dy - 2 r_x^2
                d_1 = d_1 + dx - dy + r_y^2
                [d_1 = d_1 + 2 r_y^2 x + 2 r_y^2 - (2 r_x^2 y - 2 r_x^2) + r_y^2]
    } while (dx < dy)
```

6. Calculate the initial value of decision parameter in region 2 as

$$d_2 = r_y^2\left(x+\frac{1}{2}\right)^2 + r_x^2(y-1)^2 - r_x^2\, r_y^2$$

```
7.  do
    {   Plot (x, y)
        if (d_2 > 0)
            {   x = x
                y = y - 1
                dy = dy - 2 r_x^2
                d_2 = d_2 - dy + r_x^2
                [d_2 = d_2 - (2 r_x^2 y - 2 r_x^2) + r_x^2]
            }
        else
            {
                x = x + 1
                y = y - 1
                dy = dy - 2 r_x^2
                dx = dx + 2 r_y^2
                d_2 = d_2 + dx - dy + r_x^2
                [d_2 = d_2 + 2 r_y^2 x + 2 r_y^2 - (2 r_x^2 y - 2 r_x^2) + r_y^2]
    } while (y > 0)
```

7. Determine symmetrical points in other three quadrants.
8. Stop.

'C' code for Midpoint Ellipse Drawing Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
long  d1,d2;
int i,gd,gm,x,y;
long rx, ry, rxsq,rysq,tworxsq,tworysq,dx,dy;

/* Read the radius x and y
---------------------------- */
printf("Enter the  x radius of the ellipse :");
scanf("%ld",&rx);

printf("Enter the y radius of the ellipse :");
scanf("%ld",&ry);

/* initialise graphics
------------------------ */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

rxsq = rx * rx;
rysq = ry * ry;
tworxsq = 2 * rxsq;
tworysq = 2 * rysq;

x = 0;
y = ry;
d1 = rysq - rxsq * ry + (0.25 * rxsq) ;
dx = tworysq * x;
dy = tworxsq * y;
do
{
```

```
putpixel(200+x,200+y,15);
putpixel(200-x,200-y,15);
putpixel(200+x,200-y,15);
putpixel(200-x,200+y,15);


if (d1 < 0)
{
x = x + 1;
y = y;
  dx = dx + tworysq;
d1 = d1 + dx + rysq;
}
else
{
x = x + 1;
y = y - 1;
dx = dx + tworysq;
dy = dy - tworxsq;
d1 = d1 + dx - dy + rysq;
}
delay(10);
}
while( dx < dy);
d2 = rysq* (x+0.5) * (x+0.5)+rxsq*(y-1)*(y-1)-rxsq*rysq  ;
do
{
putpixel(200+x,200+y,15);
putpixel(200-x,200-y,15);
putpixel(200+x,200-y,15);
putpixel(200-x,200+y,15);

if(d2 > 0)
{
  x = x;
  y = y -1;
dy = dy -tworxsq;
d2 = d2  - dy + rxsq;
}
else
{ x  = x+1;
  y  = y -1;
 dy  = dy -tworxsq;
```

```
            dx  = dx + tworysq;
            d2  = d2 + dx - dy + rxsq;
           }
    } while (y > 0);
    getch();
    closegraph();
    }
```

Solved Examples

Ex. 2.4 *The turnover of ABC Company for the following divisions is indicated below :*

Construction	*Rs. 125 Crores*
Engineering	*Rs. 90 Crores*
Shipping	*Rs. 530 Crores*
Consumer products	*Rs 140 crores*
Agro-tech	*Rs 115 Crores*

Write a program to represent this information in a pie-chart. **(May-97)**

Sol. : The 'C' code for above program is as given below

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float  sum=0,comp[5],d0,angle,theta,x,y,xc,yc;
int i,gd,gm;

/* initialise graphics
------------------------ */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/* Read the turnover values */
for(i=0;i<5;i++)
{
printf("Enter the turnover value for company%d :",i+1);
scanf("%f",&comp[i]);
}
for(i=0;i<5;i++)
{
```

```
sum = sum + comp[i];
}
for(i=0;i<5;i++)
{
comp[i] = (comp[i]/sum)*360;
}
xc = 320;
yc = 240;
x = 420;
y = 240;
d0 = 1/(3.2 * (abs(x-xc) + abs(y-yc)));
angle = 0;
theta = 0;
      for(i=0;i<5;i++)
      {
      theta = theta + comp[i];
      while (angle*180/3.142 < theta )
            {
             setcolor(i+1);
             line(xc,yc,x,y);
             x = x - (y-yc) * d0;
             y = y + (x - xc)*d0;
             angle = angle + d0;
             delay(100);
             }
      }
getch();
closegraph();
}
```

Ex. 2.5 *Consider the following circuit where the switch , is closed at t = 0. Develop a program to display the plot of Vo(t) for t>0.*

(May-2000)

Sol. : The 'C' code for above program is as given below

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
```

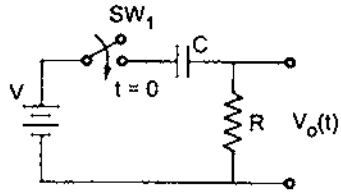


Fig. 2.23

```
main()
{
float d,t,c,r,vo,v,T;
int i,gd,gm,x,y;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");
printf("Enter the value of battery voltage :");
scanf("%f",&v);
printf("Enter the value of capacitor :");
scanf("%f",&c);
printf("Enter the value of resistor :");
scanf("%f",&r);
t=  0.0;
/* Draw axis */
line(50,400,600,400);
line(100,440,100,50);
for(i=0;i<450;i++)
{
T = -t/(c*r);
vo = v - (v*(1-exp(T)));
t = t + 0.001;
putpixel(100+t*1000,400-vo*(300/v),15);
delay(1000);
}
getch();
closegraph();
}
```

Review Questions

1. Explain the basic concept in line drawing.
2. Explain the steps in the incremental line drawing algorithm.

3. Explain the steps in DDA line drawing algorithm.
4. Discuss merits and demerits of DDA line drawing algorithm.
5. Explain the steps in Bresenham's line drawing algorithm.
6. Explain the steps in generalized Bresenham's line drawing algorithm.
7. What is aliasing ? Explain different methods of minimizing its effect.
8. Write a short note on thick line segment.
9. Explain the basic concept in circle drawing.
10. Give different methods of representing a circle.
11. Explain the steps in DDA circle drawing algorithm.
12. Explain the steps in Bresenham's circle drawing algorithm.
13. Explain the steps in midpoint circle drawing algorithm.
14. Explain the steps in midpoint ellipse drawing algorithm.

University Questions

1. Develop a procedure/function for the Bresenham's line algorithm.
   Your algorithm should take care of lines of any given slope. **(Dec-96)**
2. Using the above, write a program to plot a line graph over a set of values stored in an array "Vals". Properly scale the values before plotting. **(Dec-96)**
3. The turnover of ABC Company for the following divisions is indicated below :

Construction	Rs. 125 Crores
Engineering	Rs. 90 Crores
Shipping	Rs. 530 Crores
Consumer products	Rs 140 crores
Agro-tech	Rs 115 Crores

   Write a program to represent this information in a pie-chart. **(May-97, Dec-2001)**
4. Develop the Bresenham's line algorithm to draw lines of any slope. Compare this with the DDA algorithm. **(Dec-97)**
5. Develop a program to display a pie-chart over a set of values. You are allowed to use only the set-pixel () or put-pixel () primitive. **(Dec-97)**

   [Hint : Replace a line function in program given for question 8 with a line function using putpixel, i.e. use DDA or Bresenham's algorithm to develop line function]
6. Write an algorithm for Bresenham Circle Generation and then using it produce sequence of atleast five points along the circumference of circle with radius = 20 and centered at (50; 50) **(May-98)**
7. Write an algorithm for Bresenham Line Generation which will work for all slopes. Calculate the pixel positions along a straight line between P_1 (20, 20) and P_2 (10, 12). **(May-98)**

8. Develop an ellipse generation algorithm and use it to display elliptical arcs as shown below.

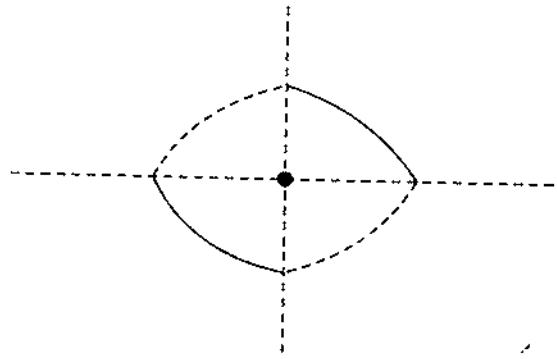


Fig. 2.24

(Dec-98)

[Note : Only solid arc is displayed : centre is at (xc, yc)]

9. What is antialiasing ? How can it be reduced? (May-99)
10. How is Bresenham's technique advantageous ? (May-99)
11. Explain the principle of any of the antialiasing techniques. (Dec-99)
12. Derive the algorithm for drawing Bresenham line between the end points : (100, 100) and (150, 200) (Dec-99)
13. Repeat the above question for the end points (100, 100) and (200, 150). (Dec-99)
14. Compute the first three Bresenham points in each of the cases mentioned in above two questions. (Dec-99)
15. Develop a program to plot a bar graph over a set of 'n' values stored in an array 'data'. Ensure that the values are scaled properly to fit onto the display extents. (Dec-2000)
16. Derive Bresenham's line drawing algorithm which will work for line of any slope. Compute the first five points for the line segment A(20, 20) B(12, 10). (May-2001)
17. Derive and give the modified Bresenham's line drawing algorithm which will work for line of any slope. (Dec-2001)
18. A dash line is shown below. Develop an algorithm to draw a dash line from point $A(x_1, y_1)$ to point $B(x_2, y_2)$. The length of dash is d pixels and length of gap between dash is g pixels.

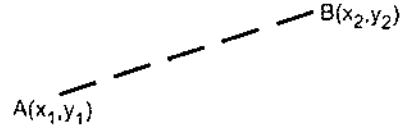


Fig. 2.25

(Dec-2001, May-2002)

19. Two points on the circumference of the circle represented by $(x - 50)^2 + (y - 50)^2 = 100$ are given by (50, 60) and (60, 50).

    Compute the pixel locations to be highlighted between the above points using Bresenham's circle algorithm. **(Dec-2001)**

20. Write a short note on antialiasing techniques. **(Dec-2001)**

21. Explain the different antialiasing methods. **(May-2002)**

22. What are the advantages of Bresenham's line drawing algorithm ?

    Bresenham's line drawing algorithm is considered as most efficient algorithm when compared with DDA line drawing algorithm. State the reason. **(May-2003)**

23. Illustrate antialiasing. **(May-2003)**

□□□

3

Area Filling

3.1 Introduction

We have seen scan conversion of lines, circles and elipse in the last chapter. In this chapter we are going to study different types of polygons, their representation and filling algorithms for them.

A polyline is a chain of connected line segments. It is specified by giving the vertices (nodes) $P_0, P_1, P_2 \ldots$ and so on. The first vertex is called the initial or starting point and the last vertex is called the final or terminal point, as shown in the Fig. 3.1 (a). When starting point and terminal point of any polyline is same, i.e. when polyline is closed then it is called polygon. This is illustrated in Fig. 3.1 (b).

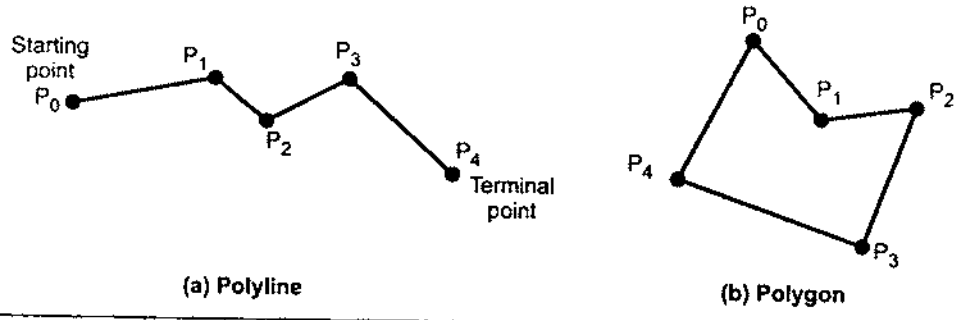


Fig. 3.1

3.2 Types of Polygons

The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie. There are two types of polygons :

- Convex and
- Concave

A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon. The Fig. 3.2 shows the examples of convex polygons.

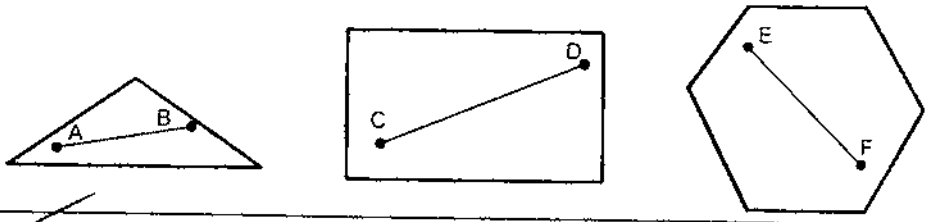


Fig. 3.2 Convex polygons

A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon. The Fig. 3.3 shows the examples of concave polygons.

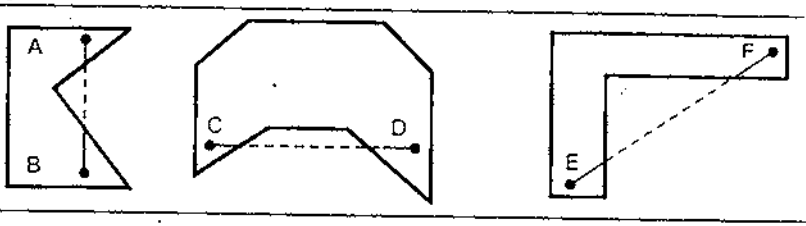


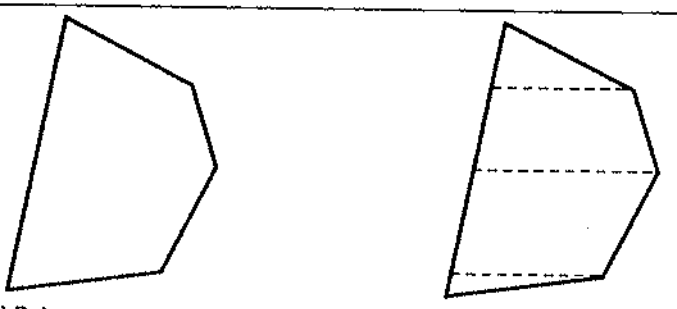
Fig. 3.3 Concave polygons

3.3 Representation of Polygons

We have seen that closed polyline is a polygon. Each polygon has sides and edges. The end points of the sides are called the polygon vertices. To add polygon to our graphics system, we must first decide how to represent it. There are three approaches to represent polygons according to the graphics systems :

- Polygon drawing primitive approach
- Trapezoid primitive approach
- Line and point approach

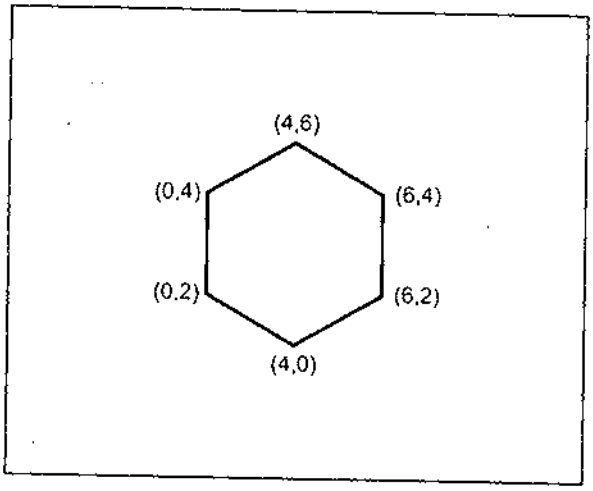
Some graphics devices supports polygon drawing primitive approach. They can directly draw the polygon shapes. On such devices polygons are saved as a unit. Some graphics devices support trapezoid primitive. In such devices, trapezoids are formed from two scan lines and two line segments as shown in the Fig. 3.4. Here, trapezoids are drawn by stepping down the line segments with two vector generators and, for each scan line, filling in all the pixels between them. Therefore every polygon is broken up into trapezoids and it is represented as a series of trapezoids.



(a) Polygon (b) Polygon as a series of trapezoids

Fig. 3.4 Representation of polygon

Most of the other graphics devices do not provide any polygon support at all. In such cases polygons are represented using lines and points. A polygon is represented as a unit and it is stored in the display file. In a display file polygon can not be stored only with series of line commands because they do not specify how many of the following line commands are the part of the polygon. Therefore, new command is used in the display file to represent polygons. The opcode for new command itself specify the number of line segments in the polygon. The Fig. 3.5 shows the polygon and its representation using display file.



DF_OP	DF_ x	DF_ y
6	0	2
2	0	4
2	4	6
2	6	4
2	6	2
2	4	0
2	0	2

Fig. 3.5 Polygon and its representation using display file

3.4 Entering Polygons

Let us see how to enter polygon command and data into the display file. We know that, we have to enter number of sides and the coordinates of the vertex points. This information can be entered using a following algorithm.

Algorithm : Entering the polygon into the display file

1. Read AX and AY of length N

   [AX and AY are arrays containing the vertices of the polygon and N is the number of polygon sides]

2. i = 0 [Initialize counter to count number of sides]

   DF_OP [i] ← N

   DF_ x [i] ← AX [i]

   DF_ y [i] ← AY [i]

   i ← i + 1

   [Load polygon command]

3. do

   {

   DF_OP [i] ← 2

   DF_ x [i] ← AX [i]

```
        DF_ y [i] ← AY [i]
        i ← i + 1
    }
    While (i < N)   [Enter line commands]
4. DF_OP [i] ← 2
    DF_ x [i] ← AX [0]
    DF_ y [i] ← AY [0]
    [Enter last line command]
5. Stop
```

3.5 An Inside Test

Once the polygon is entered in the display file, we can draw the outline of the polygon. To show polygon as a solid object we have to set the pixels inside the polygon as well as pixels on the boundary of it. Now the question is how to determine whether or not a point is inside of a polygon. One simple method of doing this is to construct a line segment between the point in question and a point known to be outside the polygon, as shown in the Fig. 3.6. Now count how many intersections of the line segment with the polygon boundary occur. If there are an odd number of intersections, then the point in question is inside; otherwise it is outside. This method is called the **even-odd method** of determining polygon inside points.

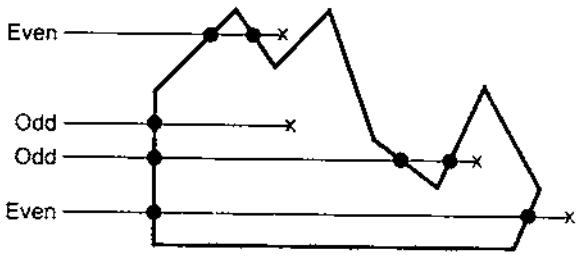


Fig. 3.6

If the intersection point is vertex of the polygon then we have to look at the other endpoints of the two segments which meet at this vertex. If these points lie on the same side of the constructed line, then the point in question counts as an even number of intersections. If they lie on opposite sides of the constructed line, then the point is counted as a single intersection. This is illustrated in Fig. 3.7.

Another approach to do the inside test is the **winding-number method**. Let us consider a point in question and a point on the polygon boundary. Conceptually, we can stretch a piece of elastic between these points and slide the end point on the polygon boundary for one complete rotation. We can then examine the point in question to see how many times the

elastic has wound around it. If it is wound at least once, then the point is inside. If there is no net winding, then the point is outside.

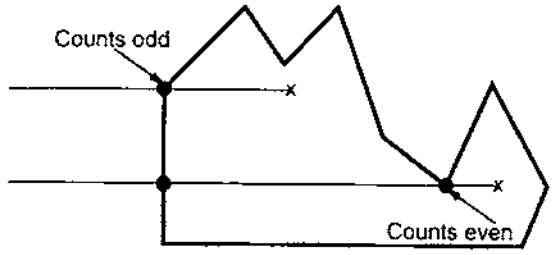


Fig. 3.7

Like even-odd method, in winding number method we have to picturise a line segment running from outside the polygon to the point in question and consider the polygon sides which it crosses. Here, instead of just counting the intersections, we have to give a **direction number** to each boundary line crossed, and we have to sum these direction numbers. The direction number indicates the direction the polygon edge was drawn relative to the line segment we have constructed for the test. This is illustrated in Fig. 3.8

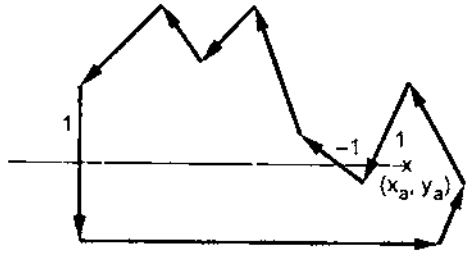


Fig. 3.8

As shown in the Fig. 3.8, point (x_a, y_a) is a test point and line $y = y_a$ is a horizontal line runs from outside the polygon to point (x_a, y_a). The polygon edges crossed by this line could be drawn in two ways. The edge could be drawn starting below the line, cross it, and end above the line or starting above the line, cross it, and end below the line. In first case we have to give direction number as −1 and in the second case we have give direction number as 1. After giving the direction numbers we have to take sum of these direction numbers which indicates whether the point in inside the polygon or not. The sum of the direction numbers for the sides that cross the constructed horizontal line segment is called the **winding number** for the point in question. For polygons or two dimensional objects, the point is said to be inside when the value of winding number is nonzero.

3.6 Polygon Filling

Filling the polygon means highlighting all the pixels which lie inside the polygon with any colour other than background colour. Polygons are easier to fill since they have linear boundaries.

There are two basic approaches used to fill the polygon. One way to fill a polygon is to start from a given "seed", point known to be inside the polygon and highlight outward from this point i.e. neighbouring pixels until we encounter the boundary pixels. This approach is called **seed fill** because colour flows from the seed pixel until reaching the polygon boundary, like water flooding on the surface of the container. Another approach to fill the polygon is to apply the inside test i.e. to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon. This approach is known as **scan-line algorithm**. It avoids the need for a seed pixel but it requires some computation. Let us see these two methods in detail.

3.6.1 Seed Fill

The seed fill algorithm is further classified as flood fill algorithm and boundary fill algorithm. Algorithms that fill interior-defined regions are called **flood-fill algorithms**; those that fill boundary-defined regions are called **boundary-fill algorithms** or **edge-fill** algorithms.

3.6.1.1 Boundary Fill Algorithm

In this method, edges of the polygons are drawn. Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixel is reached. If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.

Boundary defined regions may be either 4-connected or 8-connected as shown in the Fig. 3.9. If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions : left, right, up and down. For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical, and four diagonal directions.

(a) Four connected region (b) Eight connected region

Fig. 3.9

In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm. This is illustrated in Fig. 3.10. Here, a 4-connected algorithm produces the partial fill.

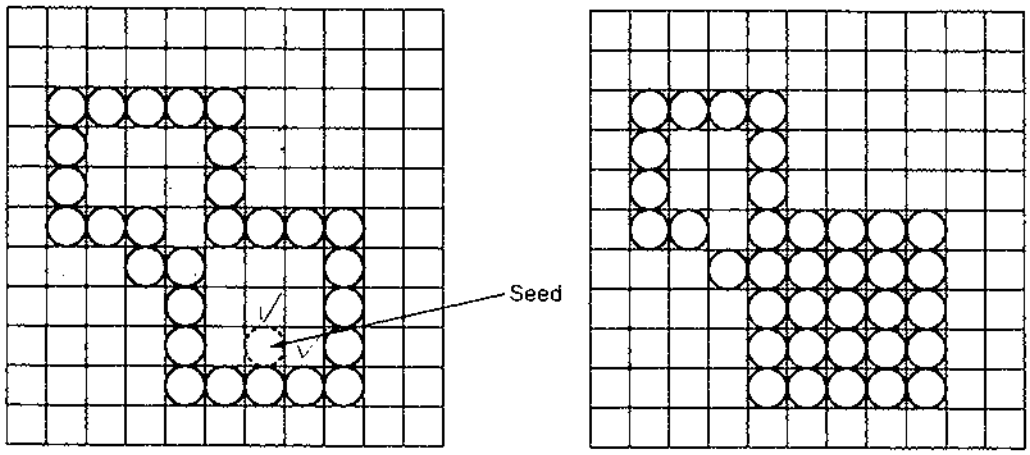


Fig. 3.10 Partial filling resulted using 4-connected algorithm

The following procedure illustrates the recursive method for filling a 4-connected region with colour specified in parameter **fill colour** (f-colour) up to a boundary colour specified with parameter **boundary colour** (b-colour)

Procedure :
```
boundary_fill (x, y, f_colour, b_colour)
{
  if (getpixel (x,y) ! = b_colour && getpixel (x, y) ! = f_colour)
    {       putpixel (x, y, f_colour)
            boundary_fill (x + 1, y, f_colour, b_colour);
            boundary_fill (x, y + 1, f_colour, b_colour);
            boundary_fill (x – 1, y, f_colour, b_colour);
            boundary_fill (x, y – 1, f_colour, b_colour);
    }
}
```

Note : 'getpixel' function gives the colour of specified pixel and 'putpixel' function draws the pixel with specified colour.

Same procedure can be modified according to 8 connected region algorithm by including four additional statements to test diagonal positions, such as (x + 1, y + 1).

'C' code for Boundaryfill Algorithm (8 connected region)

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
main()
{int gd,gm;

/* Initialise graphics mode
-------------------------------------------- */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

rectangle(50,50,100,100);
flood(55,55,4,15);
getch();
closegraph();
}

flood(seed_x,seed_y,foreground_col,background_col)
{
if(getpixel(seed_x,seed_y)!= background_col &&
   getpixel(seed_x,seed_y)!= foreground_col)
  {
   putpixel(seed_x,seed_y,foreground_col);
   flood(seed_x+1,seed_y,foreground_col,background_col);
   flood(seed_x-1,seed_y,foreground_col,background_col);
   flood(seed_x,seed_y+1,foreground_col,background_col);
   flood(seed_x,seed_y-1,foreground_col,background_col);
   flood(seed_x+1,seed_y+1,foreground_col,background_col);
   flood(seed_x-1,seed_y-1,foreground_col,background_col);
   flood(seed_x+1,seed_y-1,foreground_col,background_col);
   flood(seed_x-1,seed_y+1,foreground_col,background_col);
  }
}
```

3.6.1.2 Flood Fill Algorithm

Sometimes it is required to fill in an area that is not defined within a single colour boundary. In such cases we can fill areas by replacing a specified interior colour instead of searching for a boundary colour. This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels. However, here pixels are checked for a specified interior colour instead of boundary colour

and they are replaced by new colour. Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled. The following procedure illustrates the recursive method for filling 8-connected region using flood-fill algorithm.

```
Procedure : flood_fill (x, y, old_colour, new_colour).
{
    if (getpixel (x,  y) = old_colour)
      {  putpixel (x, y, new_colour);
         flood_fill (x + 1, y, old_colour, new_colour);
         flood_fill (x - 1, y, old_colour, new_colour);
         flood_fill (x, y + 1, old_colour, new_colour);
         flood_fill (x, y - 1, old_colour, new_colour);
         flood_fill (x + 1, y + 1, old_colour, new_colour);
         flood_fill (x - 1, y - 1, old_colour, new_colour);
         flood_fill (x + 1, y - 1, old_colour, new_colour);
         flood_fill (x - 1, y + 1, old_colour, new_colour);
      }
}
```

Note : 'getpixel' function gives the colour of specified pixel and 'putpixel' function draws the pixel with specified colour.

3.6.2 Scan Line Algorithm

Recursive algorithm for seed fill methods have got two difficulties : The first difficulty is that if some inside pixels are already displayed in fill colour then recursive branch terminates, leaving further internal pixels unfilled. To avoid this difficulty, we have to first change the colour of any internal pixels that are initially set to the fill colour before applying the seed fill procedures. Another difficulty with recursive seed fill methods is that it cannot be used for large polygons. This is because recursive seed fill procedures require stacking of neighbouring points and in case of large polygons stack space may be insufficient for stacking of neighbouring points. To avoid this problem more efficient method can be used. Such method fills horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighbouring points. This is achieved by identifying the rightmost and leftmost pixels of the seed pixel and then drawing a horizontal line between these two boundary pixels. This procedure is repeated with changing the seed pixel above and below the line just drawn until complete polygon is filled. With this efficient method we have to stack only a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.

The Fig. 3.11 illustrates the scan line algorithm for filling of polygon. For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding positions between each intersection pair are set to the specified fill colour.

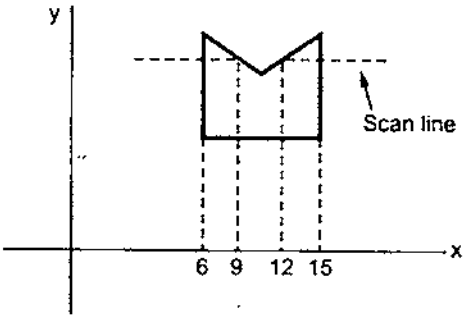


Fig. 3.11

In Fig. 3.11, we can see that there are two stretches of interior pixels from x = 6 to x = 9 and x = 12 to x = 15. The scan line algorithm first finds the largest and smallest y values of the polygon. It then starts with the largest y value and works its way down, scanning from left to right, in the manner of a raster display.

The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary. When intersection points are even, they are sorted from left to right, paired and pixels between paired points are set to the fill colour. But in some cases intersection point is a vertex. When scan line intersects polygon vertex a special handling is required to find the exact intersection points. To handle such cases, we must look at the other endpoints of the two line segments of the polygon which meet at this vertex. If these points lie on the same (up or down) side of the scan line, then the point in question counts as an even number of intersections. If they lie on opposite sides of the scan line, then the point is counted as single intersection. This is illustrated in Fig. 3.12.

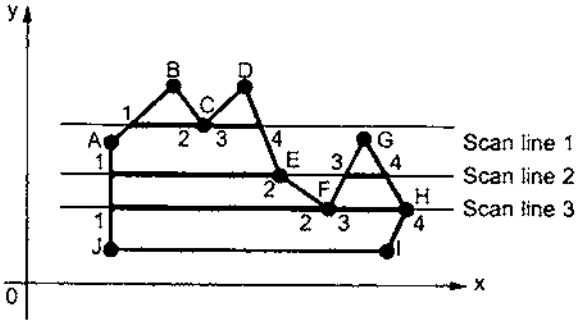


Fig. 3.12 Intersection points along the scan line that intersect polygon vertices

As shown in the Fig. 3.12, each scan line intersects the vertex or vertices of the polygon. For scan line 1, the other end points (B and D) of the two line segments of the polygon lie on the same side of the scan line, hence there are two intersections resulting two pairs : 1 -2 and 3 - 4. Intersections points 2 and 3 are actually same points. For scan line 2 the other endpoints

(D and F) of the two line segments of the polygon lie on the opposite sides of the scan line, hence there is a single intersection resulting two pairs : 1 - 2 and 3 - 4. For scan line 3, two vertices are the intersection points. For vertex F the other end points E and G of the two line segments of the polygon lie on the same side of the scan line whereas for vertex H, the other endpoints G and I of the two line segments of the polygon lie on the opposite side of the scan line. Therefore, at vertex F there are two intersections and at vertex H there is only one intersection. This results two pairs : 1 - 2 and 3 - 4 and points 2 and 3 are actually same points.

We have seen that it is necessary to calculate x intersection points for scan line with every polygon side. We can simplify these calculations by using **coherence properties**. A coherence property of a scene is a property of a scene by which we can relate one part of a scene with the other parts of a scene. Here, we can use a slope of an edge as a coherence property. By using this property we can determine the x intersection value on the lower scan line if the x intersection value for current scan line is known. This is given as

$$x_{i+1} = x_i - \frac{1}{m}$$

where m is the slope of the edge

As we scan from top to bottom value of y coordinates between the two scan line changes by 1.

$$y_{i+1} = y_i - 1$$

Many times it is not necessary to compute the x intersections for scan line with every polygon side. We need to consider only the polygon sides with endpoints straddling the current scan line. See Fig. 3.13.

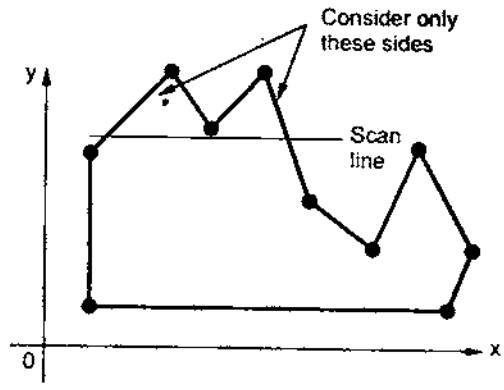


Fig. 3.13 Consider only the sides which intersect the scan line

It will be easier to identify which polygon sides should be tested for x-intersection, if we first sort the sides in order of their maximum y value. Once the sides are sorted we can process the scan lines from the top of the polygon to its bottom producing an active edge list for each scan line crossing the polygon boundaries. The active edge list for a scan line

contains all edges crossed by that scan line. The Fig. 3.14 shows sorted edges of the polygon with active edges.

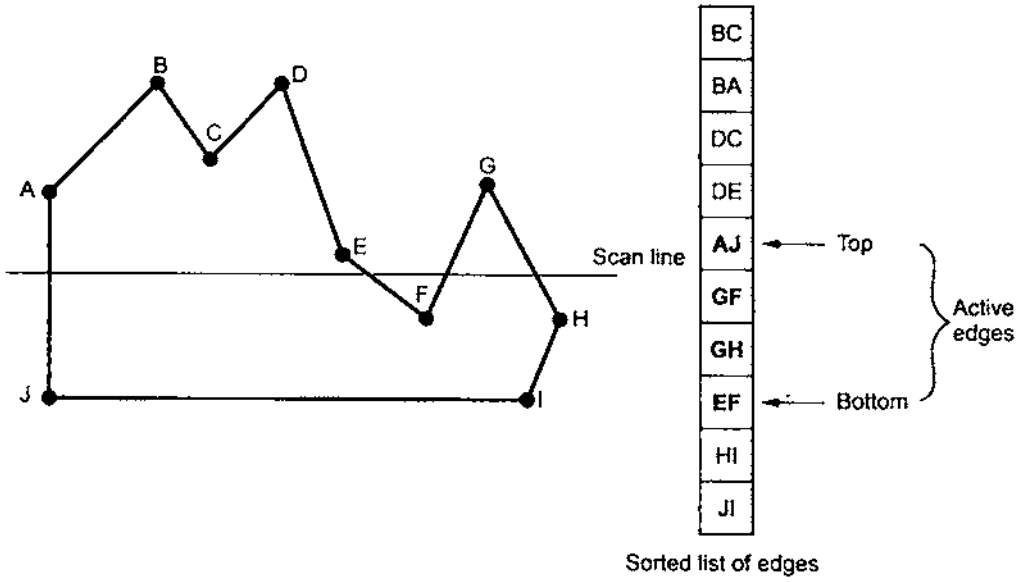


Fig. 3.14 Sorted edges of the polygon with active edges

In summary, a scan line algorithm for filling a polygon begins by ordering the polygon sides on the largest y value. It begins with the largest y value and scans down the polygon. For each y, it determines which sides can be intersected and finds the x values of these intersection points. It then sorts, pairs and passes these x values to a line drawing routine.

Scan Line Conversion Algorithm for Polygon Filling :

1. Read n, the number of vertices of polygon
2. Read x and y coordinates of all vertices in array x[n] and y[n].
3. Find y_{min} and y_{max}.
4. Store the initial x value (x_1) y values y_1 and y_2 for two endpoints and x increment Δx from scan line to scan line for each edge in the array edges [n] [4].

   While doing this check that $y_1 > y_2$, if not interchange y_1 and y_2 and corresponding x_1 and x_2 so that for each edge, y_1 represents its maximum y coordinate and y_2 represents it minimum y coordinate.
5. Sort the rows of array, edges [n] [4] in descending order of y_1, descending order of y_2 and ascending order of x_2.
6. Set $y = y_{max}$
7. Find the active edges and update active edge list :

   if ($y > y_2$ and $y \le y_1$)

   { edge is active }

   else

{ edge is not active }

8. Compute the x intersects for all active edges for current y value [initially x-intersect is x_1 and x intersects for successive y values can be given as

   $x_{i+1} \leftarrow x_i + \Delta x$

   where $\Delta x = -\frac{1}{m}$ and $m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e. slope of a line segment

9. If x intersect is vertex i.e. x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x-intersect [ ] array.
10. Sort x-intersect [ ] array in the ascending order,
11. Extract pairs of intersects from the sorted x-intersect [ ] array.
12. Pass pairs of x values to line drawing routine to draw corresponding line segments
13. Set y = y – 1
14. Repeat steps 7 through 13 until $y \geq y_{min}$.
15. Stop

In step 7, we have checked for $y \leq y_1$ and not simply $y < y_1$. Hence step 9 a becomes redundant. Following program takes care of that.

'C' code for Scan line Algorithm for Filling Polygon

(Softcopy of this program is available at vtubooks.com)

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

/* Defining the structure to store edges
----------------------------------------*/
struct edge
{
int x1;
int y1;
int x2;
int y2;
int flag;
};

void main()
{
int gd = DETECT, gm, n, i, j, k;
```

```
struct edge ed[10],temped;
float dx,dy,m[10],x_int[10],inter_x[10];
int x[10],y[10],ymax = 0, ymin = 480, yy,temp;
initgraph (&gd, &gm, " ");

/* Read the number of vertices of the polygon
---------------------------------------------- */
printf("Enter the number vertices of the graph: ");
scanf("%d", &n);

/* Read the vertices of the polygon and also find Ymax and Ymin
------------------------------------------------------------- */
printf("Enter the vertices: \n");
for(i = 0; i < n; i++)
{
printf("x[%d] : ", i);
scanf("%d", &x[i]);
printf("y[%d] : ", i);
scanf("%d", &y[i]);
if(y[i] > ymax)
ymax = y[i];
if(y[i] < ymin)
ymin = y[i];
ed[i].x1 = x[i];
ed[i].y1 = y[i];
}

/* Store the edge information
-----------------------------*/
for(i=0;i<n-1;i++)
{
ed[i].x2 = ed[i+1].x1;
ed[i].y2 = ed[i+1].y1;
ed[i].flag=0;
}
ed[i].x2 = ed[0].x1;
ed[i].y2 = ed[0].y1;
ed[i].flag=0;
```

```
/* Check for y1>y2, if not interchange y1 and y2
   with corresponding x1 and x2 ---------------*/
for(i=0;i<n;i++)
{
  if(ed[i].y1 < ed[i].y2)
  {
  temp = ed[i].x1;
  ed[i].x1=ed[i].x2;
  ed[i].x2=temp;
  temp = ed[i].y1;
  ed[i].y1=ed[i].y2;
  ed[i].y2=temp;
  }
}


/* Draw the polygon
---------------------- */
for(i=0;i<n;i++)
{
line(ed[i].x1,ed[i].y1,ed[i].x2,ed[i].y2);
}
/* sorting of edges in the order of y1,y2,x1
------------------------------------------------- */
  for(i=0;i<n-1;i++)
  {
     for(j=0;j<n-1;j++)
     {
          if(ed[j].y1<ed[j+1].y1)
          {
          temped = ed[j];
          ed[j]=ed[j+1];
          ed[j+1] = temped;
          }
          if(ed[j].y1==ed[j+1].y1)
          {
             if(ed[j].y2<ed[j+1].y2)
```

```
                {
                temped = ed[j];
                ed[j]=ed[j+1];
                ed[j+1] = temped;
                }
                   if(ed[j].y2==ed[j+1].y2)
                {
                       if(ed[j].x1<ed[j+1].x1)
                       {
                              temped = ed[j];
                              ed[j]=ed[j+1];
                              ed[j+1] = temped;
                       }
                }
            }
        }
    }
/* calculating 1/slope of each edge  and storing top x
coordinate of the edge     -------------------------- */
for(i=0;i<n;i++)
{
dx = ed[i].x2 - ed[i].x1;
dy = ed[i].y2 - ed[i].y1;
if(dy==0)
{
m[i]=0;
}
else
{
m[i] = dx/dy;
}
inter_x[i]= ed[i].x1;
}

yy=ymax;
while(yy>ymin)
{
```

```
/*  Marking active egdes
------------------------- */
        for(i=0;i<n;i++)
        {
        if(yy > ed[i].y2 && yy <= ed[i].y1)
         {
         ed[i].flag = 1;
         }
         else
         {
         ed[i].flag = 0;
         }
        }

/* Finding the x intersections
------------------------------  */
       j=0;
       for(i=0;i<n;i++)
    {
       if(ed[i].flag==1)
       {
         if(yy==ed[i].y1)
        {
         x_int[j]=ed[i].x1;
         j++;
             *if(ed[i-1].y1==yy && ed[i-1].y1<yy)
             {
              x_int[j]=ed[i].x1;
              j++;
             }
             if(ed[i+1].y1==yy && ed[i+1].y1<yy)
             {
              x_int[j]=ed[i].x1;
              j++;
```

```
                }*
            }
        else
        {
        x_int[j] = inter_x[i]+(-m[i]);
        inter_x[i]=x_int[j];
        j++;
        }
        }
    }

/* Sorting the x intersections
--------------------------------*/
        for(i=0;i<j;i++)
        {
          for(k=0;k<j-1;k++)
          {
           if(x_int[k]>x_int[k+1])
           {
            temp =x_int[k];
            x_int[k] = x_int[k+1];
            x_int[k+1] = temp;
           }
          }
        }

/* Extracting pairs of x values to draw lines
---------------------------------------------- */
        for(i=0;i<j;i+=2)
        {
        line(x_int[i],yy,x_int[i+1],yy);
        }
    yy--;
    delay(50);
}
getch();
}
```

Solved Example

Ex 3.1 : *Write a 'C' program to generate figure below. Colour the areas as shown. Assume colour codes are as follows : Background - 0, Foreground - 4, Grey - 2.*

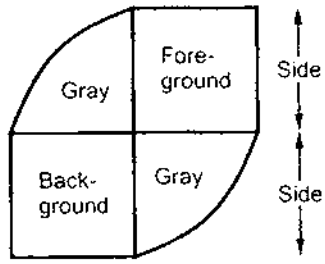


Fig. 3.15

(May-99)

Sol. : The 'C' code for above program is as given below

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
float  sum=0,d0,angle,theta,x,y,xc,yc;
int i,gd,gm;

/* initialise graphics
------------------------ */
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");
rectangle(320,190,370,240);
rectangle(270,240,320,290);
setfillstyle(1,4);
floodfill(340,210,15);
setfillstyle(1,0);
floodfill(290,270,15);
xc=320;
yc=240;
x=370;
y=240;
```

```
d0 = 1/(3.2 * (abs(x-xc) + abs(y-yc)));
angle = 0;
theta = 90;
      for(i=0;i<2;i++)
      {
      while (angle*180/3.142 < theta )
            {
             setcolor(2);
             putpixel(x,y,15);
             x = x - (y-yc) * d0;
             y = y + (x - xc)*d0;
             angle = angle + d0;
             delay(100);
             }
      angle = (180*3.142/180);
      theta = 270;
      x = 270;
      y = 240;
      }
setfillstyle(1,2);
floodfill(310,230,15);
floodfill(350,250,15);
getch();
closegraph();
}
```

Review Questions

1. What is polygon ? Explain different types of polygons.
2. Explain various approaches used to represent polygon.
3. What is a display file ?
4. Explain the polygon entering algorithm in the display file.
5. Explain the boundary fill alogrithm in detail.
6. Explain the steps required to fill the polygon using flood fill technique.
7. Explain the steps in the scan line algorithm for filling the polygon.
8. What is the need of y-bucket and active edge list?

University Questions

1. What do you understand by solid area scan converting. Develop an algorithm to scan convert a polygon. Explain the working of your algorithm for the following picture :-

   The triangle PQR is cut out from the rectangle ABCD. **(Dec-96, Dec-2000)**

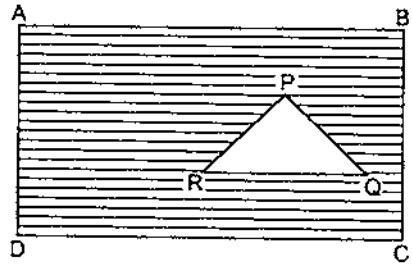


Fig. 3.16

[Hint : Draw rectangle and fill it with the background colour then draw triangle and use scan line algorithm to fill it with desired colour]

2. Write a program for the algorithm discussed above. **(Dec-96)**
3. Discuss various methods of 2D area filling and state the relative merits and demerits of each. **(May-97, Dec-97)**
4. Develop a program to scan convert a polygon. **(May-97)**
5. Develop a program to scan convert a polygon. Comment on the behaviour of your program if there are two intersecting polygons. **(Dec-97)**
6. Write a detailed note on boundary fill and flood fill algorithm. **(May-98)**
7. Discuss the scan line area fill method, how it will work for following picture ?

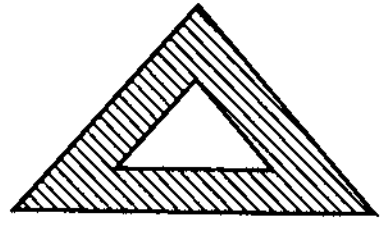


Fig. 3.17

(Dec-98)

8. Write a 'C' program to generate figure below. Colour the areas as shown. Assume colour codes are as follows : Background - 0, Foreground - 4, Grey - 2. **(May-99)**

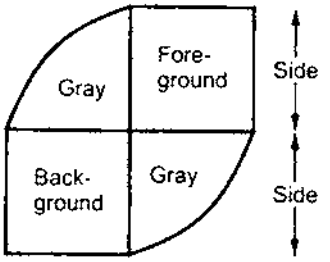


Fig. 3.18

9. Write a short note on scan line filling algorithm. (May-99)
10. What is the meaning of scan conversion of point ? (Dec-99)
11 Explain the scan-line algorithm for area filling of polygonal areas. (Dec-99)
12. Discuss any scan line area filling algorithm. Demonstrate how it would behave for the following shaded area : (May-2000)

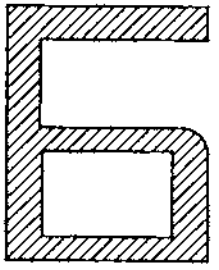


Fig. 3.19

13. Explain pattern filling algorithm with example. (May-2001)
14. Write a short note on scan line fill algorithm. (Dec-2001)
15. Write a short note on Boundary-fill algorithm. (May-2003)

❑❑❑

4

2-D Geometric Transformation

4.1 Introduction

Almost all graphics systems allow the programmer to define picture that include a variety of transformations. For example, the programmer is able to magnify a picture so that detail appears more clearly, or reduce it so that more of the picture is visible. The programmer is also able to rotate the picture so that he can see it in different angles.

In this chapter we discuss the 2D transformations.

4.2 Two Dimensional Transformations

In this section, we describe the general procedures for applying translation, rotation, and scaling parameters to reposition and resize the two dimensional objects.

4.2.1 Translation

Translation is a process of changing the position of an object in a straight-line path from one coordinate location to another. We can translate a two dimensional point by adding translation distances, t_x and t_y, to the original coordinate position (x, y) to move the point to a new position (x', y'), as shown in the Fig. 4.1.

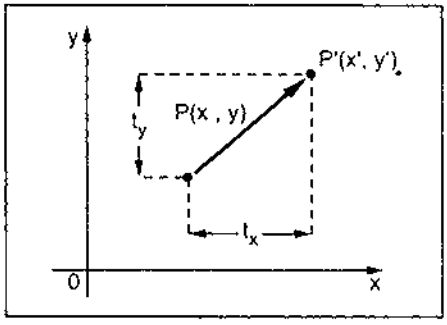


Fig. 4.1

$$x' = x + t_x \qquad \ldots (4.1)$$

$$y' = y + t_y \qquad \ldots (4.2)$$

The translation distance pair (t_x, t_y) is called a **translation vector** or **shift vector**.

It is possible to express the translation equations 4.1 and 4.2 as a single matrix equation by using column vectors to represent coordinate positions and the translation vector :

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

This allows us to write the two dimensional translation equations in the matrix form :

$$P' = P + T \qquad \ldots (4.3)$$

Ex. 4.1 : *Translate a polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by 3 units in x direction and 4 units in y direction.*

Sol. :

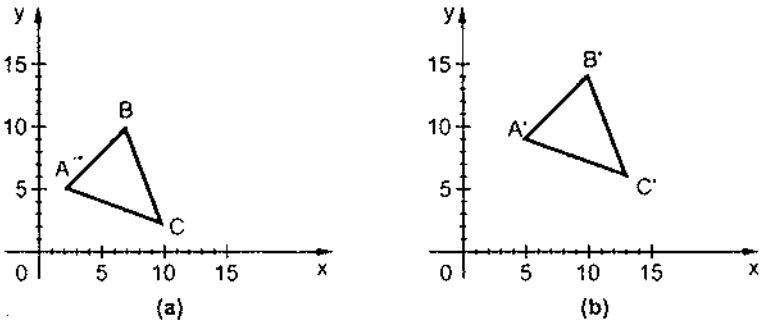


Fig. 4.2 Translation of polygon

$$A' = A + T$$

$$= \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$B' = B + T$$

$$= \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$C' = C + T$$

$$= \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

4.2.2 Rotation

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

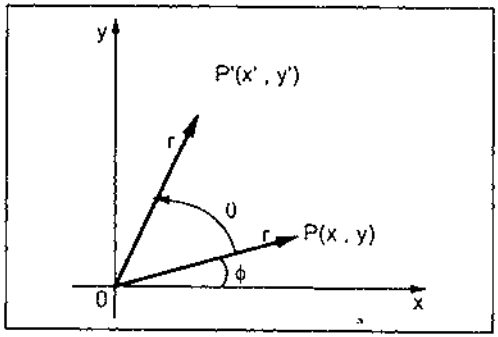


Fig. 4.3

Let us consider the rotation of the object about the origin, as shown in the Fig. 4.3.

Here, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle. Using standard trigonometric equations, we can express the transformed coordinates in terms of angles θ and ϕ as

$$\left.\begin{aligned} x' &= r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \\ y' &= r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta \end{aligned}\right\} \quad \ldots(4.4)$$

The original coordinates of the point in polar coordinates are given as

$$\left.\begin{aligned} x &= r\cos\phi \\ y &= r\sin\phi \end{aligned}\right\} \quad \ldots(4.5)$$

Substituting equations 4.5 into 4.4, we get the transformation equations for rotating a point (x, y) through an angle θ about the origin :

$$\left.\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}\right\} \quad \ldots(4.6)$$

The above equations can be represented in the matrix form as given below

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

$$\therefore \qquad P' = P \cdot R \quad \ldots(4.7)$$

where R is rotation matrix and it is given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \quad \ldots(4.8)$$

It is important to note that positive values for the rotation angle define counterclockwise rotations about the rotation point and negative values rotate objects in the clockwise sense.

For negative values of θ i.e., for clockwise rotation, the rotation matrix becomes

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \quad \because \cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta \quad \ldots(4.9)$$

Ex. 4.2 : *A point (4, 3) is rotated counterclockwise by an angle of 45°. Find the rotation matrix and the resultant point.*

Sol. :

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ \\ -\sin 45^\circ & \cos 45^\circ \end{bmatrix}$$

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

$$\therefore \quad P' = [4 \;\; 3] \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$= \left[4/\sqrt{2} - 3/\sqrt{2} \quad 4/\sqrt{2} + 3/\sqrt{2} \right]$$

$$= \left[1/\sqrt{2} \quad 7/\sqrt{2} \right]$$

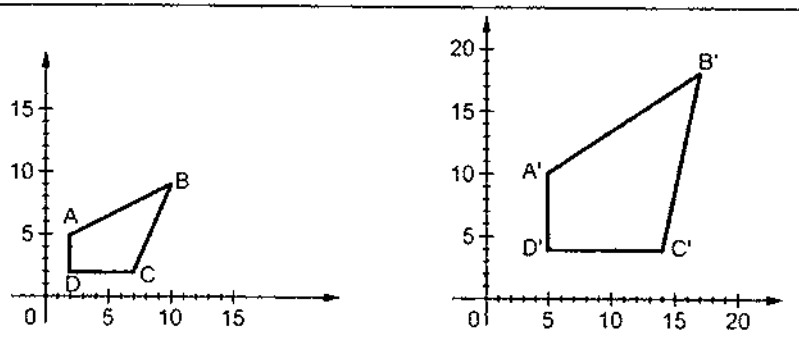
4.2.3 Scaling

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

$$x' = x \cdot S_x$$

$$\text{and} \quad y' = y \cdot S_y \qquad \ldots (4.10)$$

Scaling factor S_x scales object in the x direction and scaling factor S_y scales object in the y direction. The equations 4.10 can be written in the matrix form as given below :



(a) (b)

Fig. 4.4

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

$$= [x \cdot S_x \quad y \cdot Sy] \qquad \ldots (4.11)$$

$$= P \cdot S$$

Any positive numeric values are valid for scaling factors S_x and S_y. Values less than 1 reduce the size of the objects and values greater than 1 produce an enlarged object. For both S_x and S_y values equal to 1, the size of object does not change. To get uniform scaling it is necessary to assign same value for S_x and S_y. Unequal values for S_x and S_y result in a differential scaling.

Ex. 4.3 : *Scale the polygon with coordinates A (2, 5), B (7, 10) and C (10, 2) by two units in x direction and two units in y direction.*

Sol. : Here $S_x = 2$ and $S_y = 2$. Therefore, transformation matrix is given as

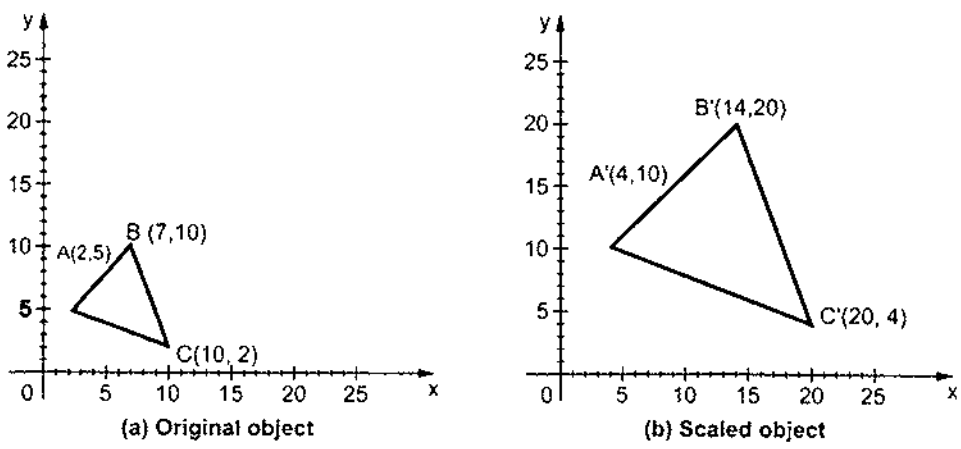
$$S = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

The object matrix is :

$$\begin{array}{c} \\ A \\ B \\ C \end{array} \begin{array}{c} \begin{array}{cc} x & y \end{array} \\ \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \end{array}$$

$$\therefore \quad \begin{array}{c} A' \\ B' \\ C' \end{array} \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 10 \\ 14 & 20 \\ 20 & 4 \end{bmatrix}$$



(a) Original object (b) Scaled object

Fig. 4.5

4.3 Homogeneous Coordinates

In design and picture formation process, many times we may require to perform translation, rotations, and scaling to fit the picture components into their proper positions. In the previous section we have seen that each of the basic transformations can be expressed in the general matrix form

$$P' = P \cdot M_1 + M_2 \qquad \ldots (4.12)$$

For translation :

$$P' = P \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

i.e. M_1 = Identity matrix

M_2 = Translation vector

For rotation :

$$P' = P \cdot \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e. M_1 = Rotational matrix

$M_2 = 0$

For scaling :

$$P' = P \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

i.e. M_1 = Scaling matrix

$M_2 = 0$

To produce a sequence of transformations with above equations, such as translation followed by rotation and then scaling, we must calculate the transformed coordinates one step at a time. First, coordinates are translated, then these translated coordinates are scaled, and finally, the scaled coordinates are rotated. But this sequential transformation process is not efficient. A more efficient approach is to combine sequence of transformations into one transformation so that the final coordinate positions are obtained directly from initial coordinates. This eliminates the calculation of intermediate coordinate values.

In order to combine sequence of transformations we have to eliminate the matrix addition associated with the translation terms in M_2 (Refer equation 4.12). To achieve this we have to represent matrix M_1 as 3×3 matrix instead of 2×2 introducing an additional dummy coordinate W. Here, points are specified by three numbers instead of two. This coordinate system is called **homogeneous coordinate system** and it allows us to express all transformation equations as matrix multiplication.

The homogeneous coordinate is represented by a triplet (X_W, Y_W, W),

where

$$x = \frac{X_W}{W} \quad \text{and} \quad y = \frac{Y_W}{W}$$

For two dimensional transformations, we can have the homogeneous parameter W to be any non zero value. But it is convenient to have W = 1. Therefore, each two dimensional position can be represented with homogeneous coordinate as (x, y, 1).

Summaring it all up, we can say that the homogeneous coordinates allow combined transformation, eliminating the calculation of intermediate coordinate values and thus save required time for transformation and memory required to store the intermediate coordinate values. Let us see the homogeneous coordinates for three basic transformations.

4.3.1 Homogeneous Coordinates for Translation

The homogeneous coordinates for translation are given as

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \qquad \ldots (4.13)$$

Therefore, we have

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

$$= [x + t_x \;\; y + t_y \;\; 1] \qquad \ldots (4.14)$$

4.3.2 Homogeneous Coordinates for Rotation

The homogeneous coordinates for rotation are given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \ldots (4.15)$$

Therefore, we have

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

$$= [x\cos\theta - y\sin\theta \quad x\sin\theta + y\cos\theta \quad 1] \qquad \ldots (4.16)$$

4.3.3 Homogeneous Coordinates for Scaling

The homogeneous coordinate for scaling are given as

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

Therefore, we have

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

$$= [x \cdot S_x \quad y \cdot S_y \quad 1] \qquad \ldots (4.17)$$

Note : In this text, the object matrix is written first and it is then multipled by the required transformation matrix. If we wish to write the transformation matrix first and then the object matrix we have to take the transpose of both the matrices and post-multiply the object matrix i.e.,

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

Ex. 4.4 : *Give a 3 × 3 homogeneous coordinate transformation matrix for each of the following translations*

a) Shift the image to the right 3-units

b) Shift the image up 2 units

c) Move the image down ½ unit and right 1 unit

d) Move the image down 2/3 unit and left 4 units

Sol. : We know that homogenous coordinates for translation are

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

a) Here, $t_x = 3$ and $t_y = 0$

$$\therefore \qquad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

b) Here, $t_x = 0$ and $t_y = 2$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix}$$

c) Here, $t_x = 1$ and $t_y = -0.5$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -0.5 & 1 \end{bmatrix}$$

d) Here, $t_x = -4$ and $t_y = -0.66$

$$\therefore \quad T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -4 & -0.66 & 1 \end{bmatrix}$$

Ex. 4.5 : *Find the transformation matrix that transforms the given square ABCD to half its size with centre still remaining at the same position. The coordinates of the square are : A(1, 1), B (3, 1), C (3, 3), D (1, 3) and centre at (2, 2). Also find the resultant coordinates of square.*

Sol. : This transformation can be carried out in the following steps.

1. Translate the square so that its center coincides with the origin.
2. Scale the square with respect to the origin.
3. Translate the square back to the original position.

Thus, the overall transformation matrix is formed by multiplication of three matrices.

$$\therefore \quad T_1 \cdot S \cdot T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 & 1.5 & 1 \\ 2.5 & 1.5 & 1 \\ 2.5 & 2.5 & 1 \\ 1.5 & 2.5 & 1 \end{bmatrix}$$

Ex. 4.6 : *Find a transformation of triangle A (1, 0), B (0, 1), C (1, 1) by*

a) Rotating 45° about the origin and then translating one unit in x and y direction.

b) Translating one unit in x and y direction and then rotating 45° about the origin.

Sol. : The rotation matrix is

$$R = \begin{bmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and}$$

The translation matrix is

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

a)
$$R \cdot T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

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

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}}+1 & \frac{1}{\sqrt{2}}+1 & 1 \\ -\frac{1}{\sqrt{2}}+1 & \frac{1}{\sqrt{2}}+1 & 1 \\ 1 & \sqrt{2}+1 & 1 \end{bmatrix}$$

b)
$$T \cdot R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 3/\sqrt{2} & 1 \\ -1/\sqrt{2} & 3/\sqrt{2} & 1 \\ 0 & 2\sqrt{2} & 1 \end{bmatrix}$$

In the above example, the resultant coordinates of a triangle calculated in part (a) and (b) are not same. This shows that the order in which the transformations are applied is important in the formation of combined or concatenated or composed transformations.

4.4 Composition of 2D Transformations

We have seen what is meant by combined or concatenated or composed transformations in the previous section. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.

4.4.1 Rotation About an Arbitrary Point

To rotate an object about an arbitrary point, (x_p, y_p) we have to carry out three steps :

1. Translate point (x_p, y_p) to the origin
2. Rotate it about the origin and
3. Finally, translate the center of rotation back where it belongs (See Fig.4.6)

We have already seen that matrix multiplication is not commutative, i.e. multiplying matrix A by matrix B will not always yield the same result as multiplying matrix B by matrix A. Therefore, in obtaining composite transformation matrix, we must be careful to order the matrices so that they correspond to the order of the transformations on the object. Let us find the transformation matrices to carry out individual steps.

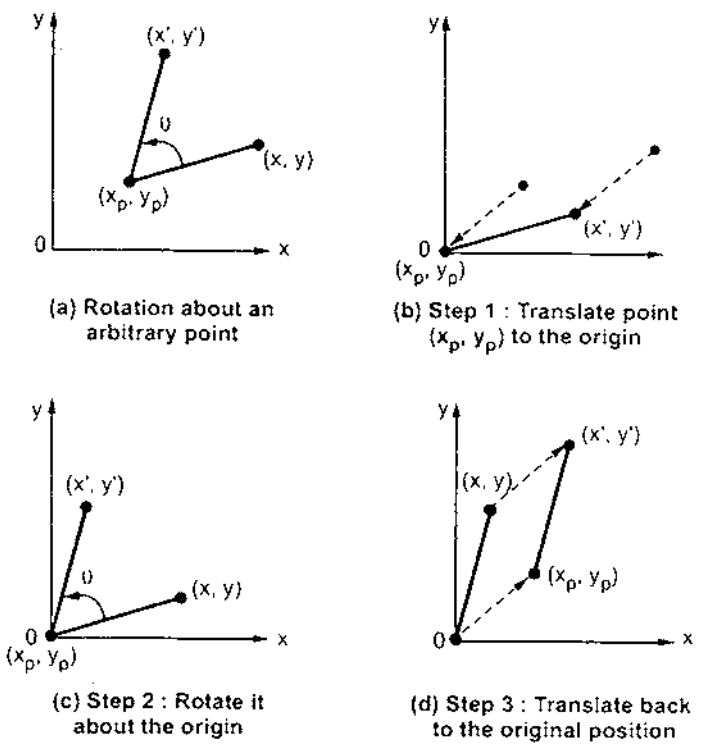


Fig. 4.6

The translation matrix to move point (x_p, y_p) to the origin is given as

$$T_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix}$$

The rotation matrix for counterclockwise rotation of point about the origin is given as

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

The translation matrix to move the center point back to its original position is given as

$$T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

Therefore, the overall transformation matrix for a counterclockwise rotation by an angle θ about the point (x_p, y_p) is given as

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p\cos\theta + y_p\sin\theta + x_p & -x_p\sin\theta - y_p\cos\theta + y_p & 1 \end{bmatrix} \quad \ldots (4.18)$$

Ex. 4.7 : *Perform a counterclockwise 45° rotation of triangle A (2, 3), B (5, 5), C (4, 3) about point (1, 1).*

Sol. : From equation 4.18 we have

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p\cos\theta + y_p\sin\theta + x_p & -x_p\sin\theta - y_p\cos\theta + y_p & 1 \end{bmatrix}$$

Here, $\theta = 45°$, $x_p = 1$ and $y_p = 1$. Substituting values we get

$$T_1 \cdot R \cdot T_2 = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 \\ 5 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -\frac{1}{\sqrt{2}}+1 & \frac{3}{\sqrt{2}}+1 & 1 \\ 1 & \frac{8}{\sqrt{2}}+1 & 1 \\ \frac{1}{\sqrt{2}}+1 & \frac{5}{\sqrt{2}}+1 & 1 \end{bmatrix}$$

4.5 Other Transformations

The three basic transformations of scaling, rotating, and translating are the most useful and most common. There are some other transformations which are useful in certain applications. Two such transformations are reflection and shear.

4.5.1 Reflection

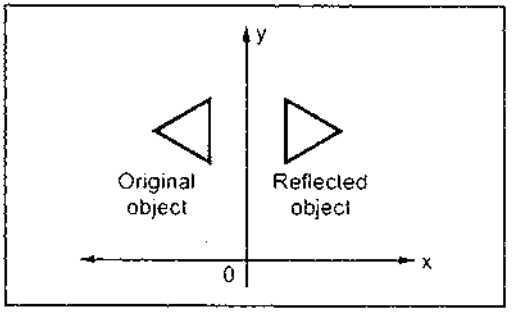


Fig. 4.7 Reflection about y axis

A reflection is a transformation that produces a mirror image of an object relative to an axis of reflection. We can choose an axis of reflection in the xy plane or perpendicular to the xy plane. The table 4.1 gives examples of some common reflections.

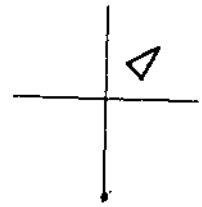
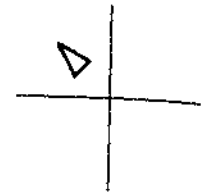
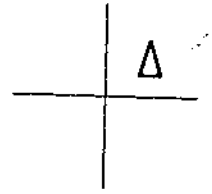
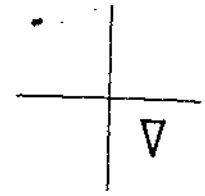
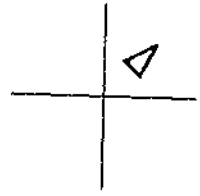
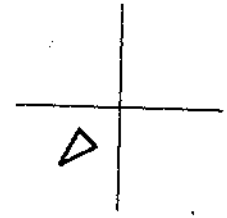
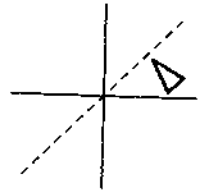
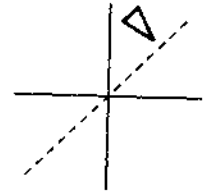
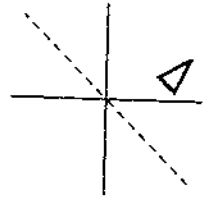
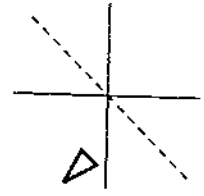
Reflection	Transformation matrix	Original image	Reflected image
Reflection about Y-axis	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about X axis	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about origin	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = x$	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
Reflection about line $y = -x$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		

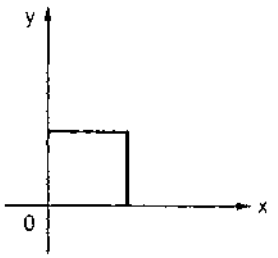
Table 4.1 Common reflections

4.5.2 Shear

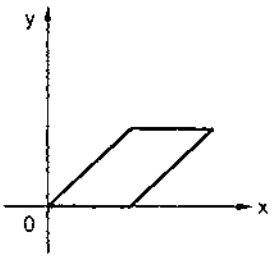
A transformation that slants the shape of an object is called the shear transformation. Two common shearing transformations are used. One shifts x coordinate values and other shifts y coordinate values. However, in both the cases only one coordinate (x or y) changes its coordinates and other preserves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

4.5.2.1 X shear

The x shear preserves the y coordinates, but changes the x values which causes vertical lines to tilt right or left as shown in the Fig. 4.8. The transformation matrix for x shear is given as



(a) Original object



(b) Object after x shear

Fig. 4.8

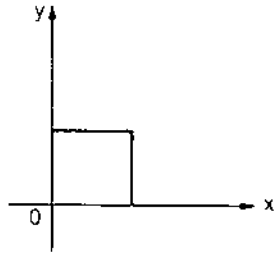
$$X_sh = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad x' = x + Sh_x \cdot y \quad \text{and}$$

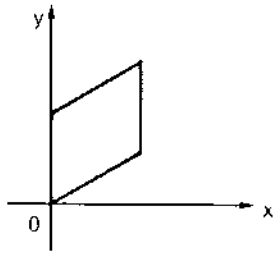
$$y' = y \qquad \ldots (4.19)$$

4.5.2.2 Y shear

The y shear preserves the x coordinates, but changes the y values which causes horizontal lines to transform into lines which slope up or down, as shown in the Fig. 4.9.



(a) Original object



(b) Object after y shear

Fig. 4.9

The transformation matrix for y shear is given as

$$Y_sh = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad x' = x \text{ and } y' = y + Sh \cdot x \qquad \ldots (4.20)$$

4.5.2.3 Shearing Relative to Other Reference Line

We can apply x shear and y shear transformations relative to other reference lines. In x shear transformation we can use y reference line and in y shear we can use x reference line. The transformation matrices for both are given below :

x shear with y reference line : $\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ -Sh_x \cdot y_{ref} & 0 & 1 \end{bmatrix}$

y shear with x reference line : $\begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & -Sh_y \cdot x_{ref} & 0 \end{bmatrix}$

Ex. 4.8 : *Apply the shearing transformation to square with A(0, 0), B (1, 0), C (1, 1) and D (0, 1) as given below*

a) Shear parameter value of 0.5 relative to the line $y_{ref} = -1$

b) Shear parameter value of 0.5 relative to the line $x_{ref} = -1$

Sol. : a) Here $Sh_x = 0.5$ and $y_{ref} = -1$

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ -Sh_x \cdot y_{ref} & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 & 1 \\ 1.5 & 0 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

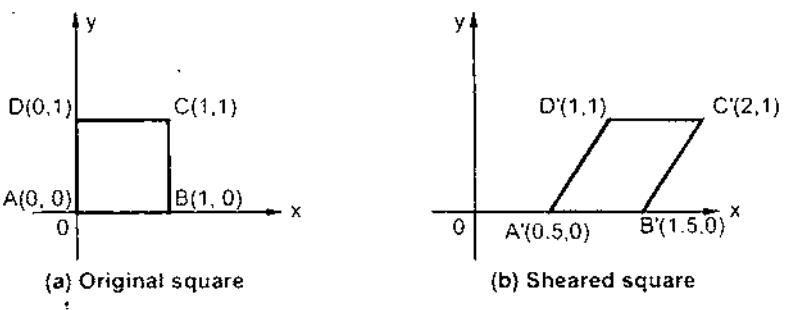


Fig. 4.10

b) Here $Sh_y = 0.5$ and $x_{ref} = -1$

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & -Sh_y \cdot x_{ref} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 1.5 & 1 \end{bmatrix}$$

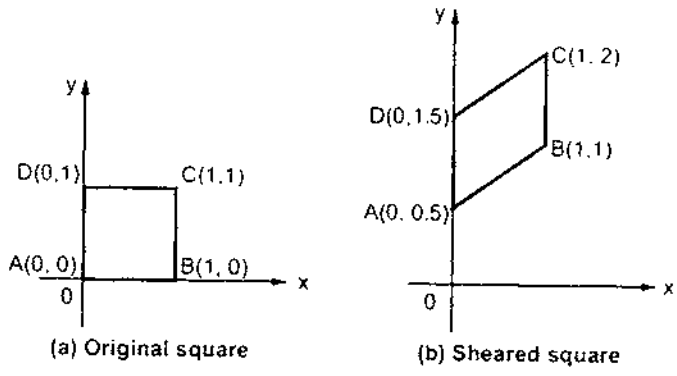


Fig. 4.11

It is important to note that shearing operations can be expressed as sequence of basic transformations. The sequence of basic transformations involve series of rotation and scaling transformations.

Ex. 4.9 : *Show how shear transformation may be expressed in terms of rotation and scaling.*

Sol. : The shear transformation matrix for x and y combinely can be given as

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

We have scaling matrix and rotation matrix as given below

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

If we combine scale matrix and rotation matrix we have,

$$S \cdot R \cdot = \begin{bmatrix} S_x \cos\theta & S_x \sin\theta & 0 \\ -S_y \sin\theta & S_y \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Comparing shear matrix and $S \cdot R \cdot$ matrix we have

$$Sh_x = -S_y \sin\theta$$

$$Sh_y = S_x \sin\theta$$

$$S_x \cos\theta = 1 \qquad \text{and}$$

$$S_y \cos\theta = 1$$

$$\therefore \qquad S_x = \frac{1}{\cos\theta} \qquad \text{and}$$

$$S_y = \frac{1}{\cos\theta}$$

Substituting values of S_x and S_y we get,

$$Sh_x = -\frac{1}{\cos\theta} \cdot \sin\theta = -\tan\theta$$

$$Sh_y = \frac{1}{\cos\theta} \cdot \sin\theta = \tan\theta$$

Therefore, the shear transformation matrix expressed in terms of rotation and scales is

$$\begin{bmatrix} 1 & \tan\theta & 0 \\ -\tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \because S_x \cos\theta = S_y \cos\theta = 1$$

where θ : angle of rotation

S_x : x scale and

S_y : y scale

4.6 Inverse Transformations

When we apply any transformation to point (x, y) we get a new point (x′, y′). Sometimes it may require to undo the applied transformation. In such a case we have to get original point (x, y) from the point (x′, y′). This can be achieved by inverse transformation. The inverse transformation uses the matrix inverse of the transformation matrix to get the original point (x, y). The inverse of a matrix is another matrix such that when the two are multiplied together, we get the identity matrix.

If the inverse of matrix T is T^{-1}, then

$$T\,T^{-1} = T^{-1}T = I \qquad \ldots (4.21)$$

where I is the identity matrix with all elements along the major diagonal having value 1, and all other elements having value zero.

The elements for the inverse matrix T^{-1} can be calculated from the elements of T as

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T} \qquad \ldots (4.22)$$

where t_{ij}^{-1} is the element in the ith row and j^{th} column of T^{-1}, and M_{ji} is the (n – 1) by (n – 1) submatrix obtained by deleting the jth row and i^{th} column of the matrix A. The det M_{ji} and det T is the determinant of the M_{ji} and T matrices.

The determinant of a 2 × 2 matrix is

$$\det \begin{vmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{vmatrix} = t_{11} \cdot t_{22} - t_{12} \cdot t_{21} \qquad \ldots (4.23)$$

The determinant of a 3 × 3 matrix is

$$\det T = t_{11} \cdot (t_{22}\, t_{33} - t_{23}\, t_{32}) - t_{12} \cdot (t_{21}\, t_{33} - t_{23}\, t_{31}) + t_{13} \cdot (t_{21}\, t_{32} - t_{22}\, t_{31}) \qquad \ldots (4.24)$$

In general form, the determinant of T is given by

$$\det T_i = \sum t_{ij}\,(-1)^{i+j} \det M_{ij} \qquad \ldots (4.25)$$

where M_{ij} is the submatrix formed by deleting row i and column j from matrix T.

The inverse of the homogeneous coordinate transformation matrix can be given as

$$\begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}^{-1} = \frac{1}{ae - bd} \begin{bmatrix} e & -d & 0 \\ -b & a & 0 \\ bf - ce & cd - af & ae - bd \end{bmatrix}$$

It is important to note that the elements of inverse matrix T^{-1} can be calculated from the element of T as

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T} \quad \ldots (4.26)$$

In the above equation the term det T is in the denominator. Hence, we can obtain an inverse matrix if and only if the determinant of the matrix is nonzero.

Solved Examples

Ex. 4.10 : *Find out the final coordinates of a figure bounded by the coordinates (1, 1), (3, 4), (5, 7), (10, 3) when rotated about a point (8, 8) by 30° in clockwise direction and scaled by two units in x-direction and three units y direction.*

Sol. : From following equation we have the transformation matrix for rotation about an arbitrary point given as

$$T_1 R\, T_2 = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta + x_p & -x_p \sin\theta - y_p \cos\theta + y_p & 1 \end{bmatrix}$$

In this case, it is clockwise rotation therefore we take value of θ negative.

$$\therefore T_1 . R . T_2 = \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} \cos(-30) & \sin(-30) & 0 \\ -\sin(-30) & \cos(-30) & 0 \\ -8\times\cos(-30) + 8\times\sin(-30) + 8 & -8\times\sin(-30) - 8\times\cos(-30) + 8 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ -2.928 & 5.072 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix}$$

$$\therefore \quad T_1 . R . T_2 S = \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -3.124 & 16.314 & 1 \\ 3.34 & 21.108 & 1 \\ 9.804 & 25.902 & 1 \\ 14.464 & 8.01 & 1 \end{bmatrix}$$

Ex. 4.11 : *Show that transformation matrix for a reflection about a line Y = X is equivalent to reflection to X- axis followed by counter - clockwise rotation of 90°.*

Sol. : The transformation matrix for reflection about a line Y = X is given as

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

The transformation matrix for reflection about x-axis and for counter clockwise rotation of 90 are given as

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \cos(90) & \sin(90) \\ -\sin(90) & \cos(90) \end{bmatrix}$$

Hence,

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

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

... Proved

Ex. 4.12 : *Find out final transformation matrix, when point P (x, y) is to be reflected about a line y = mx + C.*

Sol. : Equation of line :

$$y = mx + C$$

$$\text{slope} = m \qquad y \text{ intercept} = C$$

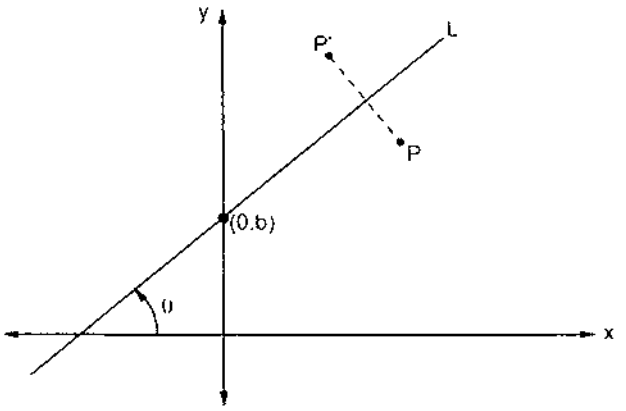


Fig. 4.12

We can relate slope m to angle θ by equation

$$m = \tan\theta$$

$$\therefore \quad \theta = \tan^{-1} m$$

where θ is in inclination of line with respect to x axis.

Translational matrix can be given as

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

Rotational matrix to match the given line with x axis can be obtained as

$$R_{\theta} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \{\text{Note : angle of rotation} = -\theta\}$$

Reflection matrix about x axis

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

Inverse transformation matrices,

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

$\therefore$ Final transformation matrix can be obtained as

$$R_T = T \cdot R_z \cdot M \cdot R_z^{-1} \cdot T^{-1}$$

As we have $\tan\theta = m$, using trigonometry we can obtain

$$\sin\theta = \frac{m}{\sqrt{m^2+1}} \quad \cos\theta = \frac{1}{\sqrt{m^2+1}}$$

$$\therefore \quad R_T = \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ -c\sin 2\theta & c(1+\cos 2\theta) & 1 \end{bmatrix}$$

By substituting values of $\sin\theta$ and $\cos\theta$ we have,

$$R_T = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & 0 \\ \frac{2m}{m^2+1} & m^2-1 & 0 \\ \frac{-2cm}{m^2+1} & \frac{2c}{m^2+1} & 1 \end{bmatrix}$$

Ex. 4.13 : *Derive the appropriate 2D transformation which reflects a figure in point (0.5, 0.5)*

Sol. : Translating given point to origin

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -0.5 & -0.5 & 1 \end{bmatrix}$$

Now obtaining reflection of the object about origin

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

Translating point back to original position.

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

The transformation can be given as

$$R_T = T \cdot M \cdot T^{-1}$$

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

Ex. 4.14 : *Find out the co-ordinates of a figure bounded by (0, 0) (1, 5) (6, 3) (– 3, – 4) when reflected along the line whose equation is y = 2x + 4 and sheared by 2 units in x direction and 2 units in y direction.*

Sol. : Equation of the line : y = 2x + 4

$\therefore$ slope = 2 and y intercept = 4

$\therefore$ $\theta = 63.43°$

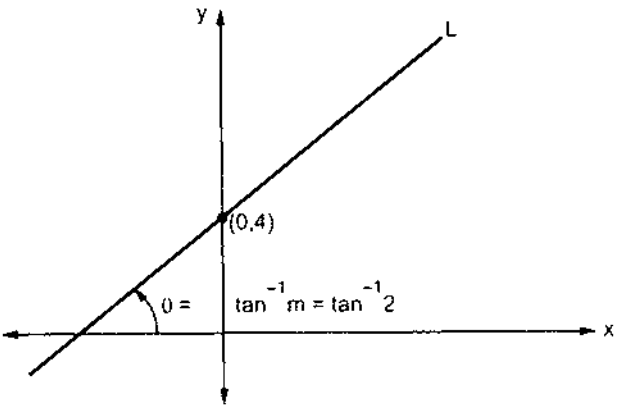


Fig. 4.13

Translational matrix can be given as

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{bmatrix}$$

For matching of given line with x axis we have

$$R_y = \begin{bmatrix} \cos(-63.43) & \sin(-63.43) & 0 \\ -\sin(-63.43) & \cos(-63.43) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} 0.4472 & -0.8944 & 0 \\ 0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For reflection about x axis we have

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

Inverse transformation matrices are

$$R_z^{-1} = \begin{bmatrix} \cos(-63.43) & -\sin(63.43) & 0 \\ +\sin(-63.43) & \cos(-63.43) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4472 & 0.8944 & 0 \\ -0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{bmatrix}$$

For shearing along x axis :

$$S_x = \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For shearing along y axis

$$S_y = \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The resultant transformation matrix can be obtained by

$$R_T = T \cdot R_y \cdot M \cdot R_y^{-1} \cdot T^{-1} \cdot S_x \cdot S_y$$

Final co-ordinates of the given figure can be obtained by

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \cdot R_T$$

Calculations are left for the students as an exercise

Ex. 4.15 : *Show that 2D reflection through X axis followed by 2-D reflection through the line Y = – X is equivalent to a pure rotation about the origin.*

Sol. : 2D reflection about X axis

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

2D reflection about Y = – X

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

∴ Resultant transformation matrix

$$R_T = R_x \cdot R'$$

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

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

For pure rotation about origin we have

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

where θ is angle of rotation

put $\theta = 90°$

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

$R_T = R_o$ Hence the result

Ex. 4.16 *Prove that successive 2D rotations are additive; i.e.*
$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$

Sol. : We can write rotation matrix $R(\theta_1)$ as

$$R(\theta_1) = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \text{and } R(\theta_2) = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

$$\therefore \quad R(\theta_1)\cdot R(\theta_2) = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \times \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix}$$

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

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

since,

$$\cos(\theta_1 + \theta_2) = \cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2$$
$$\sin(\theta_1 + \theta_2) = \cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2$$

Ex. 4.17 *Prove that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not.* **(Dec-99)**

Sol. : The matrix notation for scaling along S_x and S_y is as given below

$$S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \text{and}$$

The matrix notation for rotation is as given below

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

$$S\cdot R = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

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

$$= \begin{bmatrix} S_x\cos\theta & S_x\sin\theta \\ -S_x\sin\theta & S_x\cos\theta \end{bmatrix} \qquad \because S_x = S_y \quad \ldots I$$

or

$$= \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} \qquad \because \theta = n\pi \text{ where n is integer} \quad \ldots II$$

$$R\cdot S = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

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

$$= \begin{bmatrix} S_x\cos\theta & S_x\sin\theta \\ -S_x\sin\theta & S_x\cos\theta \end{bmatrix} \qquad \because S_x = S_y \ldots \text{III}$$

or

$$= \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} \qquad \because \theta = n\pi \text{ where n is integer} \ldots \text{IV}$$

From equations I and III, and equations II and IV it is proved that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not.

Ex. : 4.18 *A circular disc of diameter 'd' is rolling down the inclined plane starting from rest as shown below. Assume there is no slip and develop the set of transformation required to produce this animation and also write a program.* **(Dec-96)**

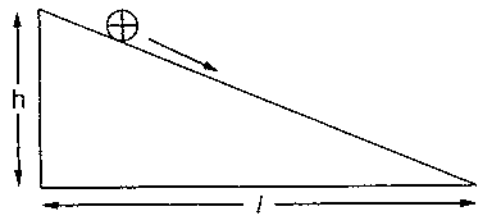


Fig. 4.14

Sol. : For rolling a circular disc of diameter d down the inclined plane starting from rest we have to consider the movement of disc in x direction and in y direction along with the rotation of disc. As length is greater then height we increment x by 1 unit and y by h/*l* units i.e.

$$x = x + 1$$

$$y = y + h/l$$

The increment of rolling angle can be calculated by relating of a circle with the diagonal length of inclined plane as given below

$$d\theta = \frac{\left(\sqrt{h^2 + l^2}/\pi d\right) \times 360}{l}$$

It is necessary to rotate two lines on the disc by $d\theta$ after increment of x and rotation is clockwise. The rotation matrix necessary for this purpose is

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

Every time rotated coordinates are calculated at the origin and at time of line drawing these coordinates are converted into actual coordinates by adding current x and y coordinates to it. Let us the listing of 'C' code for above animation.

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
int gd,gm;
double d, h,l,x,y, stx, sty,dx,dy,d0,sin0,cos0,angle=0,theta, x_offset,
    y_offset;
int i,j,k,ball[4][3]={1},ball1[4][3]={1};
float pi = 3.142;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/*  Read starting x,starting y, height,length, diameter
------------------------------------------------------ */
printf("Enter starting x coordinate :");
scanf("%lf",&x);
printf("Enter starting y coordinate :");
scanf("%lf",&y);
printf("Enter height :");
scanf("%lf",&h);
printf("Enter length :");
scanf("%lf",&l);
printf("Enter diameter :");
scanf("%lf",&d);

theta = atan(h/l);
y_offset = d/(2*cos(theta))-d/2*tan(theta);
x_offset = d/2;
circle(x+x_offset,y-y_offset,d/2);
line(x,y,x,y+h);
line(x,y+h,x+l,y+h);
```

```
line(x,y,x+l,y+h);
stx = x;
sty = y;
ball[0][0]= 0;
ball[0][1]= 0-d/2;
ball[1][0]= 0;
ball[1][1]= 0+d/2;
ball[2][0]= 0-d/2;
ball[2][1]= 0;
ball[3][0]= 0+d/2;
ball[3][1]= 0;
ball1[0][0]= 0;
ball1[0][1]= 0-d/2;
ball1[1][0]= 0;
ball1[1][1]= 0+d/2;
ball1[2][0]= 0-d/2;
ball1[2][1]= 0;
ball1[3][0]= 0+d/2;
ball1[3][1]= 0;

line(ball[0][0]+x+x_offset,ball[0][1]+y-y_offset,ball[1][0]+x+x_offset,
     ball[1][1]+y-y_offset);
line(ball[2][0]+x+x_offset,ball[2][1]+y-y_offset,ball[3][0]+x+x_offset,
     ball[3][1]+y-y_offset);
dy = h/l;
dx = l;
d0= (sqrt(h*h+l*l)/(pi*d)*360)/l;
i = 0;
do
{
setcolor(0);
circle(x+x_offset,y-y_offset,d/2);
line(ball1[0][0]+x+x_offset,ball1[0][1]+y-y_offset,ball1[1][0]+x+d/2,
     ball1[1][1]+y-y_offset);
line(ball1[2][0]+x+x_offset,ball1[2][1]+y-y_offset,ball1[3][0]+x+d/2,
     ball1[3][1]+y-y_offset);
sin0 = sin(angle*pi/180);
cos0 = cos(angle*pi/180);
```

```
    for(j=0;j<4;j++)
    {
      ball1[j][0] = ball[j][0]*cos0 + ball[j][1]*(-sin0);
      ball1[j][1] = ball[j][0]*sin0 + ball[j][1]*(cos0);
    }
setcolor(15);

angle=angle+d0;
x=x+dx;
y=y+dy;
line(stx,sty,stx+l,sty+h);
line(ball1[0][0]+x+x_offset,ball1[0][1]+y-y_offset,ball1[1][0]+x+x_offset
    ,ball1[1][1]+y-y_offset);
line(ball1[2][0]+x+x_offset,ball1[2][1]+y-y_offset,ball1[3][0]+x+x_offset
    ,ball1[3][1]+y-y_offset);
circle(x+x_offset,y-y_offset,d/2);
i=i+1;
delay(1000);
} while(i<l-d/2||y<h+sty-y_offset-d/2);
getch();
closegraph();
   }
```

Review Questions

1. Give the 2-D transformation matrix for
   a) Translation
   b) Rotation and
   c) Scaling
2. What is the need of homogeneous coordinates ? Give the homogeneous coordinates for translation, rotation and scaling.
3. What do you mean by composite transformation ? How it is useful ?
4. Derive the transformation matrix for rotation about arbitrary point.
5. Write a short note on
   a) Reflection
   b) Shearing transformation
6. Explain the inverse transformation. Derive the matrix for inverse transformation.

University Questions

1. Develop a 2D rotation and scaling transformation matrices with respect to a fixed point $P(x_p, y_p)$ (Dec-96)
2. A circular disc of diameter 'd' is rolling down the inclined plane starting from rest as shown below. Assume there is no slip and develop the set of transformation required to produce this animation. (Dec-96)

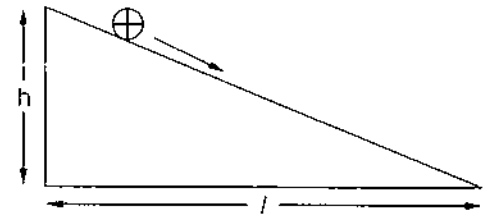


Fig. 4.15

3. What are the properties for concatenation of transformations ? What is the sequence of transformation required to change the position of object, shown in figure (a) to figure (b). (Dec-98)

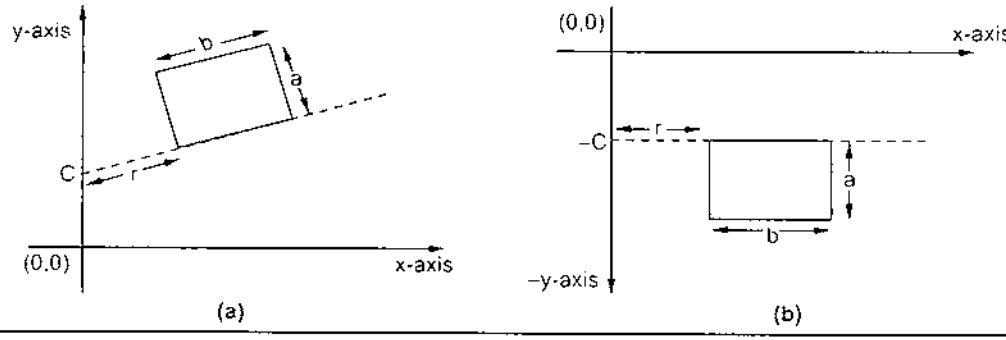


Fig. 4.16

4. What are homogeneous co-ordinates? What is the significance of this co-ordinate system ? (May-99)
5. Write a short note on shearing. (May-99)
6. For the following figure generate transformation matrix. (May-99)

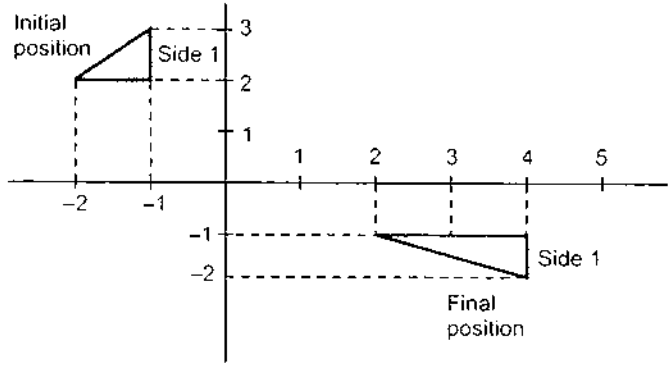


Fig. 4.17

$$\text{Ans}: T = \begin{bmatrix} 2 & 0 & 0 \\ 0 & -1 & 0 \\ 6 & 1 & 1 \end{bmatrix}$$

7. Why are matrices used or implementing transformations ? (May-99)
8. Develop the procedure to implement scaling as a raster transformation of a pixel block. (Dec-99)
9. Prove that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not. (Dec-99)
10. Proove that successive 2D rotations are additive; i.e.
    $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$ (Dec-2000)
11. Develop a 2D scaling transformation matrix with respect to a fixed point $P(x_F, y_F)$. (Dec-2000)
12. Explain the orthogonal property of rotational matrix. Using the orthogonal property of rotational matrix derive the transformation matrices required for transformation between co-ordinate system. (May-2001)

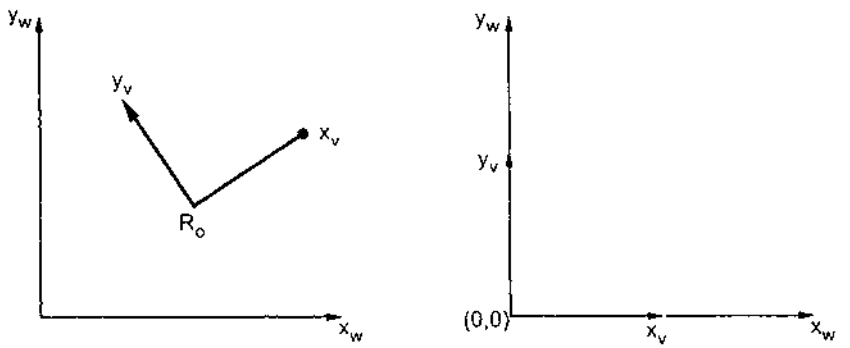


Fig. 4.18

13. Consider a wheel of diameter d rolling down on the inclined plane as shown in the Fig. 4.19 given below. Give the sequence of transformations required to perform this animation assuming that there is no slip. (Dec-2001)

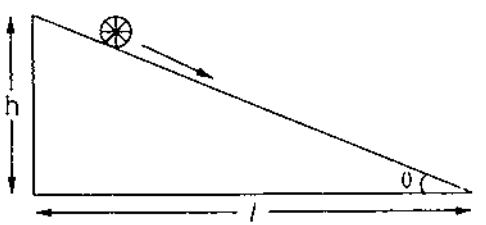


Fig. 4.19

14. Develop the transformation procedure to display the italic characters, given a vector font definition. That is all character shapes in this font are defined with straight line segments and italic characters are to be formed from this information with transformation operation. Specify the values of each parameter required for transformation operation. Illustrate the procedure by taking suitable example. **(Dec-2001)**
15. Write a short note on homogenous co-ordinates. **(May-2002)**
16. Show that two successive reflections about either of the coordinate axes is equivalent to a single rotation about the coordinate origin. **(May-2003)**

❑❑❑

5

2-D Viewing and Clipping

5.1 Introduction

Typically, a graphics package allows us to specify which part of a defined picture is to be displayed and where that part is to be displayed on the display device. Furthermore, the package also provides the use of the scaling, translation and rotation techniques described in the previous chapter to generate a variety of different views of a single picture. We can generate different view of a picture by applying the appropriate scaling and translation. While doing this, we have to identify the visible part of the picture for inclusion in the display image. This selection process is not straight forward. Certain lines may lie partly inside the visible portion of the picture and partly outside. These lines cannot be omitted entirely from the display image because the image would become inaccurate. This is illustrated in Fig. 5.1 The process of selecting and viewing the picture with different views is called **windowing**, and a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded is called **clipping**.

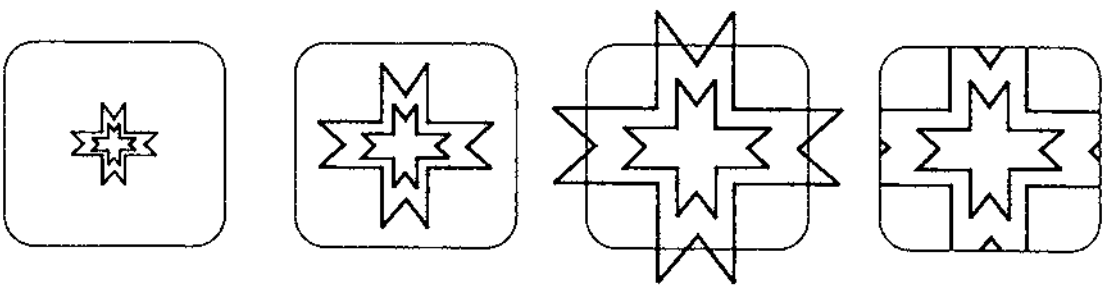


Fig. 5.1

In this chapter we are going to discuss the concepts involved in windowing and various clipping algorithms

5.2 Viewing Transformation

We know that the picture is stored in the computer memory using any convenient cartesian coordinate system, referred to as **world coordinate system** (WCS). However, when picture is displayed on the display device it is measured in **physical device coordinate system** (PDCS) corresponding to the display device. Therefore, displaying an

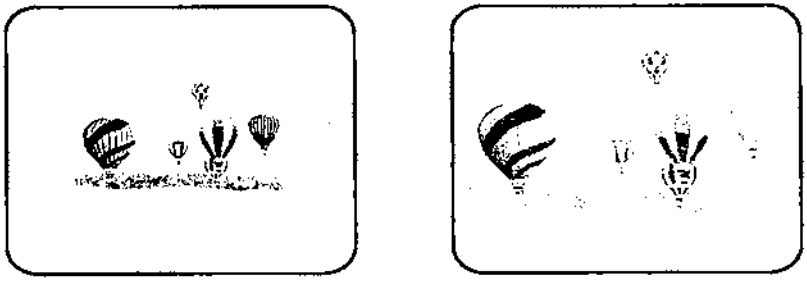
image of a picture involves mapping the coordinates of the points and lines that form the picture into the appropriate physical device coordinate where the image is to be displayed. This mapping of coordinates is achieved with the use of coordinate transformation known as **viewing transformation**.

The viewing transformation which maps picture coordinates in the WCS to display coordinates in PDCS is performed by the following transformations :

- Normalization transformation (N) and
- Workstation transformation (W)

5.2.1 Normalization Transformation

We know that, different display devices may have different screen sizes as measured in pixels. Size of the screen in pixels increases as resolution of the screen increases. When picture is defined in the pixel values then it is displayed large in size on the low resolution screen while small in size on the high resolution screen as shown in the Fig. 5.2. To avoid this and to make our programs to be device independent, we have to define the picture coordinates in some units other than pixels and use the interpreter to convert these coordinates to appropriate pixel values for the particular display device. The device independent units are called the **normalized device coordinates**. In these units, the screen measures 1 unit wide and 1 unit length as shown in the Fig. 5.3. The lower left corner of the screen is the origin, and the upper-right corner is the point (1, 1). The point(0.5, 0.5) is the center of the screen no matter what the physical dimensions or resolution of the actual display device may be.



(a) More resolution (b) Less resolution

Fig. 5.2 Picture definition in pixels

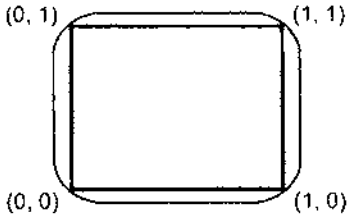


Fig. 5.3 Picture definition in normalized device coordinates

The interpreter uses a simple linear formula to convert the normalized device coordinates to the actual device coordinates.

$$x = x_n \times X_W \quad \ldots (5.1)$$

$$y = y_n \times Y_H \quad \ldots (5.2)$$

where

x : Actual device x coordinate

y : Actual device y coordinate

x_n : Normalized x coordinate

y_n : Normalized y coordinate

X_W : Width of actual screen in pixels

Y_H : Height of actual screen in pixels.

The transformation which maps the world coordinate to normalized device coordinate is called **normalization transformation**. It involves scaling of x and y, thus it is also referred to as **scaling transformation**.

5.2.2 Workstation Transformation

The transformation which maps the normalized device coordinates to physical device coordinates is called workstation transformation.

The viewing transformation is the combination of normalization transformation and workstation transformations as shown in the Fig. 5.4. It is given as

$$V = N \cdot W \quad \ldots (5.3)$$

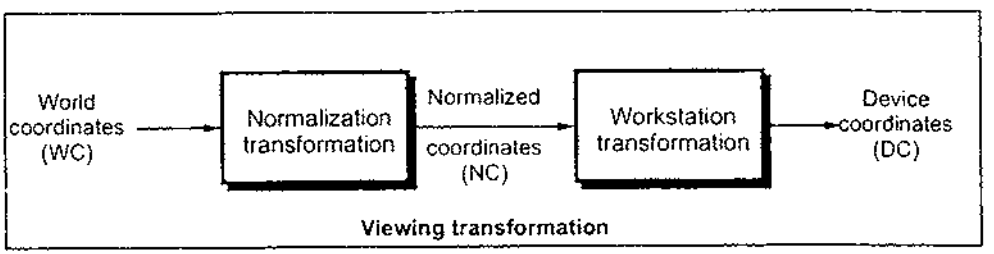


Fig. 5.4 Two dimensional viewing transformation

We know that world coordinate system (WCS) is infinite in extent and the device display area is finite. Therefore, to perform a viewing transformation we select a finite world coordinate area for display called a **window**. An area on a device to which a window is mapped is called a **viewport**. The window defines what is to be viewed; the viewport defines where it is to be displayed, as shown in the Fig. 5.5.

The window defined in world coordinates is first transformed into the normalized device coordinates. The normalized window is then transformed into the viewport coordinate. This window to viewport coordinate transformation is known as workstation transformation. It is achieved by performing following steps :

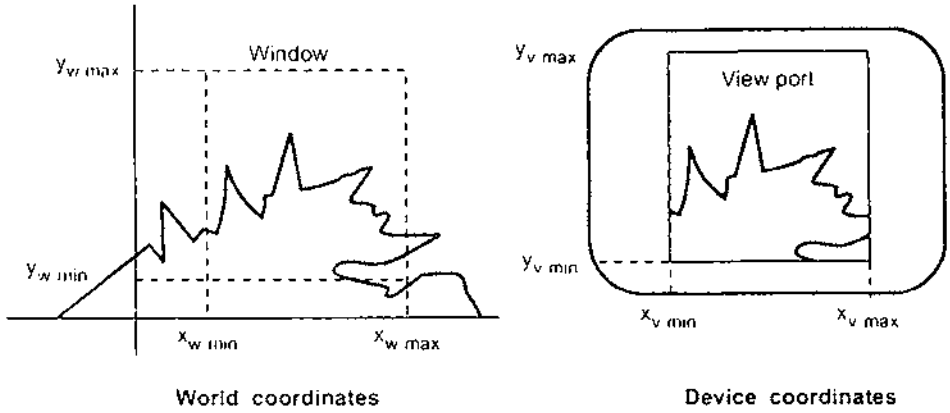


Fig. 5.5 Window and viewport

1. The object together with its window is translated until the lower left corner of the window is at the origin.
2. Object and window are scaled until the window has the dimensions of the viewport.
3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig.5.6.

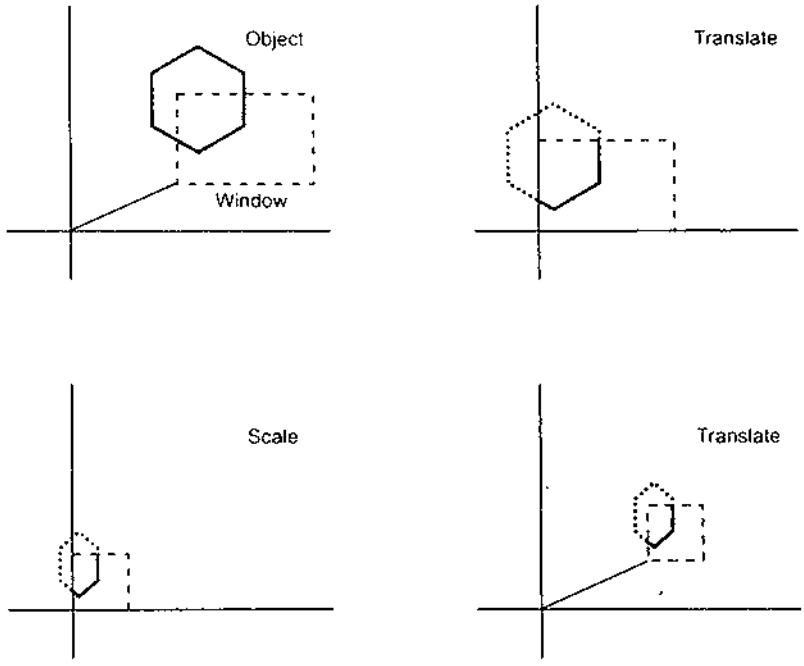


Fig. 5.6 Steps in workstation transformation

Therefore, the workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \qquad \ldots (5.4)$$

The transformation matrices for individual transformation are as given below :

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w\,min} & -y_{w\,min} & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{where} \quad S_x = \frac{x_{v\,max} - x_{v\,min}}{x_{w\,max} - x_{w\,min}}$$

$$S_y = \frac{y_{v\,max} - y_{v\,min}}{y_{w\,max} - y_{w\,min}}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{v\,min} & y_{v\,min} & 1 \end{bmatrix}$$

The overall transformation matrix for W is given as

$$W = T \cdot S \cdot T^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w\,min} & -y_{w\,min} & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{v\,min} & y_{v\,min} & 1 \end{bmatrix}$$

$$= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v\,min} - x_{w\,min} \cdot S_x & y_{v\,min} - y_{w\,min} \cdot S_y & 1 \end{bmatrix}$$

The Fig. 5.7 shows the complete viewing transformation.

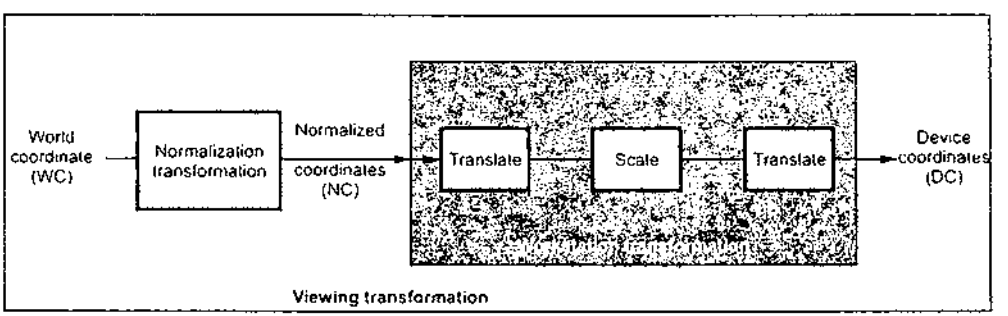


Fig. 5.7 Viewing transformation

Ex. 5.1 : *Find the normalization transformation window to viewpoint, with window, lower left corner at (1, 1) and upper right corner at (3, 5) onto a viewpoint with lower left corner at (0, 0) and upper right corner at (1/2, 1/2).*

Sol. : Given : Coordinates for window

$$x_{w\,min} = 1 \qquad y_{w\,min} = 1$$
$$x_{w\,max} = 3 \qquad y_{w\,max} = 5$$

Coordinates for view port

$$x_{v\,min} = 0 \qquad y_{v\,min} = 0$$
$$x_{v\,max} = 1/2 = 0.5 \qquad y_{v\,max} = 1/2 = 0.5$$

We know that,

$$S_x = \frac{x_{v\,max} - x_{v\,min}}{x_{w\,max} - x_{w\,min}}$$
$$= \frac{0.5 - 0}{3 - 1}$$
$$= 0.25$$

and
$$S_y = \frac{y_{v\,max} - y_{v\,min}}{y_{w\,max} - y_{w\,min}}$$
$$= \frac{0.5 - 0}{5 - 1}$$
$$= 0.125$$

We know that transformation matrix is given as

$$T \cdot S \cdot T^{-1} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v\,min} - x_{w\,min} S_x & y_{v\,min} - y_{w\,min} S_y & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 - (1 \times 0.25) & 0 - (1 \times 0.125) & 1 \end{bmatrix}$$
$$= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & -0.125 & 1 \end{bmatrix}$$

5.3 2D Clipping

The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping. The region against which an object is to be clipped is called a **clip window** or **clipping window.** It usually is in a rectangular shape, as shown in the Fig. 5.8.

The clipping algorithm determines which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display. All others are discarded.

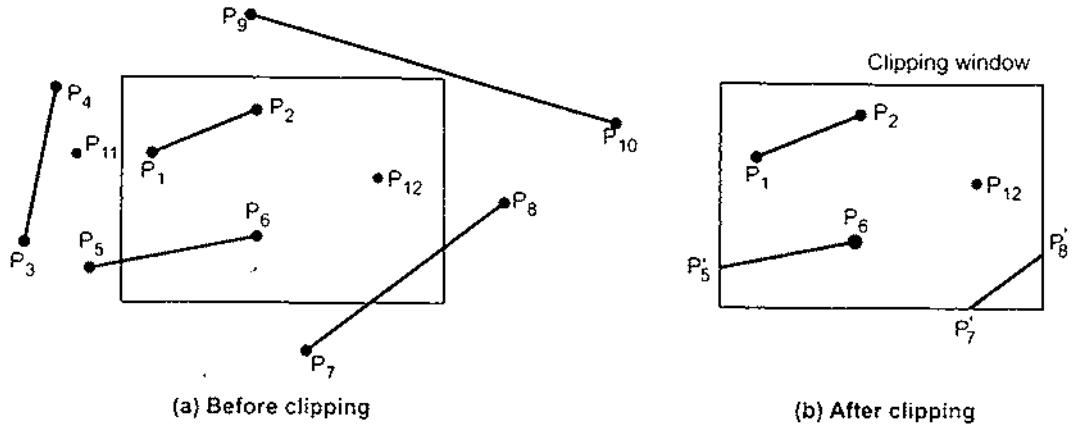


Fig. 5.8

5.3.1 Point Clipping

The points are said to be interior to the clipping window if

$$x_{w\,min} \le x \le x_{w\,max} \quad \text{and}$$

$$y_{w\,min} \le y \le y_{w\,max}$$

The equal sign indicates that points on the window boundary are included within the window.

5.3.2 Line Clipping

The lines are said to be interior to the clipping window and hence visible if both end points are interior to the window, e.g., line $P_1 P_2$ in Fig. 5.8. However, if both end points of a line are exterior to the window, the line is not necessarily completely exterior to the window, e.g. line $P_7 P_8$ in Fig. 5.8. If both end points of a line are completely to the right of, completely to the left of, completely above, or completely below the window, then the line is completely exterior to the window and hence invisible. For example, line $P_3 P_4$ in Fig. 5.8.

The lines which across one or more clipping boundaries require calculation of multiple intersection points to decide the visible portion of them. To minimize the intersection calculations and to increase the efficiency of the clipping algorithm, initially, completely visible and invisible lines are identified and then the intersection points are calculated for remaining lines. There are many line clipping algorithms. Let us discuss a few of them.

5.3.2.1 Sutherland and Cohen Subdivision Line Clipping Algorithm

This is one of the oldest and most popular line clipping algorithm developed by Dan Cohen and Ivan Sutherland. To speed up the processing this algorithm performs initial tests that reduce the number of intersections that must be calculated. This algorithm uses a four digit (bit) code to indicate which of nine regions contain the end point of line. The four bit codes are called **region codes** or **outcodes**. These codes identify the location of the point relative to the boundaries of the clipping rectangle as shown in the Fig. 5.9.

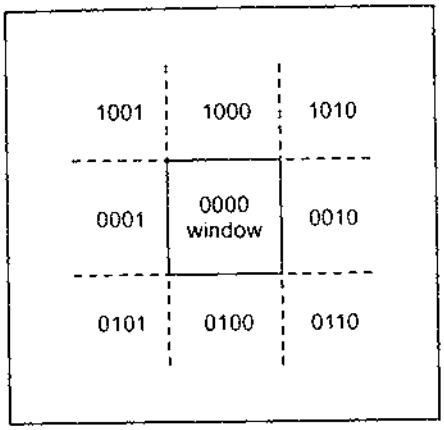


Fig. 5.9 Four-bit codes for nine regions

Each bit position in the region code is used to indicate one of the four relative coordinate positions of the point with respect to the clipping window : to the left, right, top or bottom. The rightmost bit is the first bit and the bits are set to 1 based on the following scheme :

Set Bit 1 – if the end point is to the **left** of the window

Set Bit 2 – if the end point is to the **right** of the window

Set Bit 3 – if the end point is **below** the window

Set Bit 4 – if the end point is **above** the window

Otherwise, the bit is set to zero.

Once we have established region codes for all the line endpoints, we can determine which lines are completely inside the clipping window and which are clearly outside. Any lines that are completely inside the window boundaries have a region code of 0000 for both endpoints and we trivially accept these lines. Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines. A method used to test lines for total clipping is equivalent to the logical **AND** operator. If the result of the logical AND operation with two end point codes is not 0000, the line is completely outside the clipping region. The lines that cannot be identified as completely inside or completely outside a clipping window by these tests are checked for intersection with the window boundaries.

Ex. 5.2 : *Consider the clipping window and the lines shown in Fig. 5.10. Find the region codes for each end point and identify whether the line is completely visible, partially visible or completely invisible.*

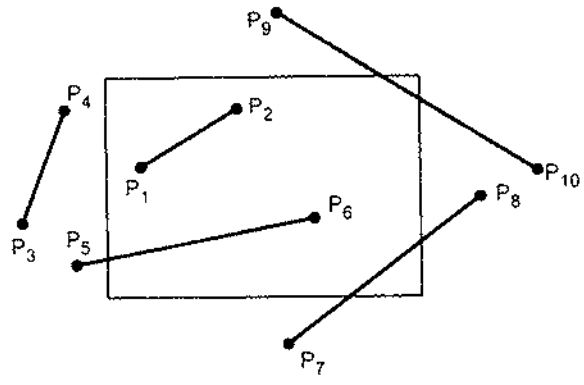


Fig. 5.10

Sol. : The Fig. 5.11 shows the clipping window and lines with region codes. These codes are tabulated and end point codes are logically ANDed to identify the visibility of the line in table 5.1.

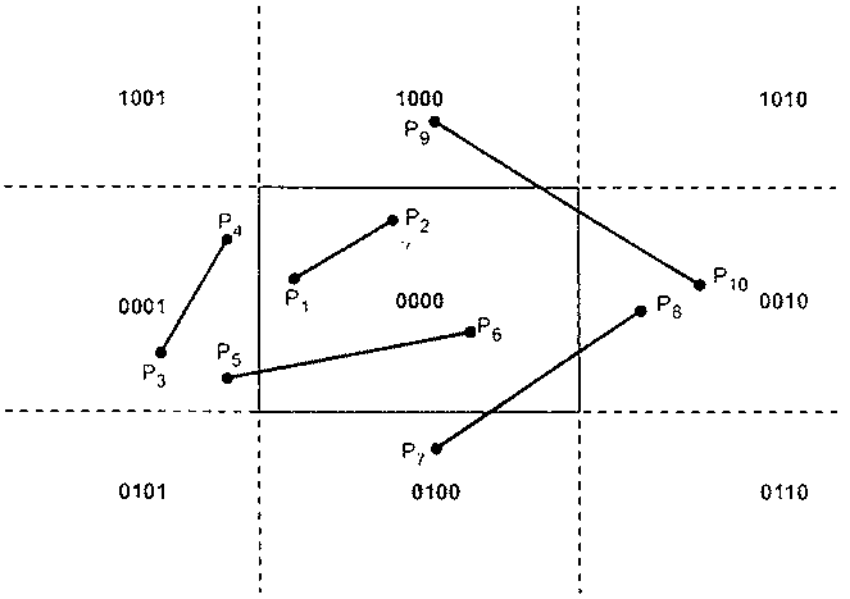


Fig. 5.11

Line	End Point Codes		Logical ANDing	Result
$P_1 P_2$	0000	0000	0000	Completely visible
$P_3 P_4$	0001	0001	0001	Completely invisible
$P_5 P_6$	0001	0000	0000	Partially visible
$P_7 P_8$	0100	0010	0000	Partially visible
$P_9 P_{10}$	1000	0010	0000	Partially visible

Table 5.1

The Sutherland - Cohen algorithm begins the clipping process for a partially visible line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the line is checked against the other boundaries, and the process is continued until either the line is totally discarded or a section is found inside the window.

This is illustrated in Fig. 5.12.

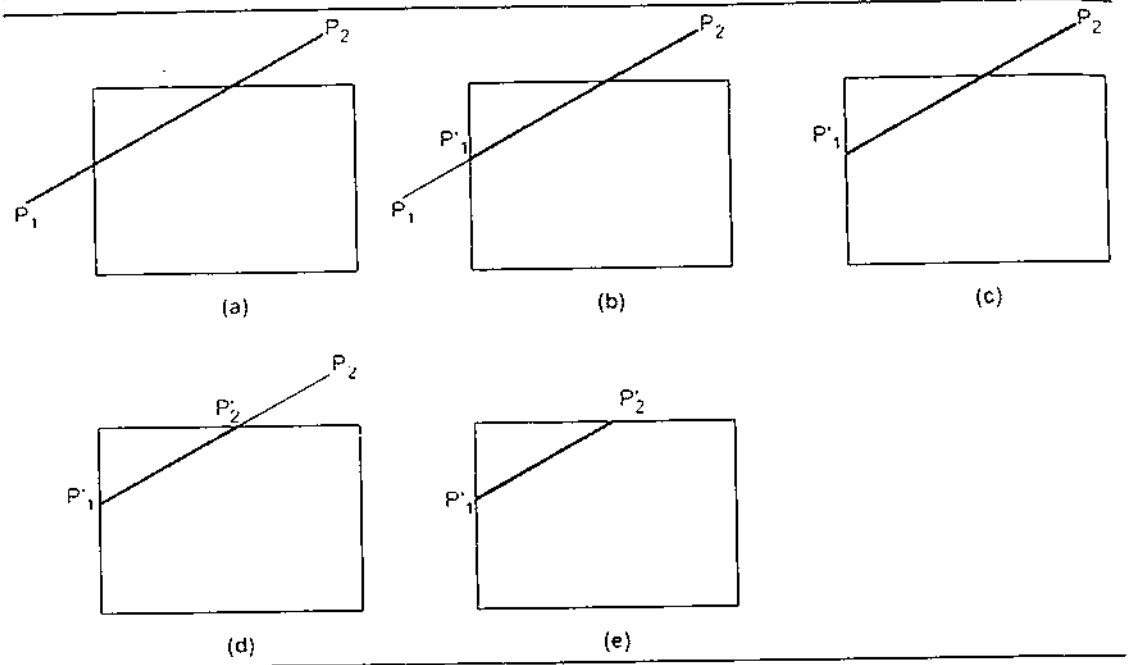


Fig. 5.12 Sutherland-Cohen subdivision line clipping

As shown in the Fig. 5.12, line $P_1 P_2$ is a partially visible and point P_1 is outside the window. Starting with point P_1, the intersection point P_1' is found and we get two line segments $P_1 - P_1'$ and $P_1' - P_2$. We know that, for $P_1 - P_1'$ one end point i.e. P_1 is outside the window and thus the line segment $P_1 - P_1'$ is discarded. The line is now reduced to the section from P_1' to P_2. Since P_2 is outside the clip window, it is checked against the boundaries and intersection point P_2' is found. Again the line segment is divided into two segments giving $P_1' - P_2'$ and $P_2' - P_2$. We know that, for $P_2' - P_2$ one end point i.e. P_2 is outside the window and thus the line segment $P_2' - P_2$ is discarded. The remaining line segment $P_1' - P_2'$ is completely inside the clipping window and hence made visible.

The intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation. The equation for line passing through points $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$ is

$$y = m(x - x_1) + y_1 \quad \text{or} \quad y = m(x - x_2) + y_2 \qquad \ldots (5.5)$$

where $$m = \frac{y_2 - y_1}{x_2 - x_1} \quad \text{(slope of the line)}$$

Therefore, the intersections with the clipping boundaries of the window are given as :

Left : $x_L, y = m(x_L - x_1) + y_1 \quad ; \quad m \neq \infty$

Right : $x_R, y = m(x_R - x_1) + y_1 \quad ; \quad m \neq \infty$

Top : $y_T, x = x_1 + \left(\frac{1}{m}\right)(y_T - y_1) \quad ; \quad m \neq 0$

Bottom : $y_B, x = x_1 + \left(\frac{1}{m}\right)(y_B - y_1) \quad ; \quad m \neq 0$

Sutherland and Cohen subdivision line clipping algorithm :

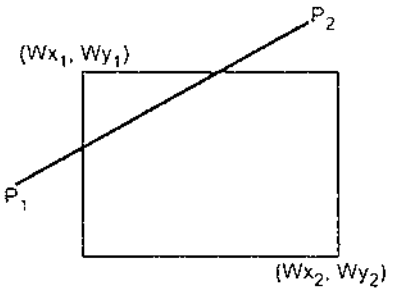


Fig. 5.13

1. Read two end points of the line say $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1 and Wx_2, Wy_2).
3. Assign the region codes for two endpoints P_1 and P_2 using following steps :

   Initialize code with bits 0000

   Set Bit 1 – if $(x < Wx_1)$

   Set Bit 2 – if $(x > Wx_2)$

   Set Bit 3 – if $(y < Wy_2)$

   Set Bit 4 – if $(y > Wy_1)$
4. Check for visibility of line $P_1 P_2$

   a) If region codes for both endpoints P_1 and P_2 are zero then the line is completely visible. Hence draw the line and go to step 9.

   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 9.

   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b)the line is partially visible.
5. Determine the intersecting edge of the clipping window by inspecting the region codes of two endpoints.

   a) If region codes for both the end points are non-zero, find intersection points P_1' and P_2' with boundary edges of clipping window with respect to point P_1 and point P_2, respectively

   b) If region code for any one end point is non zero then find intersection point P_1' or P_2' with the boundary edge of the clipping window with respect to it.
6. Divide the line segments considering intersection points.
7. Reject the line segment if any one end point of it appears outsides the clipping window.
8. Draw the remaining line segments.
9. Stop.

'C' code for Sutherland and Cohen Subdivision Line Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
/* Defining structure for end point of line */
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2,int cl);
PT setcode(PT p);
int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
 int gd=DETECT, gm,v;
 PT p1,p2,ptemp;
 initgraph(&gd,&gm," ");
 cleardevice();
 printf("\n\n\t\tENTER END-POINT 1 (x,y): ");
 scanf("%d,%d",&p1.x,&p1.y);
 printf("\n\n\t\tENTER END-POINT 2 (x,y): ");
 scanf("%d,%d",&p2.x,&p2.y);
 cleardevice();
 drawwindow();
 getch();
 drawline(p1,p2,15);
 getch();
 p1=setcode(p1);
 p2=setcode(p2);
 v=visibility(p1,p2);
```

```
    switch(v)
        {
        case 0: cleardevice();  /* Line completely visible */
                drawwindow();
                drawline(p1,p2,15);
                break;
        case 1: cleardevice();  /* Line completely invisible */
                drawwindow();
                break;
        case 2: cleardevice();  /* line partly visible */
                p1=resetendpt (p1,p2);
                p2=resetendpt(p2,p1);
                drawwindow();
                drawline(p1,p2,15);
                break;
        }
    getch();
    closegraph();
    return(0);
    }
    /* Function to draw window */
    void drawwindow()
    {
    setcolor(RED);
    line(150,100,450,100);
    line(450,100,450,350);
    line(450,350,150,350);
    line(150,350,150,100);
    }
    /* Function to draw line between two points
    ----------------------------------------------*/
    void drawline (PT p1,PT p2,int cl)
    {
    setcolor(cl);
    line(p1.x,p1.y,p2.x,p2.y);
    }
```

```
/* Function to set code of the coordinates
-----------------------------------------*/
PT setcode(PT p)
{
PT ptemp;
if(p.y<100)
ptemp.code[0]='1'; /* TOP */
else
ptemp.code[0]='0';
if(p.y>350)
ptemp.code[1]='1'; /* BOTTOM */
else
ptemp.code[1]='0';
if (p.x>450)
ptemp.code[2]='1'; /* RIGHT */
else
ptemp.code[2]='0';
if (p.x<150) /* LEFT */
ptemp.code[3]='1';
else
ptemp.code[3]='0';
ptemp.x=p.x;
ptemp.y=p.y;
return(ptemp);
}
/* Function to determine visibility of line
-----------------------------------------*/
int visibility (PT p1,PT p2)
{
int i,flag=0;
for(i=0;i<4;i++)
{
if((p1.code[i]!='0')||(p2.code[i]!='0'))
flag=1;
}
if(flag==0)
return(0);
```

```
{
if((p1.code[i]==p2.code[i]) &&(p1.code[i]=='1'))
i=4;
}
if(i==4)
return(1);
return(2);
}

/* Function to find new end points
-------------------------------------*/
PT resetendpt (PT p1,PT p2)
{
PT temp;
int x,y,i;
float m,k;
if( p1.code[3]=='1') /* Cutting LEFT Edge  */
x=150;
if(p1.code[2]=='1')  /* Cutting RIGHT Edge */
x=450;
if((p1.code[3]=='1')||(p1.code[2]=='1'))
{
m=(float) (p2.y-p1.y)/(p2.x-p1.x);
k=(p1.y+(m*(x-p1.x)));
temp.y=k;
temp.x=x;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
if(temp.y<=350&&temp.y>=100)
return(temp);
}
if(p1.code[0]=='1')  /* Cutting TOP Edge */
y=100;
if(p1.code [1]=='1') /* Cutting BOTTOM Edge */
y=350;
if((p1.code[0]=='1')||(p1.code[1]=='1'))
{
m=(float)(p2.y-p1.y)/(p2.x-p1.x);
```

```
k=(float)p1.x+(float)(y-p1.y)/m;
temp.x=k;
temp.y=y;
for(i=0;i<4;i++)
temp.code[i]=p1.code[i];
return(temp);
}
else
return(p1);
}
```

5.3.2.2 Midpoint Subdivision Algorithm

We have seen that, the Sutherland Cohen subdivision line clipping algorithm requires the calculation of the intersection of the line with the window edge. These calculations can be avoided by repeatatively subdividing the line at its midpoint.

Like previous algorithm, initially the line is tested for visibility. If line is completely visible it is drawn and if it is completely invisible it is rejected. If line is partially visible then it is subdivided in two equal parts. The visibility tests are then applied to each half. This subdivision process is repeated until we get completely visible and completely invisible line segments. This is illustrated in Fig. 5.14. (see on next page)

As shown in the Fig. 5.14, line $P_1 P_2$ is partially visible. It is subdivided in two equal parts $P_1 P_3$ and $P_3 P_2$ (see Fig. 5.14(b)). Both the line segments are tested for visibility and found to be partially visible. Both line segments are then subdivided in two equal parts to get midpoints P_4 and P_5 (see Fig. 5.14 (c)). It is observed that line segments P_1P_4 and P_5P_2 are completely invisible and hence rejected. However, line segment P_3P_5 is completely visible and hence drawn. The remaining line segment P_4P_3 is still partially visible. It is then subdivided to get midpoint P_6. It is observed that P_6P_3 is completely visible whereas P_4P_6 is partially visible. Thus P_6P_3 line segment is drawn and P_4P_6 line segment is further subdivided into equal parts to get midpoint P_7. Now, it is observed that line segment P_4P_7 is completely invisible and line segment P_7P_6 is completely visible (see Fig. 5.14 (f)), and there is no further partially visible segment.

Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1 and Wx_2, Wy_2).
3. Assign region codes for two end points using following steps :

   Initialize code with bits 0000
   Set Bit 1 - if ($x < Wx_1$)
   Set Bit 2 - if ($x > Wx_2$)
   Set Bit 3 - if ($y < Wy_1$)
   Set Bit 4 - if ($y > Wy_2$)

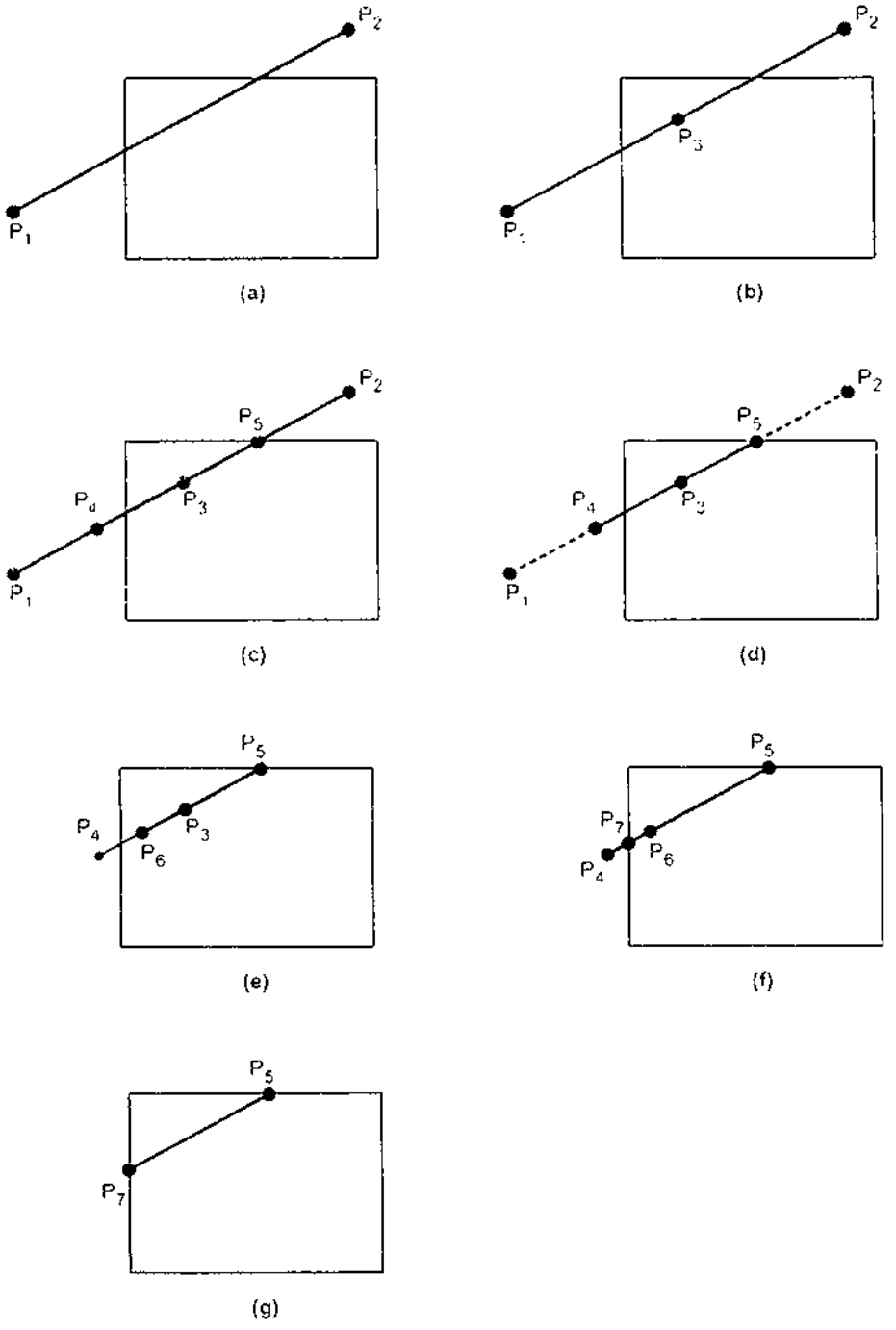


Fig. 5.14 Clipping line with midpoint subdivision algorithm

4. Check for visibility of line
   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

'C' code for Midpoint Subdivision Line Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>
/* Defining structure for end point of line */
typedef struct coordinate
{
int x,y;
char code[4];
}PT;
void drawwindow();
void drawline (PT p1,PT p2,int cl);
PT setcode(PT p);
int visibility (PT p1,PT p2);
PT resetendpt (PT p1,PT p2);
main()
{
 int gd=DETECT, gm,v;
 PT p1,p2,ptemp;
 initgraph(&gd,&gm," ");
 cleardevice();
 printf("\n\n\t\tENTER END-POINT 1 (x,y): ");
 scanf("%d,%d",&p1.x,&p1.y);
```

```
 printf("\n\n\t\tENTER END-POINT 2 (x,y): ");
 scanf("%d,%d",&p2.x,&p2.y);
 cleardevice();
 drawwindow();
 getch();
 drawline(p1,p2,15);
 getch();
 cleardevice();
 drawwindow();
 midsub(p1,p2);
 getch();
 closegraph();
 return(0);
}

midsub(PT p1,PT p2)
{
 PT mid;
 int v;
 p1=setcode(p1);
 p2=setcode(p2);
 v=visibility(p1,p2);
 switch(v)
     {
     case 0: /* Line conpletely visible */
             drawline(p1,p2,15);
             break;
     case 1: /* Line completely invisible */
             break;
     case 2: /* line partly visible */
             mid.x = p1.x + (p2.x-p1.x)/2;
             mid.y = p1.y + (p2.y-p1.y)/2;
             midsub(p1,mid);
             mid.x = mid.x+1;
             mid.y = mid.y+1;
             midsub(mid,p2);
             break;
     }
}
```

```
/* Function to draw window */
void drawwindow()
{
setcolor(RED);
line(150,100,450,100);
line(450,100,450,400);
line(450,400,150,400);
line(150,400,150,100);
}
/* Function to draw line between two points
-----------------------------------------------*/
void drawline (PT p1,PT p2,int c1)
{
setcolor(c1);
line(p1.x,p1.y,p2.x,p2.y);
}

/* Function to set code of the coordinates
-----------------------------------------------*/
PT setcode(PT p)
{
PT ptemp;
if(p.y<=100)
ptemp.code[0]='1'; /* TOP */
else
ptemp.code[0]='0';
if(p.y>=400)
ptemp.code[1]='1'; /* BOTTOM */
else
ptemp.code[1]='0';
if (p.x>=450)
ptemp.code[2]='1'; /* RIGHT */
else
ptemp.code[2]='0';
if (p.x<=150)     /* LEFT */
ptemp.code[3]='1';
else
```

```
ptemp.code[3]='0';
ptemp.x=p.x;
ptemp.y=p.y;
return(ptemp);
}

/* Function to determine visibility of line
-----------------------------------------*/
int visibility (PT p1,PT p2)
{
int i,flag=0;
for(i=0;i<4;i++)
{
if((p1.code[i]!='0')||(p2.code[i]!='0'))
flag=1;
}
if(flag==0)
return(0);
for(i=0;i<4;i++)
{
if((p1.code[i]==p2.code[i]) &&(p1.code[i]=='1'))
flag=0;
}
if(flag==0)
return(1);
return(2);
}
```

The midpoint subdivision algorithm requires repeated subdivision of line segments and hence many times it is slower than using direct calculation of the intersection of the line with the clipping window edge. However, it can be implemented efficiently using parallel architecture since it involves parallel operations.

5.3.2.3 Generalized Clipping with Cyrus-Beck Algorithm

The algorithms explained above assume that the clipping window is a regular rectangle. These algorithms are not applicable for non rectangular clipping windows. Cyrus and Beck have developed a generalized line clipping algorithm. This algorithm is applicable to an arbitrary convex region. This algorithm uses a parametric equation of a line segment to find the intersection points of a line with the clipping edges. The parametric equation of a line segment from P_1 to P_2 is

$$P(t) = P_1 + (P_2 - P_1)\,t \qquad ; 0 \le t \le 1 \qquad \ldots (5.6)$$

where t is a parameter, t = 0 at P_1 and t = 1 at P_2.

Consider a convex clipping region R, f is a boundary point of the convex region R and n is an inner normal for one of its boundaries, as shown in the Fig. 5.15.

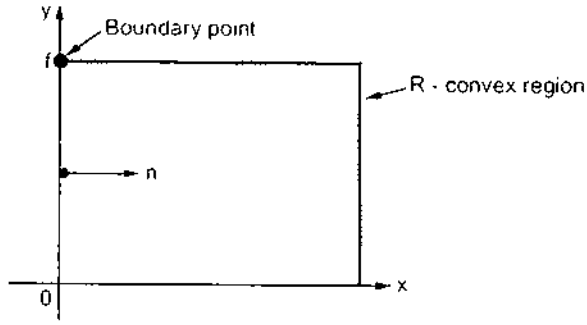


Fig. 5.15 Convex region, boundary point and inner normal

Then we can distinguish in which region a point lie by looking at the value of the dot product $n \cdot [\,P(t) - f\,]$, as shown in the Fig. 5.16.

1. If dot product is negative, i.e.

$$n \cdot [\,P(t) - f] < 0 \qquad \ldots (5.7)$$

Then the vector P(t) - f is pointed away from the interior of R.

2. If dot product is zero, i.e.,

$$n \cdot [\,P(t) - f\,] = 0 \qquad \ldots (5.8)$$

Then P(t) – f is pointed parallel to the plane containing f and perpendicular to the normal.

3. If dot product is positive, i.e.

$$n \cdot [\,P(t) - f\,] > 0 \qquad \ldots (5.9)$$

Then the vector P(t) – f is pointed towards the interior of R, as shown in Fig. 5.16.

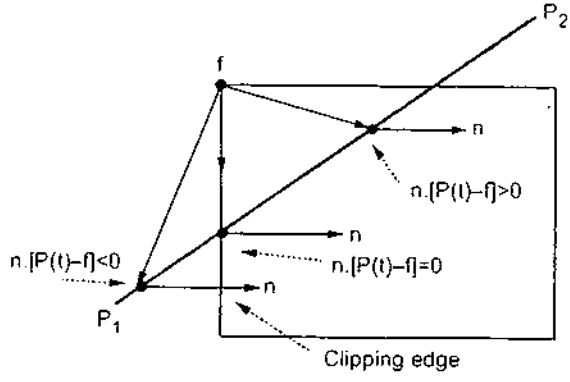


Fig. 5.16 Dot products for three points inside, outside and on the boundary of the clipping region

As shown in the Fig. 5.16, if the point f lies in the boundary plane or edge for which n is the inner normal, then that point t on the line P(t) which satisfies $n \cdot [P(t) - f] = 0$ condition is the intersection of the line with the boundary edge.

Ex. 5.3 : *Consider the line from P_1 (– 2, 1) to P_2 (8, 4) clipped to the rectangular region R as shown in the Fig. 5.17. The line P_1P_2 intersects the window. Calculate the intersection points.*

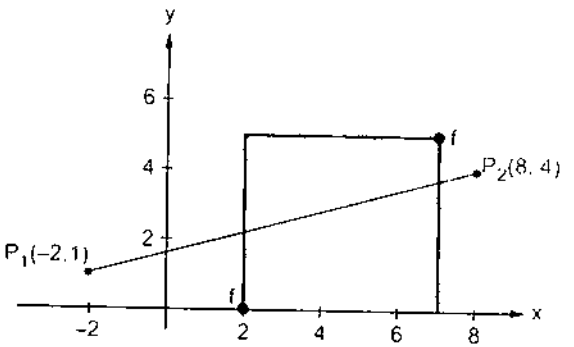


Fig. 5.17

Sol. : The parametric representation of the line P_1P_2 is

$$P(t) = P_1 + (P_2 - P_1)t = [-2 \;\; 1] + [10 \;\; 3]\, t$$
$$= (10t - 2)\, i + (3t + 1)\, j; \qquad 0 \le t \le 1$$

where i and j are the unit vectors in the x and y directions, respectively. The four inner normals are given as

Left : $n_L = i$

Right : $n_R = -i$

Bottom : $n_B = j$

Top : $n_T = -j$

Choosing f (2, 0) for the left edge gives

$$P(t) - f = (10t - 4)\, i + (3t + 1)\, j \quad \text{and}$$
$$n_L \cdot [P(t) - f] = 10t - 4 = 0$$
$$\therefore \quad t = 2/5$$

Substituting value of t in parametric equation we get,

$$P(2/5) = [-2 \;\; 1] + [10 \;\; 3]\,(2/5)$$
$$= [-2 \;\; 1] + [4 \;\; 6/5]$$
$$= [2 \;\; 2.2]$$

Choosing f(7, 5) for the right edge gives

$$P(t) - f = (10t - 9)\, i + (3t - 4)\, j \text{ and}$$
$$n_R \cdot [P(t) - f)] = -(10t - 9) = 0$$
$$\therefore \quad t = 9/10$$

Substituting value of t in parametric equation we get,

$$P(9/10) = [-2 \;\; 1] + [10 \;\; 3]\,(9/10)$$

$$= [-2 \quad 1] + [9 \quad 27/10]$$
$$= [7 \quad 37/10] = [7 \quad 3.7]$$

Using f(2, 0) for the bottom edge gives

$$n_B \cdot [P(t) - f] = 3t + 1 = 0$$
$$\therefore \quad t = -1/3$$

This value of t is **outside** the range of $0 \le t \le 1$ and **hence it is rejected.**

Using f(7, 5) for the top edge gives

$$n_T \cdot [P(t) - f] = -(3t - 4) = 0$$
$$\therefore \quad t = 4/3$$

This value of t is **outside** the range of $0 \le t \le 1$ and **hence it is rejected.**

Thus, we get two intersection points (2, 2.2) and (7, 3.7) with left edge and right edge, respectively.

To get the formal statement of the Cyrus-Beck algorithm we substitute value of P(t) in equation 5.8.

$$\therefore \quad n \cdot [P(t) - f] = n \cdot [P_1 + (P_2 - P_1)\, t - f] = 0 \qquad \ldots (5.10)$$

This relation should be applied for each boundary plane or edge of the window to get the intersection points. Thus in general form equation 5.10 can be written as,

$$n_i \cdot [P_1 + (P_2 - P_1)\, t - f_i] = 0 \qquad \ldots (5.11)$$

where i : edge number

Solving equation 5.11 we get,

$$n_i \cdot [P_1 - f_i] + n_i \cdot [P_2 - P_1]\, t = 0 \qquad \ldots (5.12)$$

Here, the vector $P_2 - P_1$ defines the direction of the line. The direction of line is important to correctly identify the visibility of the line. The vector $P_1 - f_i$ is proportional to the distance from the end point of the line to the boundary point.

Let us define,

$D = P_2 - P_1$ as the directorix or direction of a line and

$W_i = P_1 - f_i$ as a weighting factor.

Substituting newly defined variable D and W_i in equation 5.12 we get,

$$n_i \cdot W_i + (n_i \cdot D)\, t = 0 \qquad \ldots (5.13)$$

$$\therefore \quad t = -\frac{n_i \cdot W_i}{D \cdot n_i} \qquad \ldots (5.14)$$

where $D \neq 0$ and $i = 1, 2, 3, \ldots$

The equation 5.14 is used to obtain the value of t for the intersection of the line with each edge of the clipping window. We must select the proper value for t using following tips :

1. If t is outside the range $0 \le t \le 1$, then it can be ignored.
2. We know that, the line can intersect the convex window in at most two points, i.e. at two values of t. With equation 5.14, there can be several values of t in the range of

$0 \le t \le 1$. We have to choose the largest lower limit and the smallest upper limit (see Ex. 5.3).

If $D_i \cdot n_i > 0$ then equation 5.14 gives the lower limit value for t and if $D_i \cdot n_i < 0$ then equation 5.14 gives the upper limit value for t.

Cyrus-Beck Line Clipping Algorithm

1. Read two end points of the line, say P_1 and P_2
2. Read vertex coordinates of the clipping window
3. Calculate $D = P_2 - P_1$
4. Assign boundary point (f) with particular edge
5. Find inner normal vector for corresponding edge
6. Calculate $D \cdot n$ and $W = P_1 - f$
7. If $D \cdot n > 0$

   $$t_L = -\frac{W \cdot n}{D \cdot n}$$

   else

   $$t_U = -\frac{W \cdot n}{D \cdot n}$$

   end if
8. Repeat steps 4 through 7 for each edge of the clipping window
9. Find maximum lower limit and minimum upper limit
10. If maximum lower limit and minimum upper limit do not satisfy condition $0 \le t \le 1$ then ignore the line.
11. Calculate the intersection points by substituting values of maximum lower limit and minimum upper limit in the parametric equation of the line P_1P_2.
12. Draw the line segment $P(t_L)$ to $P(t_U)$.
13. Stop.

Ex. 5.4 : *Fig. 5.18 shows the Hexagonal clipping window. The line P_1 (– 2, 1) to P_2 (6, 3) is to be clipped to this window. Find the intersection points.*

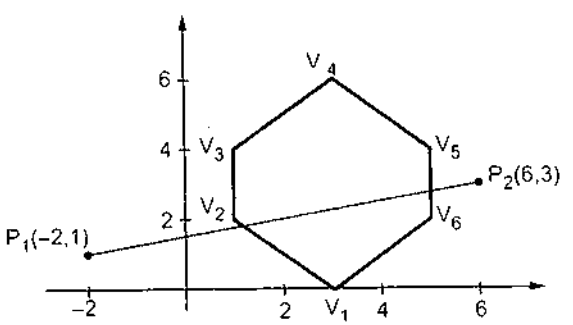


Fig. 5.18

Sol. : We know that,

$$D = P_2 - P_1 = [6\ \ 3] - [-2\ 1] = [8\ \ 2]$$

For boundary point f(3, 0)

$$W = P_1 - f = [-2\ \ 1] - [3\ \ 0] = [-5\ \ 1]$$

For the edge $V_1 V_2$ the inner normal is

$$n = [1\ \ 1]$$

Hence $$D \cdot n = [8\ \ 2] \cdot [1\ \ 1] = 10 > 0$$

∴ The lower limit can be given as

$$t_L = -\frac{W \cdot n}{D \cdot n} = -\frac{[-5\ \ 1][1\ \ 1]}{10}$$

$$= -\frac{(-4)}{10} = \frac{4}{10}$$

Similar calculation with each edge gives the complete results of the Cyrus-Beck algorithm. These results are tabulated in table 5.2.

Edge	n	f	W	W · n	D · n	t_L	t_U
$V_1 V_2$	[1 1]	(3, 0)	[-5 1]	-4	10	4/10	
$V_2 V_3$	[1 0]	(1, 4)	[-3 -3]	-3	8	3/8	
$V_3 V_4$	[1 -1]	(1, 4)	[-3 -3]	0	6	0	
$V_4 V_5$	[-1 -1]	(5, 4)	[-7 -3]	10	-10		10/10
$V_5 V_6$	[-1 0]	(5, 4)	[-7 -3]	7	-8		7/8
$V_6 V_1$	[-1 1]	(3, 0)	[-5 1]	6	-6		6/6

Table 5.2

Referring table 5.2 we have,

The maximum lower limit (t_L) = 4/10 and

The minimum upper limit (t_U) = 7/8

Substituting these values of t in parametric equation

we get,

$$P(4/10) = [-2\ \ 1] + [8\ \ 2]\,(4/10)$$
$$= [-2\ \ 1] + [3.2\ \ 0.8]$$
$$= [1.2\ \ 1.8]$$
$$P(7/8) = [-2\ \ 1] + [8\ \ 2]\ (7/8)$$
$$= [-2\ \ 1] + [7\ \ 1.75]$$
$$= [5\ \ 2.75]$$

Thus, the two intersection points to line P_1P_2 are [1.2 1.8] and [5 2.75] with edges V_1V_2 and V_5V_6, respectively.

5.3.2.4 Liang-Barsky Line Clipping Algorithm

In the last section we have seen Cyrus-Beck line clipping algorithm using parametric equations. It is more efficient than Cohen-Sutherland algorithm. Liang and Barsky have developed even more efficient algorithm than Cyrus-Beck algorithm using parametric equations. These parametric equations are given as

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y, \qquad 0 \le t \le 1$$

where $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$

The point clipping conditions (Refer section 5.3.1) for Liang-Barsky approach in the parametric form can be given as

$$x_{wmin} \le x_1 + t\Delta x \le x_{wmax} \text{ and}$$
$$y_{wmin} \le y_1 + t\Delta y \le y_{wmax}$$

Liang-Barsky express these four inequalities with two parameters p and q as follows :

$$tp_i \le q_i \qquad i = 1, 2, 3, 4$$

where parameters p and q are defined as

$$p_1 = -\Delta x, \quad q_1 = x_1 - x_{wmin}$$
$$p_2 = \Delta x, \quad q_2 = x_{wmax} - x_1$$
$$p_3 = -\Delta y, \quad q_3 = y_1 - y_{wmin}$$
$$p_4 = \Delta y, \quad q_4 = y_{wmax} - y_1$$

Following observations can be easily made from above definitions of parameters p and q.

If $p_1 = 0$: Line is parallel to left clipping boundary.

If $p_2 = 0$: Line is parallel to right clipping boundary.

If $p_3 = 0$: Line is parallel to bottom clipping boundary.

If $p_4 = 0$: Line is parallel to top clipping boundary.

If $p_i = 0$, and for that value of i,

- If $q_i < 0$: Line is completely outside the boundary and can be eliminated.
- If $q_i \ge 0$: Line is inside the clipping boundary.

If $p_i < 0$: Line proceeds from outside to inside of the clipping boundary.

If $p_i > 0$: Line proceeds from inside to outside of the clipping boundary.

Therefore, for nonzero value of p_i, the line crosses the clipping boundary and we have to find parameter t. The parameter t for any clipping boundary i can be given as

$$t = \frac{q_i}{p_i} \qquad i = 1, 2, 3, 4$$

Liang-Barsky algorithm calculates two values of parameter t : t_1 and t_2 that define that part of the line that lies within the clip rectangle. The value of t_1 is determined by checking

the rectangle edges for which the line proceeds from the outside to the inside ($p < 0$). The value of t_1 is taken as a largest value amongst various values of intersections with all edges. On the other hand, the value of t_2 is determined by checking the rectangle edges for which the line proceeds from the inside to the outside ($p > 0$). The minimum of the calculated value is taken as a value for t_2.

Now, if $t_1 > t_2$, the line is completely outside the clipping window and it can be rejected. Otherwise the values of t_1 and t_2 are substituted in the parametric equations to get the end points of the clipped line.

Algorithm

1. Read two endpoints of the line say p_1 (x_1, y_1) and p_2 (x_2, y_2).
2. Read two corners (left-top and right-bottom) of the window, say $(x_{wmin}, y_{wmax}, x_{wmax}, y_{wmin})$
3. Calculate the values of parameters p_i and q_i for $i = 1, 2, 3, 4$ such that

   $p_1 = -\Delta x \quad q_1 = x_1 - x_{wmin}$

   $p_2 = \Delta x \quad q_2 = x_{wmax} - x_1$

   $q_1 = -\Delta y \quad q_3 = y_1 - y_{wmin}$

   $q_2 = \Delta y \quad q_4 = y_{wmax} - y_1$
4. if $p_i = 0$, then

   { The line is parallel to i^{th} boundary.

   Now, if $q_i < 0$ then

   { line is completely outside the boundary, hence discard the line segment and goto stop.

   }

   else

   { Check whether the line is horizontal or vertical and accordingly check the line endpoint with corresponding boundaries. If line endpoint/s lie within the bounded area then use them to draw line otherwise use boundary coordinates to draw line. Go to stop.

   }

   }
5. Initialise values for t_1 and t_2 as

   $t_1 = 0$ and $t_2 = 1$
6. Calculate values for q_i/p_i for $i = 1, 2, 3, 4$
7. Select values of q_i/p_i where $p_i < 0$ and assign maximum out of them as t_1.
8. Select values of q_i/p_i where $p_i > 0$ and assign minimum out of them as t_2.
9. If $(t_1 < t_2)$

   { Calculate the endpoints of the clipped line as follows :

   $xx_1 = x_1 + t_1 \Delta x$

$xx_2 = x_1 + t_2 \Delta x$

$yy_1 = y_1 + t_1 \Delta y$

$yy_2 = y_1 + t_2 \Delta y$

Draw line (xx_1, yy_1, xx_2, yy_2)

}

10. Stop.

'C' code for Liang-Barsky Line Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
main()
{
int i,gd,gm;
int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2;
float t1,t2,p[4],q[4],temp;
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");
x1=10;
y1=10;
x2=60;
y2=30;
xmin = 15;
xmax = 25;
ymin = 15;
ymax = 25;

rectangle(xmin,ymin,xmax,ymax);
p[0] = -(x2-x1);
p[1] = (x2-x1);
p[2] = -(y2-y1);
p[3] = (y2-y1);
q[0] = (x1-xmin);
q[1] = (xmax-x1);
q[2] = (y1-ymin);
q[3] = (ymax-y1);
```

```
for(i=0;i<4;i++)
{
    if(p[i]==0)
    {
    printf("line is parallel to one of the clipping boundary");
    if(q[i] >= 0)
    {
        if(i < 2)
        {       if (y1 < ymin)
            {
            y1 = ymin;
            }
            if (y2 > ymax)
            {
            y2 = ymax;
            }
            line(x1,y1,x2,y2);
        }
        if(i > 1)
        {
            if (x1 < xmin)
            {
            x1 = xmin;
            }
            if (x2 > xmax)
            {
            x2 = xmax;
            }
            line(x1,y1,x2,y2);
        }
    }
    getch();
    return(0);
    }
}
t1 = 0;
t2 = 1;
for(i=0;i<4;i++)
```

```
{
    temp = q[i]/p[i];
    if(p[i] < 0)
    {
        if(t1 <= temp)
        {
        t1 = temp;
        }
    }
    else
    {
        if(t2 > temp)
        {
        t2 = temp;
        }
    }
}
if(t1<t2)
{
xx1 = x1 + t1 * p[1];
xx2 = x1 + t2 * p[1];
yy1 = y1 + t1 * p[3];
yy2 = y1 + t2 * p[3];
line(xx1,yy1,xx2,yy2);
}
getch();
closegraph();
}
```

Advantages

1. It is more efficient than Cohen-Sutherland algorithm, since intersection calculations are reduced.
2. It requires only one division to update parameters t_1 and t_2.
3. Window intersections of the line are computed only once.

Ex. 5.5 *Find the clipping coordinates for a line p_1p_2 where $p_1 = (10, 10)$ and p_2 (60, 30), against window with $(x_{wmin}, y_{wmin}) = (15, 15)$ and $(x_{wmax}, y_{wmax}) = (25, 25)$.*

Sol. : Here,

$x_1 = 10$ $x_{wmin} = 15$

$y_1 = 10$ $y_{wmin} = 15$

$x_2 = 60$ $x_{wmax} = 25$

$y_2 = 30$ $y_{wmax} = 25$

$p_1 = -50$ $q_1 = -5$ $p_1/q_1 = 0.1$

$p_2 = 50$ $q_2 = 15$ $p_2/q_2 = 0.3$

$p_3 = -20$ $q_3 = -5$ $p_3/q_3 = 0.25$

$p_4 = 20$ $q_4 = 15$ $p_4/q_4 = 0.75$

$t_1 = \max(0.25, 0.1) = 0.25$ since for these values $p < 0$

$t_2 = \min(0.3, 0.75) = 0.3$ since for these values $p > 0$

Here, $t_1 < t_2$ and the endpoints of clipped line are :

$$\begin{aligned} xx_1 &= x_1 + t_1 \Delta x \\ &= 10 + 0.25 \times 50 \\ &= 22.5 \\ yy_2 &= y_1 + t_1 \Delta y \\ &= 10 + 0.25 \times 20 \\ &= 15 \\ xx_2 &= x_1 + t_2 \Delta x \\ &= 10 + 0.3 \times 50 \\ &= 25 \\ yy_2 &= y_1 + t_2 \Delta y \\ &= 10 + 0.3 \times 20 \\ &= 16 \end{aligned}$$

5.4 Polygon Clipping

In the previous sections we have seen line clipping algorithms. A polygon is nothing but the collection of lines. Therefore, we might think that line clipping algorithm can be used directly for polygon clipping. However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines as shown in the Fig. 5.19. Thus, we need to modify the line clipping algorithm to clip polygons.

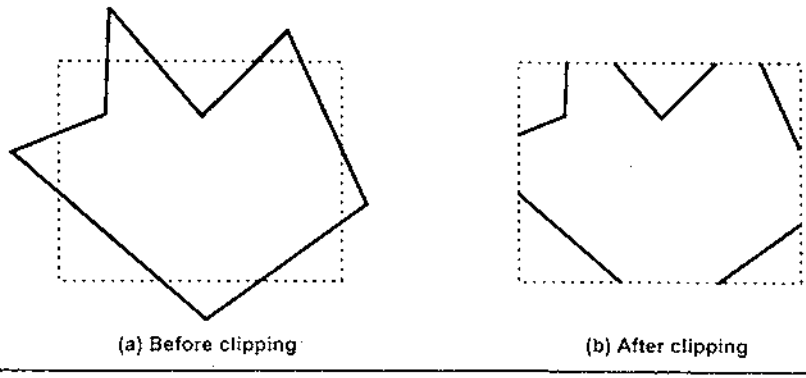


Fig. 5.19 Polygon clipping done by line clipping algorithm

We consider a polygon as a closed solid area. Hence after clipping it should remain closed. To achieve this we require an algorithm that will generate additional line segment which make the polygon as a closed area. For example, in Fig. 5.20 the lines a – b, c – d, d – e, f – g, and h – i are added to polygon description to make it closed.

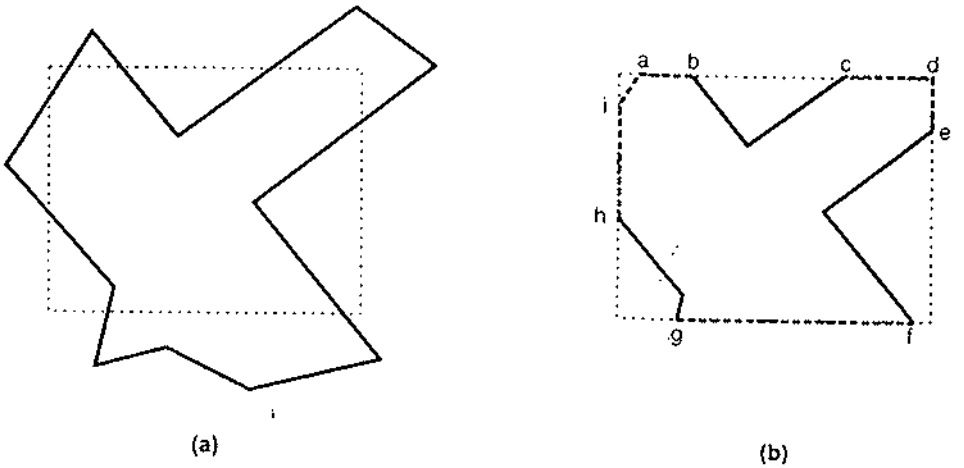


Fig. 5.20 Modifying the line clipping algorithm for polygon

Adding lines c – d and d – e is particularly difficult. Considerable difficulty also occurs when clipping a polygon results in several disjoint smaller polygons as shown in the Fig. 5.21. For example, the lines a – b, c – d, d – e and g – f are frequently included in the clipped polygon description which is not desired.

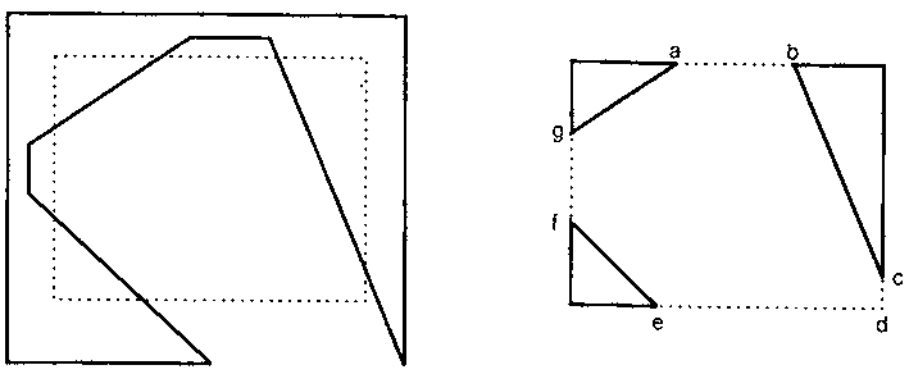


Fig. 5.21 Disjoint polygons in polygon clipping

5.5 Sutherland - Hodgeman Polygon Clipping

A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new

set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper, as shown in Fig. 5.22. At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland - Hodgeman algorithm.

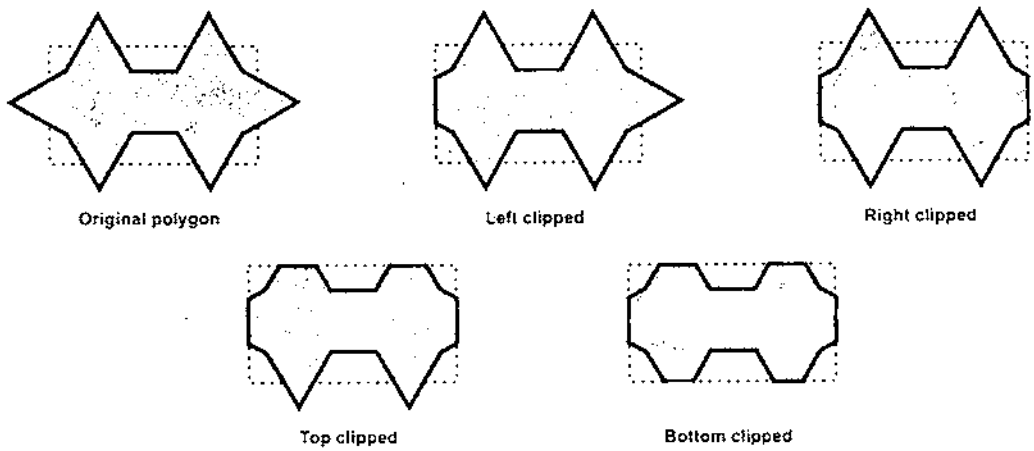


Fig. 5.22 Clipping a polygon against successive window boundaries

The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane. This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This results in four possible relationships between the edge and the clipping boundary or plane. (See Fig. 5.23).

1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (See Fig. 5.23 a).
2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list. (See Fig. 5.23 b).
3. If the first vertex of the edge is inside the window boundary and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (See Fig. 5.23 c).
4. If both vertices of the edge are outside the window boundary, nothing is added to the output list. (See Fig. 5.23 d).

Once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

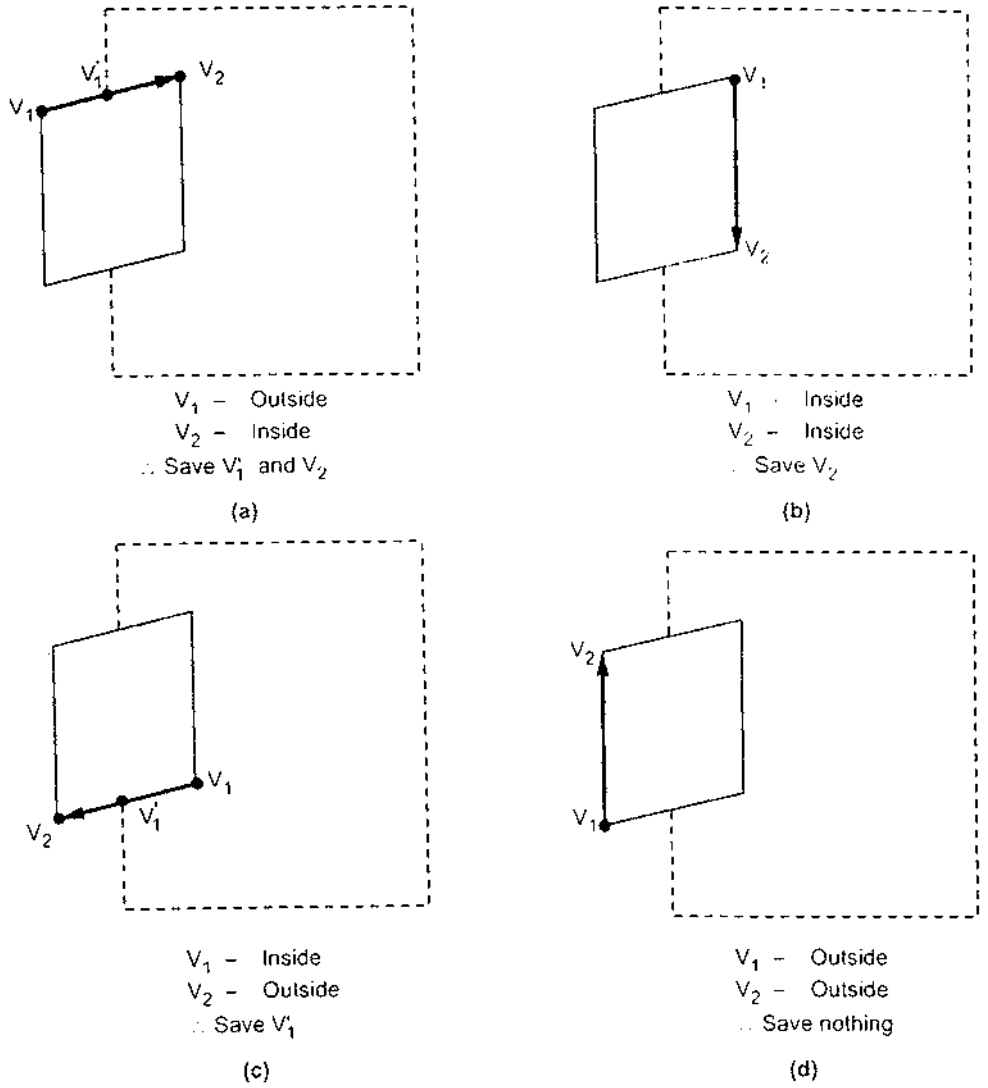


Fig. 5.23 Processing of edges of the polygon against the left window boundary

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

1. Determining the visibility of a point or vertex (Inside - Outside test) and

2. Determining the intersection of the polygon edge and the clipping plane.

One way of determining the visibility of a point or vertex is described here. Consider that two points A and B define the window boundary and point under consideration is V, then these three points define a plane. Two vectors which lie in that plane are AB and AV. If this plane is considered in the xy plane, then the vector cross product AV × AB has only a z component given by $(x_V - x_A)(y_B - y_A) - (y_V - y_A)(x_B - x_A)$. The sign of the z component decides the position of point V with respect to window boundary.

If z is : Positive – Point is on the **right side** of the window boundary

Zero – Point is **on** the window boundary

Negative – Point is on the **left side** of the window boundary

Ex. 5.6 : *Consider the clipping boundary as shown in the Fig. 5.24 and determine the positions of points V_1 and V_2.*

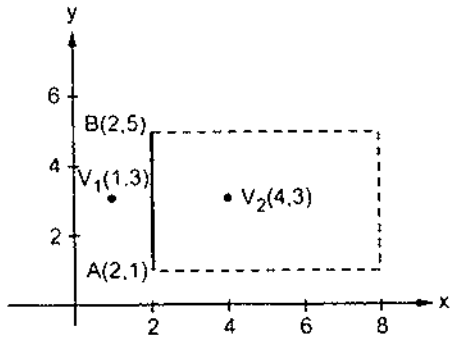


Fig. 5.24

Sol. : Using the cross product for V_1 we get,

$$(x_V - x_A)(y_B - y_A) - (y_V - y_A)(x_B - x_A)$$
$$= (1-2)(5-1) - (3-1)(2-2)$$
$$= (-1)(4) - 0$$
$$= -4$$

The result of the cross product for V_1 is negative hence V_1 is on the left side of the window boundary.

Using the cross product for V_2 we get, $(4-2)(5-1) - (3-1)(2-2)$

$$= (2)(4) - 0$$
$$= 8$$

The result of the cross product for V_2 is positive hence V_1 is on the right side of the window boundary.

The second key process in Sutherland - Hodgeman polygon clipping algorithm is to determine the intersection of the polygon edge and the clipping plane. Any of the line intersection (clipping) techniques discussed in the previous sections such as Cyrus-Beck or mid point subdivision can be used for this purpose.

Sutherland-Hodgeman Polygon Clipping Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane

5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

Ex. 5.7 : *For a polygon and clipping window shown in Fig. 5.25 give the list of vertices after each boundary clipping.*

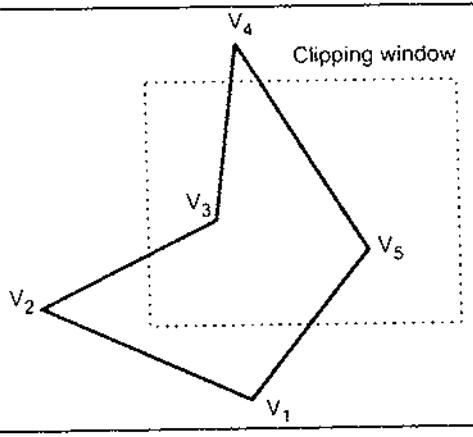


Fig. 5.25

Sol. : Original polygon vertices are V_1, V_2, V_3, V_4, V_5. After clipping each boundary the new vertices are given in Fig. 5.26

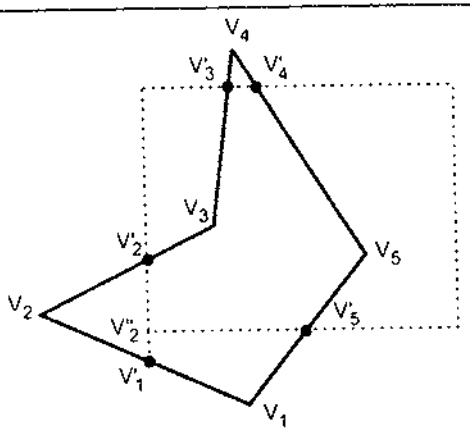


Fig. 5.26

After left clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$

After right clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$

After top clipping : $V_1, V'_1, V'_2, V_3, V'_3, V'_4, V_5$

After bottom clipping : $V''_2, V'_2, V_3, V'_3, V'_4, V_5, V'_5$

'C' code for Sutherland-Hodgeman Polygon Clipping Algorithm

(Softcopy of this program is available at vtubooks.com)

```
#include <stdio.h>
#include<graphics.h>
#include<dos.h>
typedef struct
{
float x;
float y;
}PT;
int n;
main()
{
int i,j,gd,gm;
PT d,p1,p2,p[20],pi1,pi2,pp[20];
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"");

/* Read coordinates of clipping window
---------------------------------------- */
printf("Enter coordinates (left,top) of point1 : ");
scanf("%f,%f",&p1.x,&p1.y);
printf("Enter coordinates (right,bottom) of point2 : ");
scanf("%f,%f",&p2.x,&p2.y);

/* Enter the number of vertex
----------------------------- */
printf("Enter the number of vertex : ");
scanf("%d",&n);

/* Read vertex coordinates of clipping window
---------------------------------------- */
for(i=0;i<n;i++)
{
printf("Enter coordinates of vertex%d : ",i+1);
scanf("%f,%f",&p[i].x,&p[i].y);
}
p[i].x = p[0].x;
```

```
p[i].y = p[0].y;
cleardevice();
drawpolygon(p,n);
rectangle(p1.x,p1.y,p2.x,p2.y);
getch();
left(p1,p,pp);
right(p2,p,pp);
top(p1,p,pp);
bottom(p2,p,pp);
cleardevice();
rectangle(p1.x,p1.y,p2.x,p2.y);
drawpolygon(p,n);
getch();
closegraph();
}

left(PT p1,PT p[20],PT pp[20])
{
int i,j=0;
for(i=0;i<n;i++)
{
 if(p[i].x < p1.x && p[i+1].x >= p1.x)
   {
   if(p[i+1].x-p[i].x!=0)
      {
      pp[j].y = (p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)* (p1.x-p[i].x)+p[i].y;
      }
   else
      {
      pp[j].y = p[i].y;
      }
   pp[j].x = p1.x;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
   }
```

```
  if(p[i].x > p1.x && p[i+1].x >= p1.x)
  {
  pp[j].y = p[i+1].y;
  pp[j].x = p[i+1].x;
  j++;
  }
  if(p[i].x > p1.x && p[i+1].x <= p1.x)
  {
     if(p[i+1].x-p[i].x!=0)
     {
     pp[j].y = (p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)* (p1.x-p[i].x)+p[i].y;
     }
     else
     {
     pp[j].y = p[i].y;
     }
  pp[j].x = p1.x;
  j++;
  }
}
  for(i=0;i<j;i++)
  {
  p[i].x = pp[i].x;
  p[i].y = pp[i].y;
  }
  p[i].x = pp[0].x;
  p[i].y = pp[0].y;
  n=j;
}

right(PT p2,PT p[20],PT pp[20])
{
int i,j=0;
for(i=0;i<n;i++)
{
   if(p[i].x > p2.x && p[i+1].x <= p2.x)
   {
            if(p[i+1].x-p[i].x!=0)
```

```
      {
      pp[j].y = (p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)* (p2.x-p[i].x)+p[i].y;
      }
      else
      {
      pp[j].y = p[i].y;
      }
   pp[j].x = p2.x;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
   }
   if(p[i].x < p2.x && p[i+1].x <= p2.x)
   {
   pp[j].y = p[i+1].y;
   pp[j].x = p[i+1].x;
   j++;
   }
   if(p[i].x < p2.x && p[i+1].x >= p2.x)
   {
       if(p[i+1].x-p[i].x!=0)
      {
      pp[j].y = (p[i+1].y-p[i].y)/(p[i+1].x-p[i].x)* (p2.x-p[i].x)+p[i].y;
      }
      else
      {
      pp[j].y = p[i].y;
      }
   pp[j].x = p2.x;
   j++;
   }
}
   for(i=0;i<j;i++)
   {
   p[i].x = pp[i].x;
   p[i].y = pp[i].y;
   }
```

```
    [illegible]
    [illegible]
    }


[illegible](PT p1,PT p[200],PT pp[200])
{
[illegible]
[illegible]
{

    if(p[i].y > p1.y && p[i+1].y < p1.y)

      if(p[i+1].y-p[i].y!=0)
      {
      pp[j].x = (p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)* (p1.y-p[i].y)+p[i].x;
      [illegible]
      [illegible]
      [illegible]
      [illegible]

    pp[j].y = p1.y;
    j++;
    pp[j].x=p[i+1].x;
    pp[j].y=p[i+1].y;
    [illegible]
    [illegible]

    if(p[i].y < p1.y && p[i+1].y > p1.y)
    {
    pp[j].y = p[i+1].y;
    pp[j].x = p[i+1].x;
    j++;
    }
    if(p[i].y < p1.y && p[i+1].y == p1.y)
    {
      if(p[i+1].y-p[i].y!=0)
      {
      pp[j].x = (p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)* (p1.y-p[i].y)+p[i].x;
      }
```

```
      else
      {
      pp[j].x = p[i].x;
      }
   pp[j].y = pi.y;
   j++;
   }
}
   for(i=0;i<j;i++)
   {
   p[i].x = pp[i].x;
   p[i].y = pp[i].y;
   }
   p[i].x = pp[0].x;
   p[i].y = pp[0].y;
   n=j;
}
bottom(PT p2,PT p[20],PT pp[20])
{
int i,j=0;
for(i=0;i<n;i++)
{
   if(p[i].y > p2.y && p[i+1].y <= p2.y)
   {
      if(p[i+1].y-p[i].y!=0)
      {
      pp[j].x = (p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)* (p2.y-p[i].y)+p[i].x;.
      }
      else
      {
      pp[j].x = p[i].x;
      }
   pp[j].y = p2.y;
   j++;
   pp[j].x=p[i+1].x;
   pp[j].y=p[i+1].y;
   j++;
   }
```

```
    if(p[i].y < p2.y && p[i+1].y <= p2.y)
    {
    pp[j].y = p[i+1].y;
    pp[j].x = p[i+1].x;
    j++;
    }
    if(p[i].y < p2.y && p[i+1].y >= p2.y)
    {
       if(p[i+1].y-p[i].y!=0)
       {
       pp[j].x = (p[i+1].x-p[i].x)/(p[i+1].y-p[i].y)* (p2.y-p[i].y)+p[i].x;
       }
       else
       {
       pp[j].x = p[i].x;
       }
    pp[j].y = p2.y;
    j++;
    }
  }
    for(i=0;i<j;i++)
    {
    p[i].x = pp[i].x;
    p[i].y = pp[i].y;
    }
    p[i].x = pp[0].x;
    p[i].y = pp[0].y;
    n=j;
  }
  drawpolygon(PT x[20],int n)
  {
  int i;
  for(i=0;i<n-1;i++)
     {
      line(x[i].x,x[i].y,x[i+1].x,x[i+1].y);
     }
      line(x[i].x,x[i].y,x[0].x,x[0].y);
  }
```

The Sutherland-Hodgeman polygon clipping algorithm clips convex polygons correctly, but in case of concave polygons, clipped polygon may be displayed with extraneous lines, as shown in Fig. 5.27.

Fig. 5.27 Clipping the can cave polygon in (a) with the Sutherland-Hodgeman algorithm produces the two connected areas in (b)

The problem of extraneous lines for concave polygons in Sutherland-Hodgeman polygon clipping algorithm can be solved by separating concave polygon into two or more convex polygons and processing each convex polygon separately.

5.6 Weiler-Atherton Algorithm

The clipping algorithms previously discussed require a convex polygon. In context of many applications, e.g., hidden surface removal, the ability to clip to concave polygon is required. A powerful but somewhat more complex clipping algorithm developed by Weiler and Atherton meets this requirement. This algorithm defines the polygon to be clipped as a **subject polygon** and the clipping region is the clip polygon.

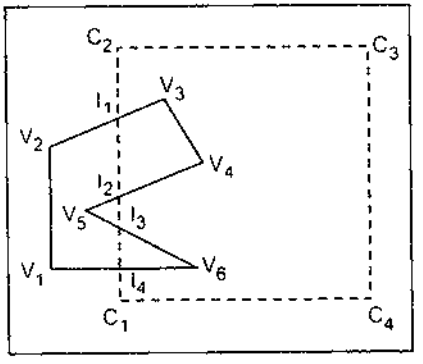


Fig. 5.28

The algorithm describes both the subject and the clip polygon by a circular list of vertices. The boundaries of the subject polygon and the clip polygon may or may not intersect. If they intersect, then the intersections occur in pairs. One of the intersections occurs when a subject polygon edge enters the inside of the clip polygon and one when it leaves. As shown in the Fig. 5.28, there are four intersection vertices I_1, I_2 , I_3 and I_4. In these intersections I_1 and I_3 are entering intersections, and I_2 and I_3 are leaving intersections. The clip polygon vertices are marked as C_1, C_2, C_3 and C_4.

In this algorithm two separate vertices lists are made one for clip polygon and one for subject polygon including intersection points. The Table 5.3 shows these two lists for polygons shown in Fig. 5.28.

For subject polygon		For clip polygon	
	V_1	C_1	
	V_2	I_4	
Start	I_1	I_3	Finish
	V_3	I_2	
	V_4	I_1	Finish
	I_2	C_2	
	V_5	C_3	
Start	I_3	C_4	
	V_6		
	I_4		
	V_1		

Table 5.3 List of polygon vertices

The algorithm starts at an entering intersection (I_1) and follows the subject polygon vertex list in the downward direction (i.e. I_1, V_3, V_4, I_2). At the occurrence of leaving intersection the algorithm follows the clip polygon vertex list from the leaving intersection vertex in the downward direction (i.e. I_2, I_1). At the occurrence of the entering intersection the algorithm follows the subject polygon vertex list from the entering intersection vertex. This process is repeated until we get the starting vertex. This process we have to repeat for all remaining entering intersections which are not included in the previous traversing of vertex list. In our example, entering vertex I_3 was not included in the first traversing of vertex list,. Therefore, we have to go for another vertex traversal from vertex I_3.

The above two vertex traversals gives two clipped inside polygons. There are :

I_1, V_3, V_4, I_2, I_1 and I_3, V_6, I_4, I_3

5.7 Generalized Clipping

We have seen that in Sutherland - Hodgeman polygon clipping algorithm we need separate clipping routines, one for each boundary of the clipping window. But these routines are almost identical. They differ only in their test for determining whether a point is inside or outside the boundary. It is possible to generalize these routines so that they will be exactly identical and information about the boundary is passed to the routines through their parameters. Using recursive technique the generalized routine can be 'called' for each boundary of the clipping window with a different boundary specified by its parameters. This form of algorithm allows us to have any number of boundaries to the clipping window, thus the generalized algorithm with recursive technique can be used to clip a polygon along an arbitrary convex clipping window.

5.8 Interior and Exterior Clipping

So far we have discussed only algorithms for clipping point, line and polygon to the interior of a clipping region by eliminating every thing outside the clipping region. However, it is also possible to clip a point, line or polygon to the exterior of a clipping region, i.e., the point, portion of line and polygon which lie outside the clipping region. This is referred to as **exterior clipping**.

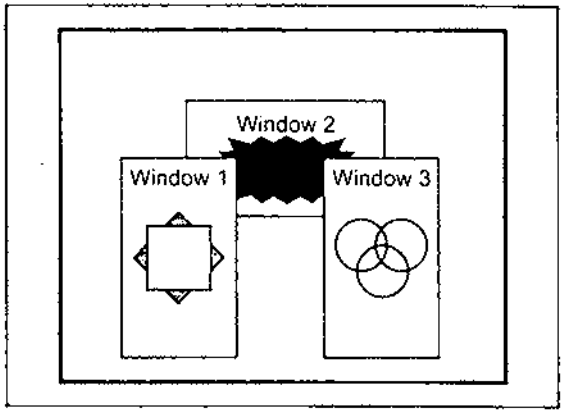


Fig. 5.29 Clipping in multiwindow environment

Exterior clipping is important in a multiwindow display environment, as shown in Fig. 5.29. The Fig. 5.29 shows the overlapping windows with window 1 and window 3 having priority over window 2. The objects within the window are clipped to the interior of that window. When other higher-priority windows such as window 1 and/or window 3 overlap these objects, the objects are also clipped to the exterior of the overlapping windows.

Solved Examples

Ex. 5.8 : *Use the Cohen-Sutherland outcode algorithm to clip two lines P_1 (40, 15) – P_2 (75, 45) and P_3 (70, 20) – P_4 (100, 10) against a window A (50, 10), B (80, 10), C (80, 40), D(50,40).*

Sol. : **Line 1 :** P_1 (40, 15) P_2 (75, 45) $W_{x1} = 50$ $W_{y1} = 40$
$W_{x2} = 80$ $W_{y2} = 10$

Point	Endcode	ANDing	
P_1	0001	0000	Partially visible
P_2	0000		

$$y_1 = m(x_L - x) + y = \frac{6}{7}(50 - 40) + 15 \quad m = \frac{45-15}{75-40} = \frac{6}{7}$$

$$= 23.57$$

$$x_1 = \frac{1}{m}(y_T - y) + x \Rightarrow \frac{7}{6}(40 - 15) + 40 = 69.16$$

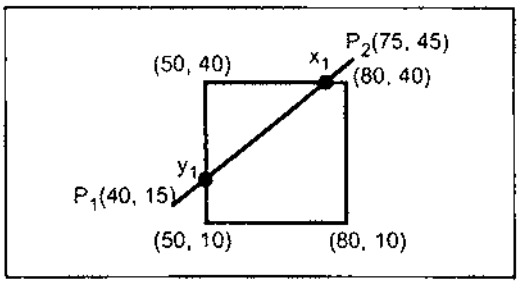


Fig. 5.30

$$y_2 = m(x_R - x) + y$$
$$= \frac{6}{7}(80 - 40) + 15$$
$$= 49.28$$
$$x_2 = \frac{1}{m}(y_B - y) + x$$
$$= \frac{7}{6}(10 - 15) + 40$$
$$= 34.16$$

Line 2 : P_3 (70, 20) P_4 (100, 10)

Point	End code	ANDing	Position
P_3	0 0 0 0	0 0 0 0	Partially visible
P_4	0 0 1 0		

$$\text{Slope } m' = \frac{10-20}{100-70} = \frac{-10}{30} = \frac{-1}{3}$$

$$y_1' = m(x_L - x) + y = \frac{-1}{3}(50-70) + 20$$

$$= 26.66$$

$$x_1' = \frac{1}{m}(y_T - y) + x = -3(40-20) + 70$$

$$= 10$$

$$y_2' = m(x_R - x) + y = \frac{-1}{3}(80-70) + 20$$

$$= 16.66$$

$$x_2' = \frac{1}{m}(y_B - y) + x = -3(10-20) + 70$$

$$= 100$$

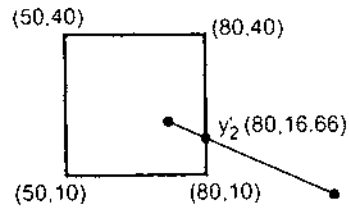


Fig. 5.31

Ex. 5.9 : *Find the normalization transformation window to viewport, with window, lower left corner at (1, 1) and upper right corner at (3, 5) onto a viewport, for entire normalized device screen.*

Sol. : $x_{w\,min} = 1$ $y_{w\,min} = 1$

$x_{w\,max} = 3$ $y_{w\,max} = 5$

Entire normalized screen

$$x_{v\,min} = 0 \qquad y_{v\,min} = 0$$

$$x_{v\,max} = 1 \qquad y_{v\,max} = 1$$

$$S_x = \frac{x_{v\,max} - x_{v\,min}}{x_{w\,max} - x_{w\,min}}$$

$$= \frac{1-0}{3-1} = \frac{1}{2}$$

$$S_y = \frac{y_{v\,max} - y_{v\,min}}{y_{w\,max} - y_{w\,min}}$$

$$= \frac{1-0}{5-1}$$

$$= \frac{1}{4}$$

Transformation matrix is given as

$$T.S.T^{-1} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v\,min} - x_{w\,min}S_x & y_{v\,min} - y_{w\,min}S_y & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.25 & 0 \\ -0.5 & -0.25 & 1 \end{bmatrix}$$

Review Questions

1. What is windowing and clipping ?
2. Write a short note on viewing transformation.
3. Distinguish between viewport and window.
4. What do you mean by normalization transformation ? Why it is needed ?
5. Derive the transformation matrix for 2-D viewing transformation.
6. What is point clipping and line clipping ?
7. Explain the Sutherland and Cohen subdivision algorithm for line clipping.
8. Explain the mid-point subdivision method of clipping a line segment.
9. Explain the Cyrus-Beck algorithm for generalized line clipping.
10. Explain Liang-Barsky line clipping algorithm.
11. What is polygon clipping ?
12. Explain Sutherland - Hodgeman algorithm for polygon clipping ?
13. What are the limitations of Sutherland-Hodgeman polygon clipping algorithm ?
14. Explain Weiler-Atherton polygon clipping algorithm and state its advantage over Sutherland-Hodgeman polygon clipping algorithm.
15. Write a short note on generalized clipping.

University Questions

1. Explain the Cohen-Sutherland techniques for line clipping. **(Dec-96)**
2. Develop a PASCAL program/C program to clip a polygon against a rectangular window inclined at an angle θ to the x-axis. **(Dec-96)**
3. What do you understand by the terms "window" and "viewpoint". Derive the mapping for any given point (x_w, y_w) from the window onto the viewpoint. **(May-97)**

4. Discuss the algorithm and develop a program for polygon clipping. Illustrate the working of your program for any sample polygon of your choice. (May-97)
5. Define the terms world-co-ordinates, device co-ordinates, normalised co-ordinates and homogeneous co-ordinates. (May-97, May-2000)
6. Figure below depicts a picture in the "window". For the "view port" shown alongside evaluate and draw the mapped picture. (Dec-97)

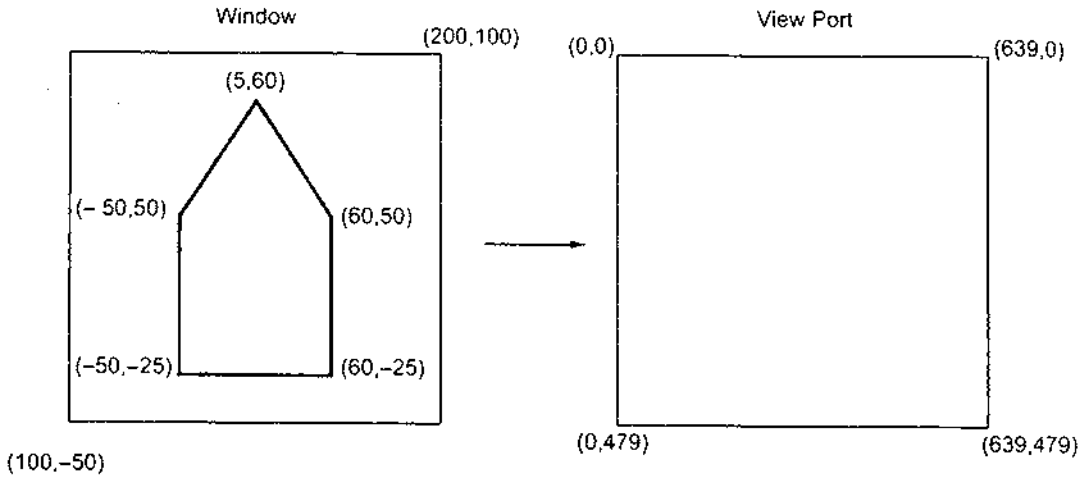


Fig. 5.32

(Note : The picture is not to scale)

7. Write an line clipping algorithm which uses parametric form of equations. Test it for a line P_1, P_2 where $P_1 \equiv (10, 10)$ and $P_2 \equiv (60, 30)$, against window with $(x_{wmin}, y_{wmin}) \equiv (15, 15)$ and $(x_{wmax}, y_{wmax}) \equiv (25, 25)$. (May-98)
8. Figure below shows a window (A, B, C, D) and a viewport (L, M, N, O). Show how the window and object in it, is mapped to viewport. (May-98)

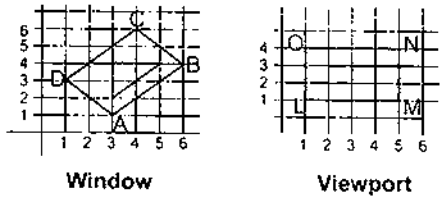


Fig. 5.33

9. Develop an function/procedure which performs line clipping using Cohen-Sutherland method. How the line between (2, 2) and (12, 9) is clipped against window with $(x_{wmin}, y_{wmin}) = (4, 4)$ and $(x_{wmax}, y_{wmax}) \equiv (9, 8)$. (Dec-98, May-2002)
10. Develop formulae for window to view port mapping in 2D. Write a routine to map a POINT (X1, Y1) (May-99)
11. Indicate a mechanism to map an elliptical window to circular view port. Hint : map centre of ellipse to centre of circle. (May-99)

12. Explain Sutherland - Hodgman clipping algorithm with example. **(May-99)**
13. What is the use of Normalised Device Co-ordinates. **(May-99)**
14. Explain why the Sutherland - Hodgman polygon - clipping algorithm works for only convex clipping regions. **(Dec-99)**
15. Explain Cohen-Sutherland line clipping algorithm. Develop a program to clip a line between (x_1, y_1) (x_2, y_2) against a window (x_{min}, y_{min}) (x_{max}, y_{max}) **(May-2000)**
16. Give Liang Barsky line clipping algorithm. **(May-2001, May-2003)**
17. Using Liang Barsky line clipping algorithm find the clipping co-ordinates of line segment with end co-ordinates A(– 10, 50) and B(30, 80) against window

    $(x_{wmin} = -30, y_{wmin} = 10)$ $(x_{wmax} = 20, y_{wmax} = 60)$. **(May-2001)**
18. Suggest modification to Sutherland Hodgman polygon clipping algorithm to clip concave polygon. **(May-2001)**
19. What do you understand by the terms WCS, DCS, NDS, WINDOW and VIEWPORT. Derive the mapping for any given point P(xw, yw) from window to viewpoint using matrix method. Window is defined in right handed world coordinate system and view port is defined in left handed device coordinate system. **(May-2001)**

❑❑❑

6

3-D Concepts

6.1 Introduction

Manipulation viewing and construction of three dimensional graphic image requires the use of three dimensional geometric and coordinate transformations. Three dimensional geometric transformations are extended from two-dimensional methods by including considerations for the z coordinate. Like two dimensional transformations, these transformations are formed by composing the basic transformations of translation, scaling, and rotation. Each of these transformations can be represented as a matrix transformation with homogeneous coordinates. Therefore, any sequence of transformations can be represented as a single matrix, formed by combining the matrices for the individual transformations in the sequence.

6.2 Translation

Three dimensional transformation matrix for translation with homogeneous coordinates is as given below. It specifies three coordinates with their own translation factor.

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

$$\therefore \quad P' = P \cdot T$$

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

$$= [x + t_x \quad y + t_y \quad z + t_z \quad 1] \qquad \ldots (6.1)$$

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

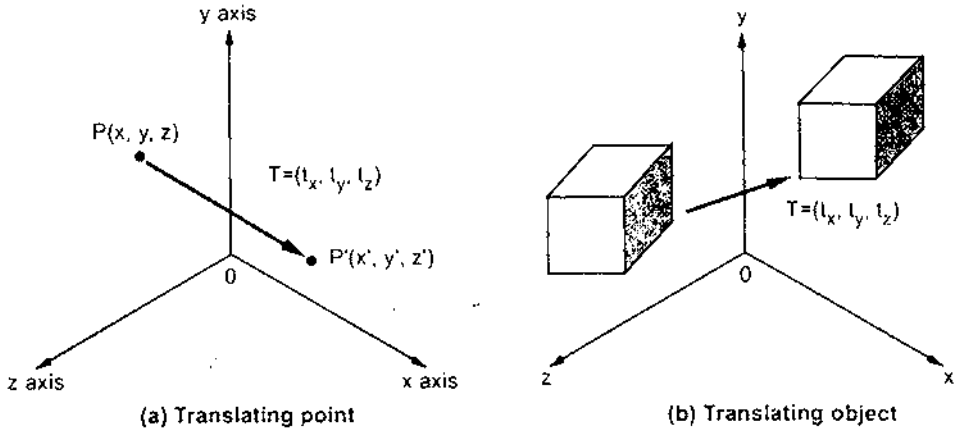


Fig. 6.1 3 D translation

6.3 Scaling

Three dimensional transformation matrix for scaling with homogeneous coordinates is as given below.

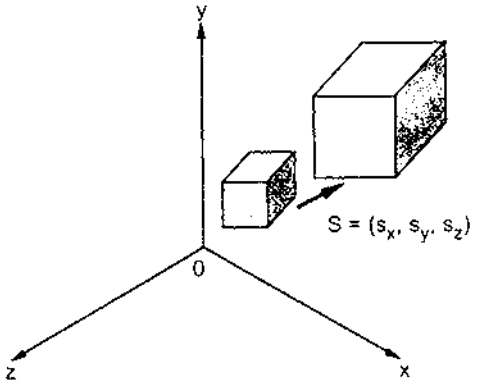


Fig. 6.2 3 D Scaling

It specifies three coordinates with their own scaling factor.

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

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

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

$$= \begin{bmatrix} x \cdot S_x & y \cdot S_y & z \cdot S_z & 1 \end{bmatrix} \quad \text{... (6.2)}$$

A scaling of an object with respect to a selected fixed position can be represented with the following transformation sequence.

1. Translate the fixed point to the origin.
2. Scale the object
3. Translate the fixed point back to its original position.

6.4 Rotation

Unlike two dimensional rotation, where all transformations are carried out in the xy plane, a three-dimensional rotation can be specified around any line in space. Therefore, for three dimensional rotation we have to specify an axis of rotation about which the object is to be rotated alongwith the angle of rotation. The easiest rotation axes to handle are those that are parallel to the coordinate axes. It is possible to combine the coordinate axis rotations to specify any general rotation.

Coordinate Axes Rotations

Three dimensional transformation matrix for each coordinate axes rotations with homogeneous coordinate are as given below

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

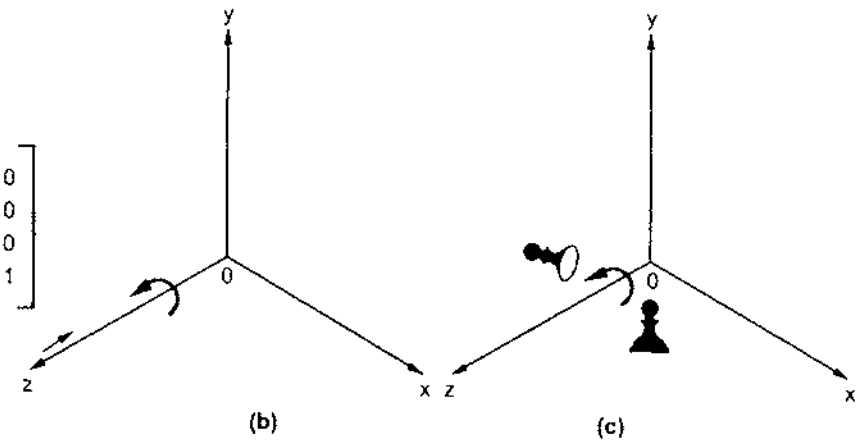


Fig. 6.3 Rotation about z axis

The positive value of angle θ indicates counterclockwise rotation. For clockwise rotation value of angle θ is negative.

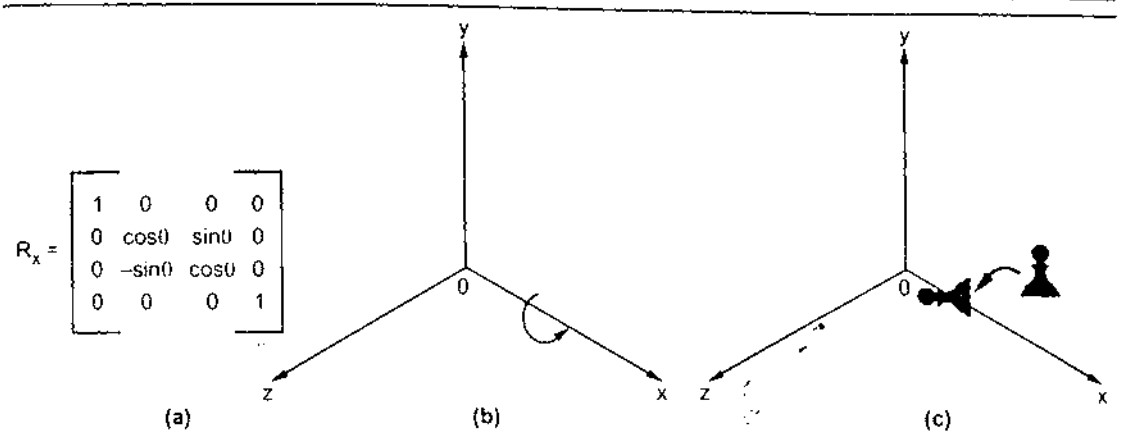


Fig. 6.4 Rotation about x axis

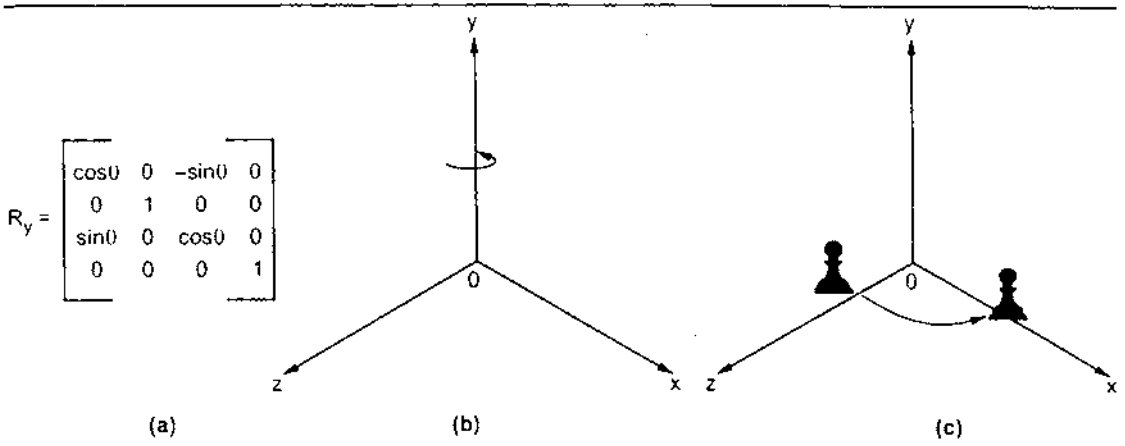


Fig. 6.5 Rotation about y axis

6.5 Rotation about Arbitrary Axis

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

In a special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes we can obtain the resultant coordinates with the following transformation sequence.

1. Translate the object so that the rotation axis coincides with the parallel coordinate axis.
2. Perform the specified rotation about that axis.
3. Translate the object so that the rotation axis is moved back to its original position.

When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we have to perform some additional transformations. The sequence of these transformations is given below.

1. Translate the object so that rotation axis specified by unit vector u passes through the coordinate origin. (see Fig. 6.6 (a) and (b))
2. Rotate the object so that the axis of rotation coincides with one of the coordinate axes. Usually the z axis is preferred. To coincide the axis of rotation to z axis we have to first perform rotation of unit vector u about x axis to bring it into xz plane and then perform rotation about y axis to coincide it with z axis. (see Figs. 6.6 (c) and (d))
3. Perform the desired rotation θ about the z axis.
4. Apply the inverse rotation about y axis and then about x axis to bring the rotation axis back to its original orientation.
5. Apply the inverse translation to move the rotation axis back to its original position.

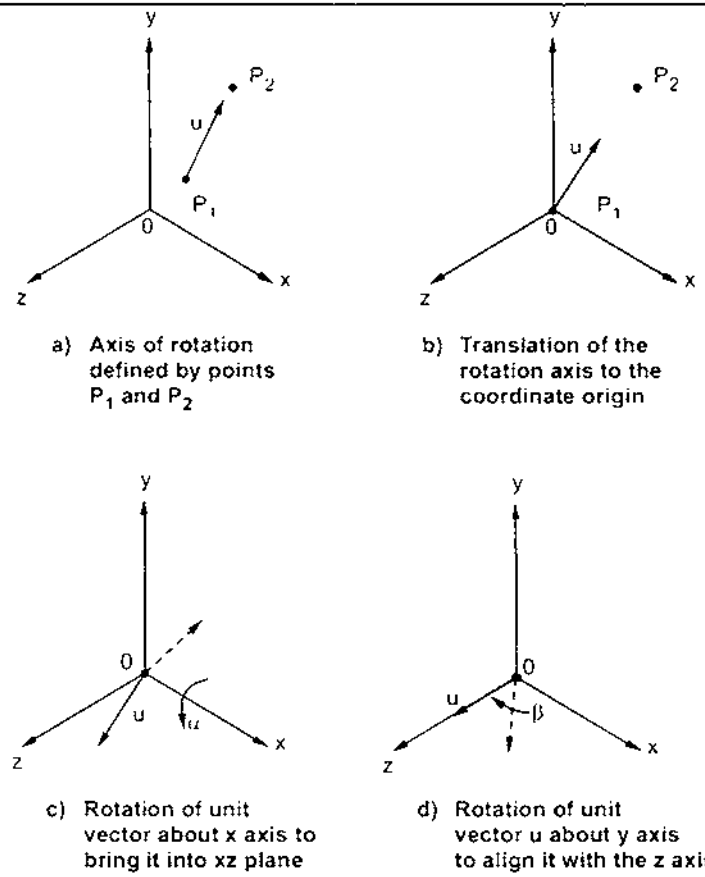


Fig. 6.6

As shown in the Fig. 6.6 (a) the rotation axis is defined with two coordinate points P_1 and P_2 and unit vector u is defined along the rotation of axis as

$$u = \frac{V}{|V|} = (a, b, c)$$

where V is the axis vector defined by two points P_1 and P_2 as

$$V = P_2 - P_1$$
$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

The components a, b, and c of unit vector u are the direction cosines for the rotation axis and they can be defined as

$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$

As mentioned earlier, the first step in the transformation sequence is to translate the object to pass the rotation axis through the coordinate origin. This can be accomplished by moving point P_1 to the origin. The translation is as given below

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

Now we have to perform the rotation of unit vector u about x axis. The rotation of u around the x axis into the xz plane is accomplished by rotating u′ (0, b, c) through angle α into the z axis and the cosine of the rotation angle α can be determined from the dot product of u′ and the unit vector u_z (0, 0, 1) along the z axis.

$$\cos\alpha = \frac{u' \cdot u_z}{|u'||u_z|} = \frac{c}{d} \quad \text{where } u'(0, b, c) = bJ + cK \text{ and}$$

$$u_z(0, 0, 1) = K$$

$$= \frac{c}{|u'||u_z|}$$

$$= \frac{c}{|u'|} \qquad \text{Since } |u_z| = 1$$

$$= \frac{c}{d}$$

where d is the magnitude of u′ :

$$d = \sqrt{b^2 + c^2}$$

Similarly, we can determine the sine of α from the cross product of u′ and u_z.

$$u' \times u_z = u_x \, |u'| \, |u_z| \sin\alpha \qquad \ldots (6.3)$$

and the Cartesian form for the cross product gives us

$$u' \times u_z = u_x \cdot b \qquad \ldots (6.4)$$

Equating the right sides of equations 6.3 and 6.4 we get

$$u_x \, |u'| \, |u_z| \sin\alpha = u_x \cdot b$$

$$\therefore \quad |u'| \, |u_z| \sin\alpha = b$$

$$\therefore \quad \sin\alpha = \frac{b}{|u'| \, |u_z|}$$

$$= \frac{b}{d} \qquad \text{since } |u_z| = 1 \text{ and } |u'| = d$$

This can also be verified graphically as shown in Fig. 6.7

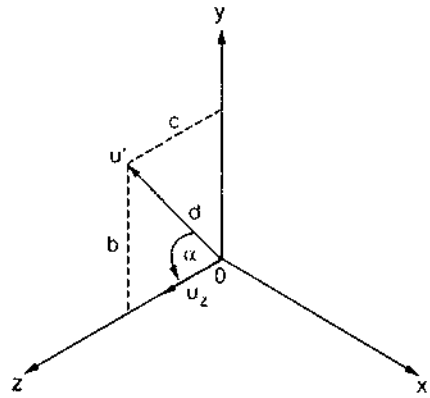


Fig. 6.7

By substituting values of $\cos\alpha$ and $\sin\alpha$ the rotation matrix R_x can be given as

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next we have to perform the rotation of unit vector about y axis. This can be achieved by rotating u" (a, 0, d) through angle β onto the z axis. Using similar equations we can determine $\cos\beta$ and $\sin\beta$ as follows.

We have angle of rotation $= -\beta$

$$\therefore \quad \cos(-\beta) = \cos\beta = \frac{u'' \cdot u_z}{|u''| \, |u_z|} \text{ where } u'' = aI + dK \text{ and}$$

$$u_z = K$$

$$= \frac{d}{|u''||u_z|}$$

$$= \frac{d}{|u''|} \qquad \because |u_z| = 1$$

$$= \frac{d}{\sqrt{a^2 + d^2}}$$

Consider cross product of u'' and u_z.

$$u'' \times u_z = u_y |u''| |u_z| \sin(-\beta)$$

$$= -u_y |u''| |u_z| \sin\beta \qquad \because \sin(-\theta) = -\sin\theta$$

Cartesian form of cross product gives us

$$u'' \times u_z = u_y (+a)$$

Equating above equations,

$$-|u''| |u_z| \sin\beta = a$$

$$\therefore \quad \sin\beta = \frac{-a}{|u''||u_z|}$$

$$= \frac{-a}{|u''|} \qquad \because |u_z| = 1$$

$$= \frac{-a}{\sqrt{a^2 + d^2}}$$

but we have, $\quad d = \sqrt{b^2 + c^2}$

$$\therefore \quad \cos\beta = \frac{d}{\sqrt{a^2 + d^2}}$$

$$= \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}}$$

and $\quad \sin\beta = \dfrac{-a}{\sqrt{a^2 + d^2}}$

$$= \frac{-a}{\sqrt{a^2 + b^2 + c^2}}$$

By substituting values of $\cos\beta$ and $\sin\beta$ in the rotation matrix R_y can be given as

$$R_y = \begin{bmatrix} \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 & \frac{+a}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-a}{\sqrt{a^2+b^2+c^2}} & 0 & \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Let $\lambda = \sqrt{b^2+c^2}$ and $|V| = \sqrt{a^2+b^2+c^2}$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\lambda} & \frac{+b}{\lambda} & 0 \\ 0 & \frac{-b}{\lambda} & \frac{c}{\lambda} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{+a}{|V|} & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-a}{|V|} & 0 & \frac{\lambda}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\therefore$ Resultant rotation matrix $R_{xy} = R_x \cdot R_y$

$$\therefore \quad R_{xy} = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{|V|\lambda} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{|V|\lambda} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We have,

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T}$$

Using above equation we get inverse of R_{xy} as

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{|V|\lambda} & \frac{-ac}{|V|\lambda} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse of translation matrix can be given as

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

With transformation matrices T and R_{xy} we can align the rotation axis with the positive z axis. Now the specified rotation with angle θ can be achieved by rotation transformation as given below

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

To complete the required rotation about the given axis, we have to transform the rotation axis back to its original position. This can be achieved by applying the inverse transformations T^{-1} and R_{xy}^{-1}. The overall transformation matrix for rotation about an arbitrary axis then can be expressed as the concatenation of five individual transformations.

$$R(\theta) = T \cdot R_{xy} \cdot R_z \cdot R_{xy}^{-1} \cdot T^{-1}$$

i.e.

$$R(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{\lambda|V|} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{\lambda|V|} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{\lambda|V|} & \frac{-ac}{\lambda|V|} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{bmatrix}$$

6.6 Reflection with Respect to Given Plane

6.6.1 Reflection with Respect to xy Plane

Consider point P(x, y, z). The reflection of this point with respect to xy plane is given by ponit P'(x, y, – z), as shown in Fig. 6.8. Corresponding to this reflection the transformation matrix can be given as

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

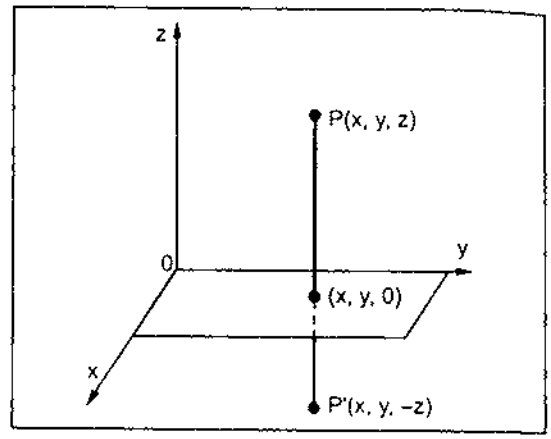


Fig. 6.8

6.6.2 Reflection with Respect to Any Plane

Often it is necessary to reflect an object through a plane other than x = 0 (yz plane), y = 0 (xz plane) or z = 0 (xy plane). Procedure to achieve such a reflection (reflection with respect to any plane) can be given as follows :

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

Let P_o (x_o, y_o, z_o) be the given known point. Translate this point to the origin by using corresponding translation matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_o & -y_o & -z_o & 1 \end{bmatrix}$$

Let the normal vector $N = n_1 I + n_2 J + n_3 K$

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

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

As we want to match this vector with z axis, (so that the plane of reflection will be parallel to xy plane), we will use the same procedure as used in rotation.

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

As seen earlier for reflection about xy plane we have

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

Now for inverse transformation we have,

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_o & y_o & z_o & 1 \end{bmatrix}$$

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

$\therefore$ Resultant transformation matrix can be given as

$$R_T = T \cdot R_{xy} \cdot M \cdot R_{xy}^{-1} \cdot T^{-1}$$

Ex. 6.1 : *Find the matrix for mirror reflection with respect to the plane passing through the origin and having a normal vector whose direction is $M = I + J + K$*

Sol. : Here, P_o (0, 0, 0) and the plane passes through the origin hence translation matrix is not necessary.

The normal vector $N = I + J + K$

$$\therefore \quad n_1 = 1, \quad n_2 = 1, \quad n_3 = 1$$

$$|N| = \sqrt{3} \quad \text{and} \quad \lambda = \sqrt{2}$$

$$\therefore \quad R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{xy}^{-1} = \begin{bmatrix} \sqrt{\frac{2}{3}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{+1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$\therefore$ The reflection matrix is given by

$$R_T = R_{xy} \cdot M \cdot R_{xy}^{-1}$$

$$\therefore \quad R_T = \begin{bmatrix} 1/3 & -2/3 & -2/3 & 0 \\ -2/3 & 1/3 & -2/3 & 0 \\ -2/3 & -2/3 & +1/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Solved Examples

Ex. 6.2 : *Obtain transformation matrix for rotation about the line joining the points (0, 0, 0) and (1, 1, 1) with the angle of rotation 45° in counter-clockwise sense.*

Sol. : In this case the line passes through the origin, so the translation is not required. Therefore, R_T can be given

$$R_T = R_{xy} \cdot R \cdot R_{xy}^{-1}$$

by usual notations,

$$\lambda = \sqrt{1+1} = \sqrt{2}$$

$$|V| = \sqrt{1+1+1} = \sqrt{3}$$

Here, a = 1, b = 1, c = 1. By using derived rotation matrices for R_{xy}, R and R_{xy}^{-1} from section 6.5 we have

$$R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad R_T = R_{xy} \cdot R \cdot R_{xy}^{-1}$$

$$= \begin{bmatrix} 0.80473 & 0.5058 & -0.3106 & 0 \\ -0.3106 & 0.80473 & 0.5058 & 0 \\ 0.5058 & -0.3106 & 0.80473 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ex. 6.3 : *A triangle is defined by 3 vertices A (0, 2, 1) B, (2, 3, 0), C (1, 2, 1). Find the final co-ordinates after it is rotated by 45° around a line joining the points (1, 1, 1) and (0, 0, 0).*

Sol. : The required transformation matrix for this example is already obtained in previous example.

$$\therefore \quad \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot R_r$$

$$= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.804 & 0.505 & -0.3106 & 0 \\ -0.3106 & 0.804 & 0.505 & 0 \\ 0.5058 & -0.3106 & 0.804 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.116 & 1.297 & -1.814 & 0 \\ 0.676 & 3.422 & 0.893 & 0 \\ 0.687 & 1.802 & 0.998 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Therefore final co-ordinates are : A′ (-0.116, 1.297, – 1.814)
B′ (0.676, 3.422, 0.893)
C′ (0.687, 1.802, 0.998)

Ex. 6.4 : *A triangle is defined by 3 vertices A (0, 2, 1) B (2, 3, 0), C (1, 2, 1). Find the final co-ordinates after it is rotated by 45° around a line joining the points (2, 2, 2) and (1, 1, 1).*

Sol. : Here the given axis of rotation is not at the origin, therefore translation matrix is required. The translation matrix can be given as :

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

Therefore the inverse of translation matrix can be obtained as

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

It can be seen that after translating the given axis to the origin we get line points as (1, 1, 1) and (0, 0, 0) which are same as the line points considered in the previous example. The rotation angle (45°) in this example also matches with that in the previous example. Therefore we can use resultant matrix of the previous problem, in deriving the transformation matrix for the given problem. The transformation matrix for this problem can be given as

$$R_1 = T \cdot R_{xy} \cdot R \cdot R_{xy}^{-1} \cdot T^{-1}$$

Substituting resultant matrix from the previous problem we have,

$$R_1 = \begin{bmatrix} 0.8047 & 0.505 & -0.310 & 0 \\ -0.3106 & 0.8047 & 0.505 & 0 \\ 0.505 & -0.3106 & 0.80473 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0.804 & 0.505 & -0.310 & 0 \\ -0.310 & 0.804 & 0.505 & 0 \\ 0.505 & -0.310 & 0.804 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8047 & 0.505 & -0.3106 & 0 \\ -0.3106 & 0.8047 & 0.505 & 0 \\ 0.505 & -0.3106 & 0.8047 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot R_Y$$

$$= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8047 & 0.505 & -0.3106 & 0 \\ -0.3106 & 0.8047 & 0.505 & 0 \\ 0.505 & -0.3106 & 0.8047 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Calculation part is left for the student as an exercise.

Ex. 6.5 *A cube defined by 8 vertices*

A (0, 0, 0) *B (2, 0, 0)* *C (2, 2, 0)* *D (0, 2, 0)*

E (0, 0, 2) *F (2, 0, 2)* *G (2, 2, 2)* *H (0, 2, 2)*

Find the final co-ordinates after it is rotated by 45° around a line joining the points (2, 0, 0) and (0, 2, 2).

Sol : Let P (2, 0, 0) and Q (0, 2, 2)

Step 1 : Shifting the arbitrary axis to origin by using translational matrix.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 1 \end{bmatrix}$$

Step 2 : Finding inverse translation matrix

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix}$$

Step 3 : Finding matrix for coinciding the given axis with the z axis

Here, $a = 2, \quad b = -2, \quad c = -2$

by usual notations, $\lambda = \sqrt{b^2 + c^2}$

$$|V| = \sqrt{a^2 + b^2 + c^2}$$

$$\therefore \quad \lambda = \sqrt{4+4}$$

$$= \sqrt{8}$$

$$= 2\sqrt{2}$$

$$|V| = \sqrt{2^2 + (-2)^2 + (-2)^2}$$

$$= \sqrt{12}$$

$$= 2\sqrt{3}$$

By using derived rotation matrix for R_{xy}

$$R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{-1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4 : Finding R_{xy}^{-1}

By using derived rotation matrix for R_{xy}^{-1}

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 5 : Matrix for rotation about z axis. Here, $\theta = 45°$

$$\therefore \quad R = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$\therefore$ Resultant transformation matrix can be given as

$$R_T = T \cdot R_{xy} \cdot R \cdot R_{xy}^{-1} \cdot T^{-1}$$

From the resultant matrix the final co-ordinates of the cube can be obtained as

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{bmatrix} \cdot R_1$$

Calculation part is left for the student as an exercise.

Ex. 6.6 : *A mirror is placed vertically such that it passes through the points (10, 0) and (0, 10). Find the reflected view of triangle ABC with co-ordinates A(5, 50), B(20, 40), C(10, 70)*

Sol. : The Fig. 6.9 shows the representation of reflection plane. As shown in the Fig. 6.9.

intercept on x axis : 10

intercept on y axis : 10

intercept on z axis : ∞

$\therefore$ The equation of the plane using intercept form can be given as

$$\frac{x}{10} + \frac{y}{10} = 1$$

$$\therefore \quad \frac{x}{10} + \frac{y}{10} - 1 = 0$$

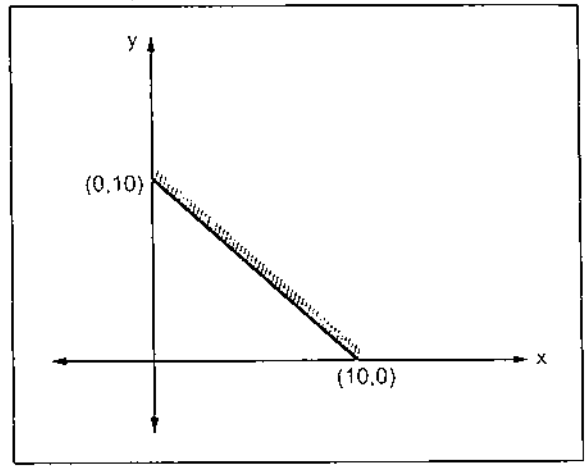


Fig. 6.9

$$\therefore \quad \text{The normal vector} = \frac{1}{10} I + \frac{1}{10} J$$

$$= \left[\frac{1}{10}, \frac{1}{10}, 0\right]$$

Normalizing,

$$\frac{[1/10, 1/10, 0]}{\sqrt{\frac{1}{100} + \frac{1}{100}}}$$

$$= \frac{[1/10, 1/10, 0]}{\frac{1}{5\sqrt{2}}}$$

$$= \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right]$$

$$\therefore \quad n_1 = \frac{1}{\sqrt{2}} \quad n_2 = \frac{1}{\sqrt{2}} \text{ and } n_3 = 0$$

∴ Translating the given plane using matrix

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

$$\therefore \quad T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & +10 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad |N| = \sqrt{1} = 1 \quad \lambda = \sqrt{1/2} = 1/\sqrt{2}$$

$$R_{xy} = \begin{bmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{xy}^{-1} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

∴ Resultant transformation matrix is given by

$$R_1 = T \cdot R_{xy} \cdot M \cdot R_{xy}^{-1} \cdot T^{-1}$$

$$R_1 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

∴ Final co-ordinate of the triangle can be given as

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot R_T$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 5 & 50 & 0 & 0 \\ 20 & 40 & 0 & 0 \\ 10 & 70 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -50 & -5 & 0 & 0 \\ -40 & -20 & 0 & 0 \\ -70 & -10 & 0 & 0 \end{bmatrix}$$

$\therefore$ A′ (– 50, – 5), B′ (– 40, – 20) and C′ (– 70, – 10) are the co-ordinates of the reflected triangle.

Ex. 6.7 : *Describe the transformation M_L which reflects an object about a line L.*

Sol. : Let line L have a y intercept (0, b) and an angle of inclination θ degrees (with respect to x axis).

We follow the steps given below to achieve the transformation

1. Translate (0, b) to origin
2. Rotate by – θ degrees so that line L algins with the x axis.
3. Mirror reflect about the x axis.
4. Rotate back by θ degrees
5. Translate the origin back to the point (0, b).

By transformation notation we have,

$M_L = T \cdot R_z \cdot M_x \cdot R_z^{-1} \cdot T^{-1}$

Review Questions

1. Give the 3-D transformation matrix for
   a) Translation
   b) Scaling and
   c) Rotation
2. Derive the transformation matrix for rotation about an arbitrary axis.
3. Derive the transformation matrix for rotation about an arbitrary plane.

University Questions

1. Write a program to rotate any given solid object by an angle θ about any arbitrary axis given by the line-

   y = mx + c (May-97)
2. Write are the sequence of transformations required to rotate an object about an axis P_0P_1 where $P_0 \equiv (x_0, y_0, z_0)$ and $P_1 \equiv (x_1, y_1, z_1)$ with angle θ. (May-98)
3. Develop a 3D transformation matrix for translation and rotation. (May-2000)
4. Given a unit cube with one corner at (0, 0, 0) and the opposite corner at (1, 1, 1), derive the transformations necesary to rotate the cube by 'θ' degrees about the main diagonal (from (0, 0, 0) to (1, 1, 1)) in the counter clockwise direction when looking along the diagonal towards the origin. (Dec-2000)

5. Give the sequence of transformations required to reflect the object about the line $y = 2x - 10$. (May-2001)

6. What are the sequence of transformations required to rotate an object about an axis P_0P_1 with angle ϕ in clockwise direction ? (Dec-2001)

7. What are the properties of concatenation of transformations ? Derive a 2-dimensional transformation matrix for rotating a point $P(x_1, y_2)$ about a line $y = mx + c$. (May-2002)

8. Illustrate the rotation of a 3-D object about an arbitrary axis with derivation. (May-2002)

9. Give the sequence of transformations required to reflect an object about the line $y = mx + c$. Derive composite transformation matrix for the same. (May-2003)

❑❑❑

7 Three Dimensional Viewing, Projection and Clipping

7.1 Introduction

In chapter 6, we have seen that two dimensional viewing operations transfer positions from the world coordinate plane to pixel positions in the plane of display device. In three dimensional viewing the situation is bit more complex, since we now have more options as to how views are to be generated. First of all, we can view an object from any spatial position : from front, from back or from above. Further more we can generate a view of what we would see if we were standing in the middle of a group of objects or inside a single object. Another important aspect must be considered in the three dimensional viewing is that even though the object is three dimensional it must be projected onto the flat viewing surface of the display device.

In this chapter, we discuss the general operations required to generate three dimensional viewing. It includes the study of parallel and perspective projections, viewing parameters and three dimensional clipping.

7.2 Three Dimensional Viewing

As mentioned earlier, the 3D viewing process is inherently more complex than the 2D viewing process. In two dimensional viewing we have 2D window and 2D viewport and objects in the world coordinates are clipped against the window and are then transformed into the viewport for display. The complexity in added in the three dimensional viewing is because of the added dimension and the fact that eventhough objects are three dimensional the display devices are only 2D.

The mismatch between 3D objects and 2D displays is compensated by introducing projections. The projections transform 3D objects into a 2D projection plane. The Fig. 7.1 shows the conceptual model of the 3D transformation process.

In 3D viewing, we specify a view volume in the world coordinates using modelling transformation. The world coordinate positions of the objects are then converted into viewing coordinates by viewing transformation. The projection transformation is then used to convert 3D description of objects in viewing coordinates to the 2D projection coordinates. Finally, the workstation transformation transforms the projection coordinates into the device coordinates.

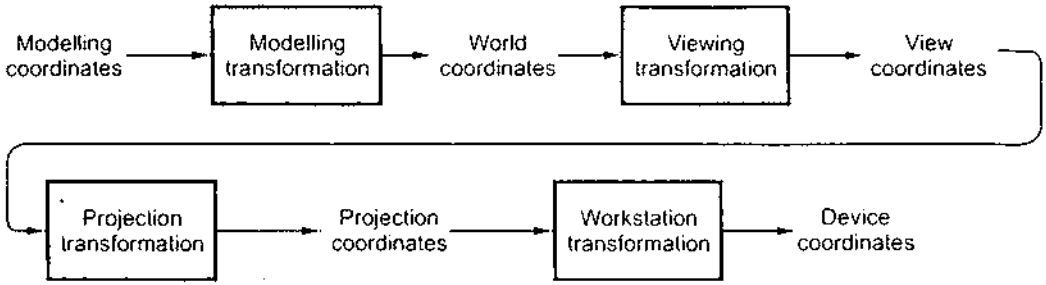


Fig. 7.1 Conceptual model of 3D transformation process

7.3 Viewing Parameters

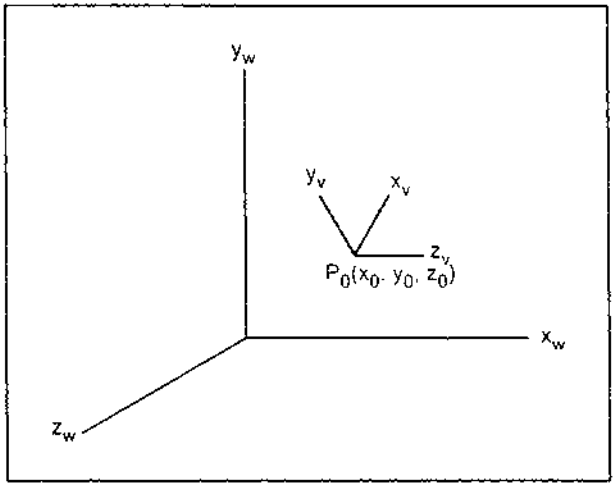


Fig. 7.2 Right handed viewing coordinate system

As mentioned earlier, we can view the object from the side, or the top, or even from behind. Therefore, it is necessary to choose a particular view for a picture by first defining a view plane. A view plane is nothing but the film plane in a camera which is positioned and oriented for a particular shot of the scene. World coordinate positions in the scene are transformed to viewing coordinates, then viewing coordinates are projected onto the view plane. A view plane can be defined by establishing the viewing - coordinate system or view reference coordinate system, as shown in the Fig. 7.2

The first viewing parameter we must consider is the **view reference point**. This point is the center of our viewing coordinate system. It is often chosen to be close to or on the surface of some object in a scene. Its coordinates are specified as X_R, Y_R and Z_R.

The next viewing parameter is a **view-plane normal vector**, N. This normal vector is the direction perpendicular to the view plane and it is defined as [DXN, DYN, DZN]. We know that the view plane is the film in the camera and we focus camera towards the view reference point. This means that the camera is pointed in the direction of the view plane normal. This is illustrated in Fig. 7.3. (See on next page)

As shown in the Fig. 7.3, the view plane normal vector is a directed line segment from the view plane to the view reference point. The length of this directed line segment is referred to as **view - distance**. This is another viewing parameter. It tells how far the camera is positioned from the view reference point. In other words we can say that a view plane is positioned view - distance away from the view reference point in the direction of the view plane normal. This is illustrated in Fig. 7.4.

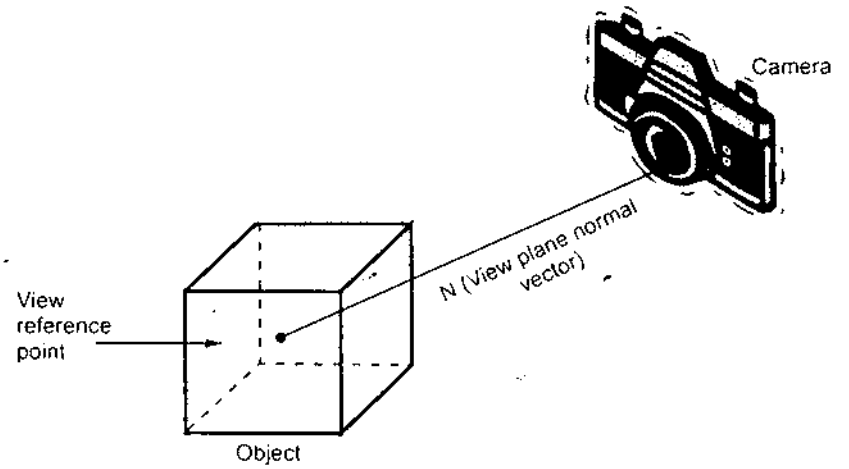


Fig. 7.3 View reference point and view plane normal vector

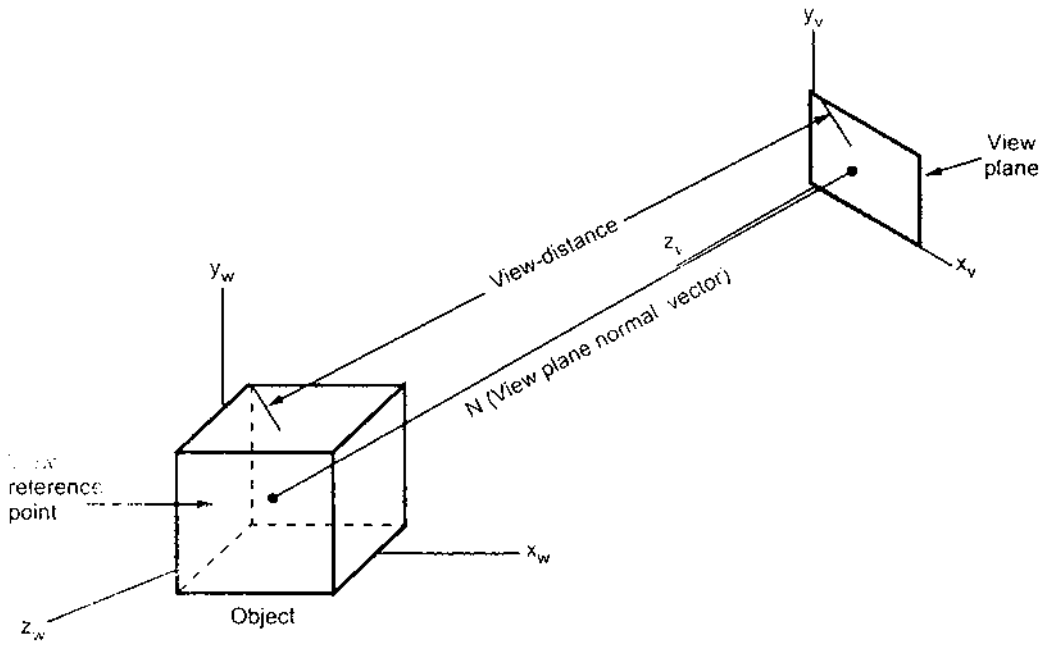


Fig. 7.4 3-D viewing parameters

As shown in the Fig. 7.4 we have world coordinate system which we used to model our object, and we have view plane coordinates, which are attached to the view plane.

It is possible to obtain the different views by rotating the camera about the view plane normal vector and keeping view reference point and direction of N vector fixed, as shown in the Fig. 7.5

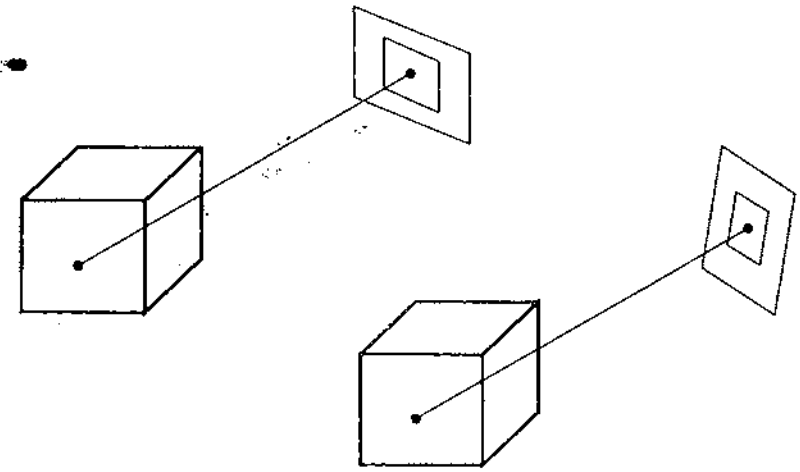


Fig. 7.5 Rotating view plane

At different angles, the view plane will show the same scene, but rotated so that a different part of the object is up. The rotation of a camera or view plane is specified by a view-up vector V [XUP YUP ZUP] which is another important viewing parameter.

We can also obtain a series of views of a scene, by keeping the view reference point fixed and changing the direction of N, as shown in the Fig. 7.6 changing the view plane normal changes the orientation of camera or view plane giving different views.

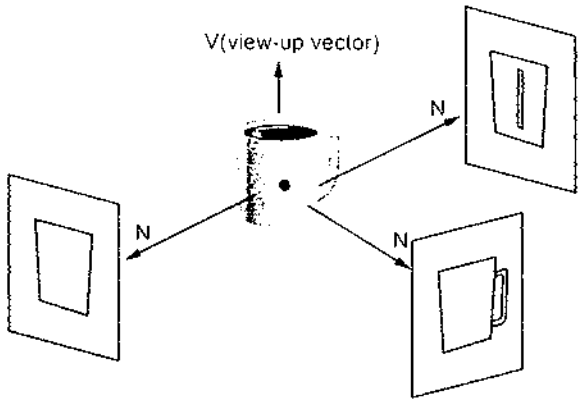


Fig. 7.6 Viewing object by changing view plane vector N

In this section we have seen viewing parameters such as view reference point, view plane normal vector, view-distance and view-up vector. These parameters allow the user to select the desired view of the object.

7.4 Transformation from World Coordinate to Viewing Coordinates

The conversion of object description from world coordinates to viewing coordinates is achieved by following transformation sequence.

1. Translate the view reference point to the origin of the world coordinate system.
2. Apply rotations to align the x_v, y_v and z_v axes with the world coordinate x_w, y_w and z_w axes, respectively.

The view point specified at world position (x_p, y_p, z_p) can be translated to the world coordinate origin with the matrix transformation

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_p & -y_p & -z_p & 1 \end{bmatrix}$$

$$\therefore \quad P' = P \cdot T$$

For alignment of three axes we require the three coordinate-axis rotations, depending on the direction we choose for N. In general, if N is not aligned with any world coordinate axis, we can align the viewing and world coordinate systems with the transformation sequence $R_x \cdot R_y \cdot R_z$. That is, we first rotate around the world x_w axis to bring z_v into the $x_w z_w$ plane. Then, we rotate around the world y_w axis to align the z_w and z_v axes. Finally, we rotate about the z_w axis to align the y_w and y_v axes. In case of left handed view reference system, a reflection of one of the viewing axes is also necessary. This is illustrated in Fig. 7.7.

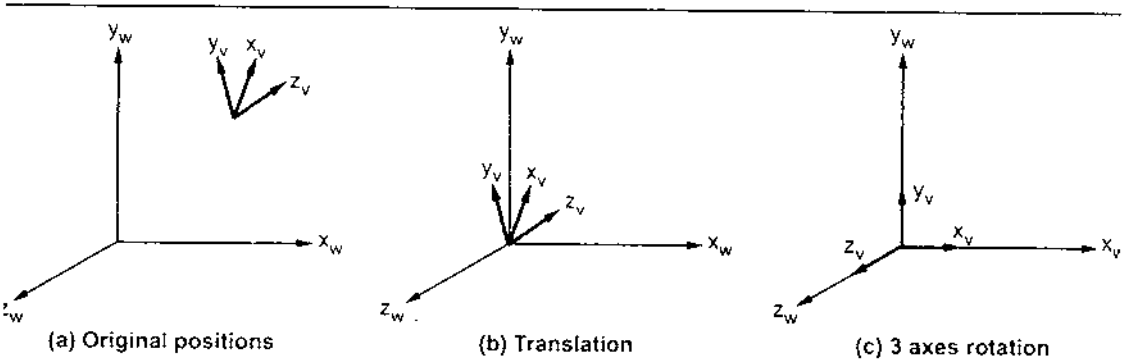


Fig. 7.7 Aligning of viewing and world coordinate axes using a sequence of translate - rotate transformations

Therefore, the composite transformation matrix is given as

$$T_C = T \cdot R_x \cdot R_y \cdot R_z$$

There is another way to generate composite rotation matrix. A composite rotation matrix can be directly generated by calculating unit u, v, n vectors. If we know N and V vectors, the unit vectors are calculated as

$$n = \frac{N}{|N|} = (n_1, n_2, n_3)$$

$$u = \frac{v \times n}{|v \times N|} = (u_1, u_2, u_3) \qquad v = n \times u = (v_1, v_2, v_3)$$

This method of generating composite rotation matrix automatically adjusts the direction of V so that v is perpendicular to n. The composite rotation matrix for the viewing transformation is given as

$$R = \begin{bmatrix} u_1 & v_1 & n_1 & 0 \\ u_2 & v_2 & n_2 & 0 \\ u_3 & v_3 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transforms u onto the world x_w axis, v onto the y_w axis, and n onto the z_w axis. Furthermore, this matrix automatically performs the reflection necessary to transform a left-handed viewing system onto the right handed world system.

With second method, the composite transformation matrix is given as

$$T_c = T \cdot R$$

7.5 Projections

After converting the description of objects from world coordinates to viewing coordinates, we can project the three dimensional objects onto the two dimensional view plane. There are two basic ways of projecting objects onto the view plane : Parallel projection and Perspective projection.

7.5.1 Parallel Projection

In parallel projection, z - coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which correspond to connections on the original object.

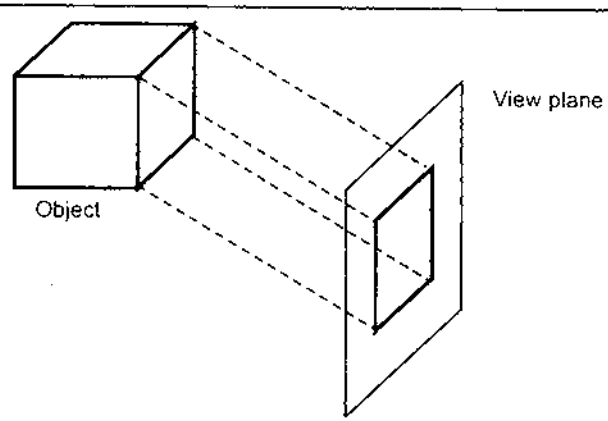


Fig. 7.8 Parallel projection of an object to the view plane

As shown in the Fig. 7.8, a parallel projection preserves relative proportions of objects but does not produce the realistic views.

7.5.2 Perspective Projection

The perspective projection, on the other hand, produces realistic views but does not preserve relative proportions. In perspective projection, the lines of projection are not parallel. Instead, they all converge at a single point called the **center of projection** or **projection reference point**. The object positions are transformed to the view plane along these converged projection lines and the projected view of an object is determined by calculating the intersection of the converged projection lines with the view plane, as shown in the Fig. 7.9.

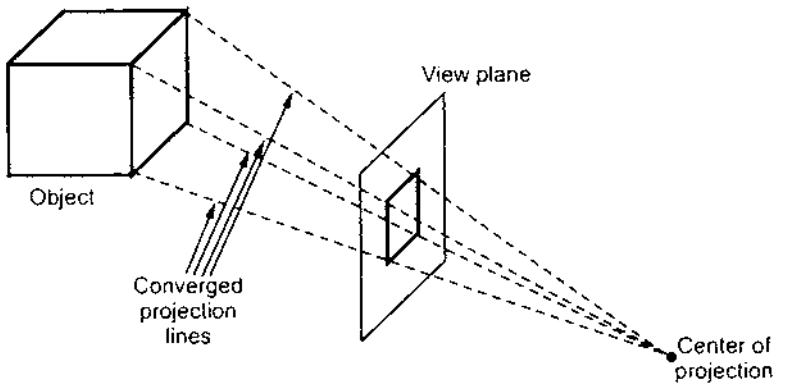
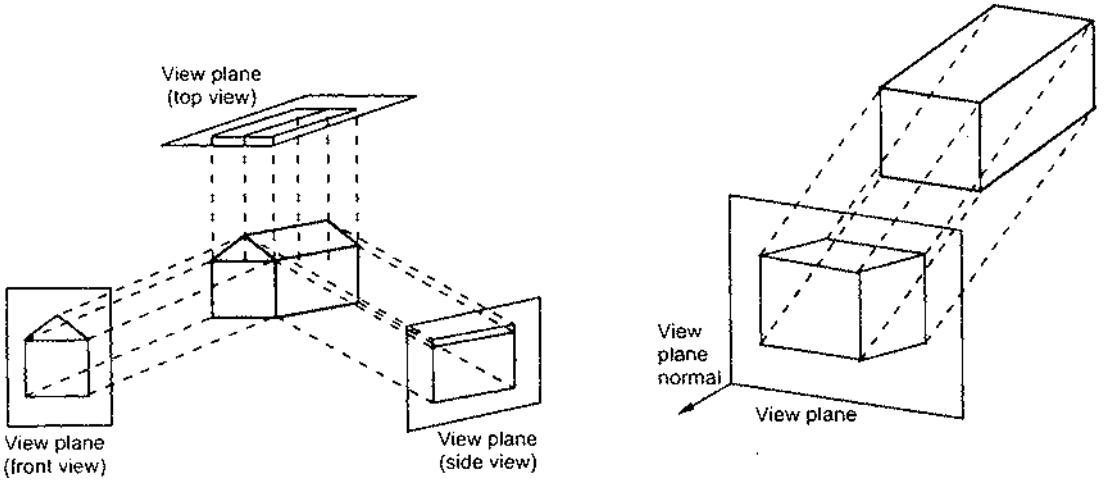


Fig. 7.9 Perspective projection of an object to the view plane

7.5.3 Types of Parallel Projections

Parallel projections are basically categorized into two types, depending on the relation between the direction of projection and the normal to the view plane. When the direction of the projection is normal (perpendicular) to the view plane, we have an orthographic parallel projection. Otherwise, we have an oblique parallel projection. Fig. 7.10 illustrates the two types of parallel projection.



(a) Orthographic parallel projection **(b) Oblique parallel projection**

Fig. 7.10

7.5.3.1 Orthographic Projection

As shown in Fig. 7.10 (a), the most common types of orthographic projections are the front projection, top projection and side projection. In all these, the projection plane (view plane) is perpendicular to the principle axis. These projections are often used in engineering drawing to depict machine parts, assemblies, buildings and so on.

The orthographic projection can display more than one face of an object. Such an orthographic projection is called **axonometric** orthographic projection. It uses projection planes (view planes) that are not normal to a principle axis. They resemble the perspective projection in this way, but differ in that the foreshortening is uniform rather than being related to the distance from the center of projection. Parallelism of lines is preserved but angles are not. The most commonly used axonometric orthographic projection is the isometric projection.

The isometric projection can be generated by aligning the view plane so that it intersects each coordinate axis in which the object is defined at the same distance from the origin. As shown in the Fig. 7.11, the isometric projection is obtained by aligning the projection vector with the cube diagonal. It uses an useful property that all three principle axes are equally foreshortened, allowing measurements along the axes to be made to the same scale (hence the name : iso for equal, metric for measure).

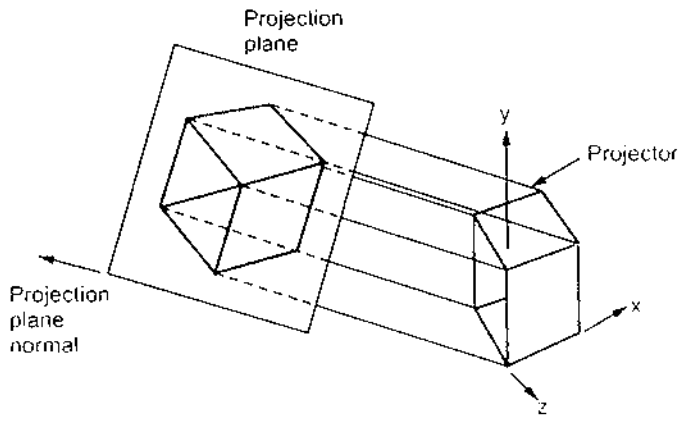


Fig. 7.11 Isometric projection of an object onto a viewing plane

7.5.3.2 Oblique Projection

An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane. Fig. 7.10 (b) shows the oblique projection. Notice that the view plane normal and the direction of projection are not the same. The oblique projections are further classified as the **cavalier** and **cabinet projections**. For the cavalier projection, the direction of projection makes a 45° angle with the view plane. As a result, the projection of a line perpendicular to the view plane has the same length as the line itself; that is, there is no foreshortening. Fig. 7.12 shows cavalier projections of a unit cube with $\alpha = 45°$ and $\alpha = 30°$.

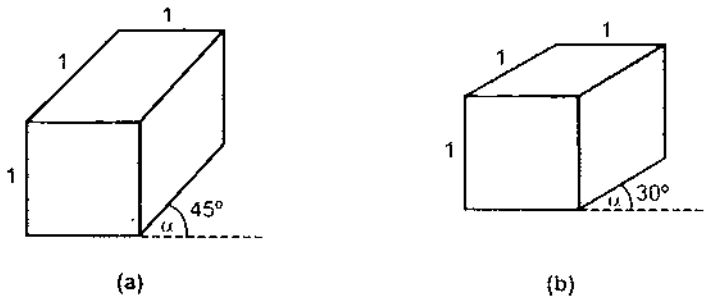


Fig. 7.12 Cavalier projections of the unit cube

When the direction of projection makes an angle of arctan (2) = 63.4° with the view plane, the resulting view is called a **cabinet projection**. For this angle, lines perpendicular to the viewing surface are projected at one-half their actual length. Cabinet projections appear more realistic than cavalier projections because of this reduction in the length of perpendiculars. Fig. 7.13 shows the examples of cabinet projections for a unit cube.

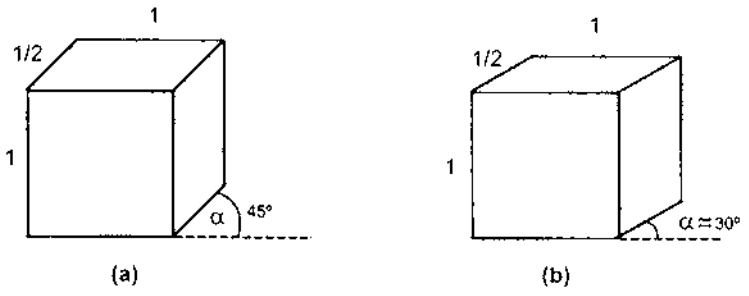


Fig. 7.13 Cabinet projections of the unit cube

7.5.4 Types of Perspective Projections

The perspective projection of any set of parallel lines that are not parallel to the projection plane converge to a **vanishing point**. The vanishing point for any set of lines that are parallel to one of the three principle axes of an object is referred to as a **principle vanishing point** or **axis vanishing point**. There are at most three such points, corresponding to the number of principle axes cut by the projection plane. The perspective projection is classified according to number of principle vanishing points in a projection : one-point, two points or three-point projections. Fig. 7.14 shows the appearance of one-point and two-point perspective projections for a cube.

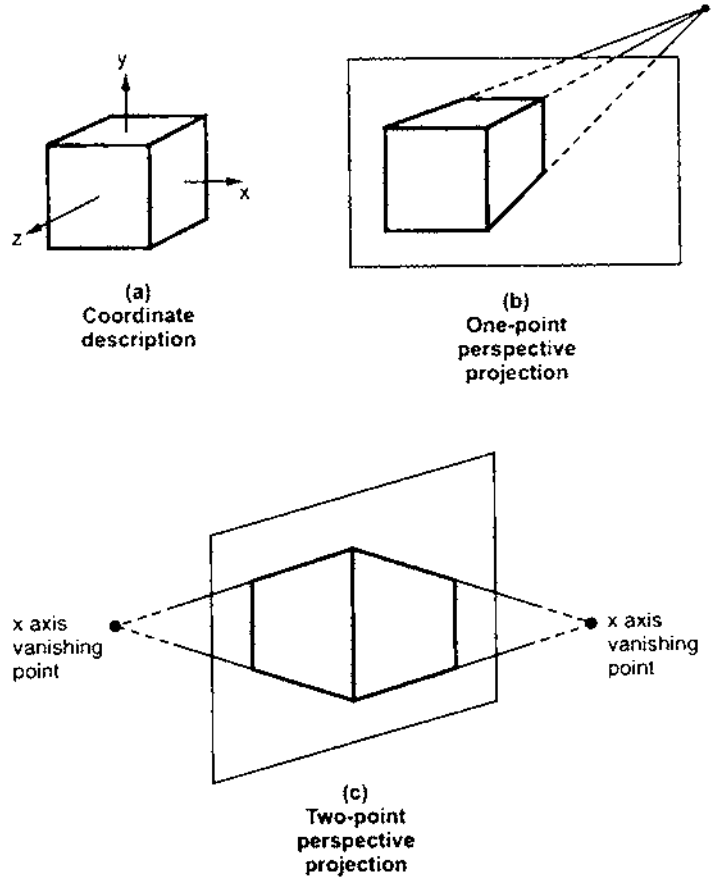


Fig. 7.14 Perspective projections

The Fig. 7.15 summarizes the logical relationship among the various types of projections

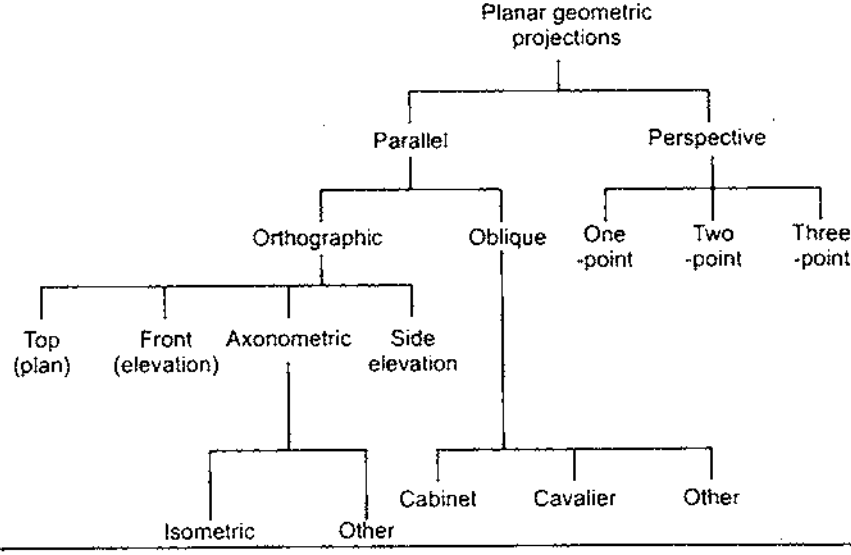


Fig. 7.15

7.5.5 Transformation Matrices for General Parallel Projection

7.5.5.1 On XY Plane

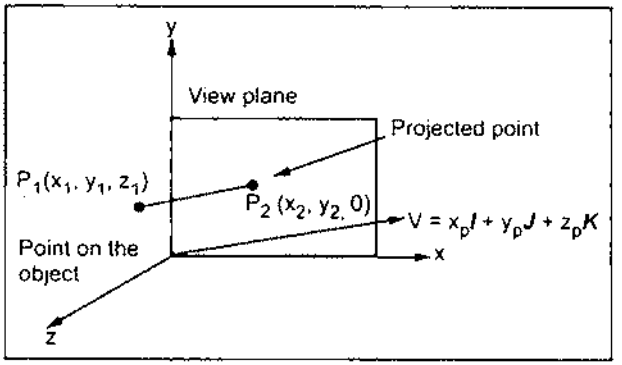


Fig. 7.16

In a general parallel projection, we may select any direction for the lines of projection. Suppose that the direction of projection is given by the vector $[x_p\ y_p\ z_p]$ and that the object is to be projected onto the xy plane. If the point on the object is given as (x_1, y_1, z_1), then we can determine the projected point (x_2, y_2) as given below :

The equations in the parametric form for a line passing through the projected point (x_2, y_2, z_2) and in the direction of projection are given as

$$x_2 = x_1 + x_p u$$
$$y_2 = y_1 + y_p u$$
$$z_2 = z_1 + z_p u$$

For projected point z_2 is 0, therefore, the third equation can be written as,

$$0 = z_1 + z_p u$$
$$\therefore \quad u = \frac{-z_1}{z_p}$$

Substituting the value of u in first two equations we get,

$$x_2 = x_1 + x_p(-z_1/z_p) \quad \text{and}$$
$$y_2 = y_1 + y_p(-z_1/z_p)$$

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

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

or in homogeneous coordinates we have,

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

i.e. $\quad P_2 = P_1 \cdot Par_v$

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

Here, we ignore the value of z_2 when drawing the projected image.

7.5.5.2 On Given View Plane

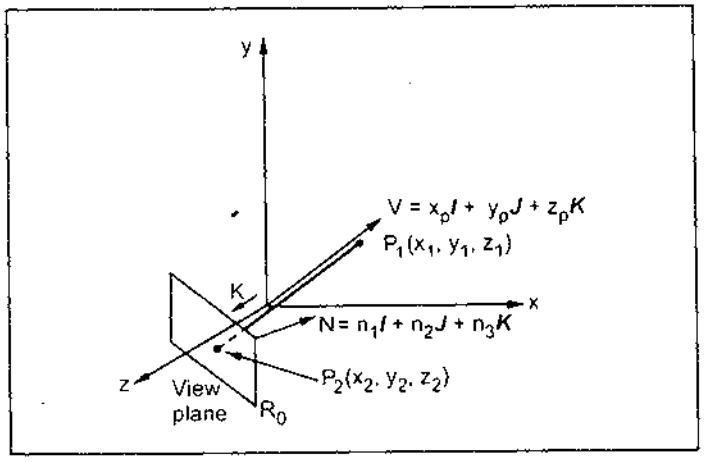


Fig. 7.17

From the above transformation matrix we can derive the general equation of parallel projection onto a given view plane instead of xy plane in the direction of a given vector $V(x_p, y_p, z_p)$ as follows :

Let us consider that R_0 is the view reference point, P_1 is the object point and P_2 is the projected point. Now perform the following steps :

1. Translate the view reference point R_0 of the view plane to the origin using the translation matrix T.
2. Perform an alignment transformation R_{xy} so that the view normal vector N of the view plane points in the direction K, the normal to the xy plane.
3. Project point P_1 on to the xy plane.
4. Perform the inverse of steps 2 and 1.

$$\therefore \quad Par_{v,N,R_0} = T \cdot R_{xy} \cdot Par_v \cdot R_{xy}^{-1} \cdot T^{-1}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_0 & -y_0 & -z_0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda}{|N|} & 0 & \frac{n_1}{|N|} & 0 \\ \frac{-n_1 n_2}{\lambda |N|} & \frac{n_3}{\lambda} & \frac{n_2}{|N|} & 0 \\ \frac{-n_1 n_3}{\lambda |N|} & \frac{-n_2}{\lambda} & \frac{n_3}{|N|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-x_p}{z_p} & \frac{-y_p}{z_p} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\lambda}{|N|} & \frac{-n_1 n_2}{\lambda |N|} & \frac{-n_1 n_3}{\lambda |N|} & 0 \\ 0 & \frac{n_3}{\lambda} & \frac{-n_2}{\lambda} & 0 \\ \frac{n_1}{|N|} & \frac{n_2}{|N|} & \frac{n_3}{|N|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_0 & y_0 & z_0 & 1 \end{bmatrix}$$

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

7.5.6 Transformation Matrix for Oblique Projection onto xy Plane

Oblique projections to the xy plane can be specified by a number f and an angle θ. Here f prescribes the ratio that any line L perpendicular to the xy plane will be foreshortened after the projection. The angle θ is the angle that the projection of any line perpendicular to the xy plane makes with positive x axis.

To find the projection transformation, we need to determine the direction vector V. From Fig. 7.18, with line L of length l, we choose vector V to have the direction same as that of vector $\overline{P_1P_2}$.

$$\therefore \quad V = \overline{P_1P_2} = x_2 I + y_2 J - l K$$

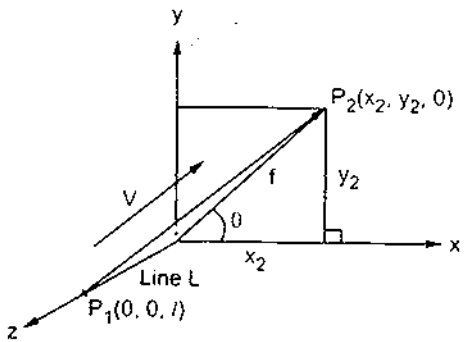


Fig. 7.18

Comparing with

$$V = x_p I + y_p J + z_p K$$

we get,

$$x_p = x_2 = f \cos\theta$$
$$y_p = y_2 = f \sin\theta$$
$$z_p = -l$$

∴Using result of previous article, we have

$$Par_v = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{f\cos\theta}{l} & \frac{f\sin\theta}{l} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is the general form of an oblique projection onto the xy plane.

7.5.7 Transformation Matrix for Perspective Projection

Let us consider the center of projection is at $P_c(x_c, y_c, z_c)$ and the point on object is $P_1(x_1, y_1, z_1)$, then the parametric equation for line containing these points can be given as

$$x_2 = x_c + (x_1 - x_c)\,u$$

$$y_2 = y_c + (y_1 - y_c)\,u$$

$$z_2 = z_c + (z_1 - z_c)\,u$$

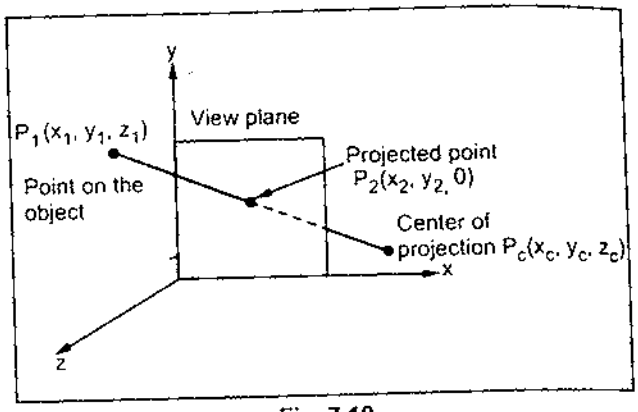


Fig. 7.19

For projected point z_2 is 0, therefore, the third equation can be written as

$$0 = z_c + (z_1 - z_c)\,u$$

$$\therefore\ u = -\frac{z_c}{z_1 - z_c}$$

Substituting the value of u in first two equations we get,

$$x_2 = x_c - z_c\,\frac{x_1 - x_c}{z_1 - z_c}$$

$$= \frac{x_c z_1 - x_c z_c - x_1 z_c + x_c z_c}{z_1 - z_c}$$

$$= \frac{x_c z_1 - x_1 z_c}{z_1 - z_c} \quad \text{and}$$

$$y_2 = y_c - z_c\,\frac{y_1 - y_c}{z_1 - z_c}$$

$$= \frac{y_c z_1 - y_c z_c - y_1 z_c + y_c z_c}{z_1 - z_c}$$

$$= \frac{y_c z_1 - y_1 z_c}{z_1 - z_c}$$

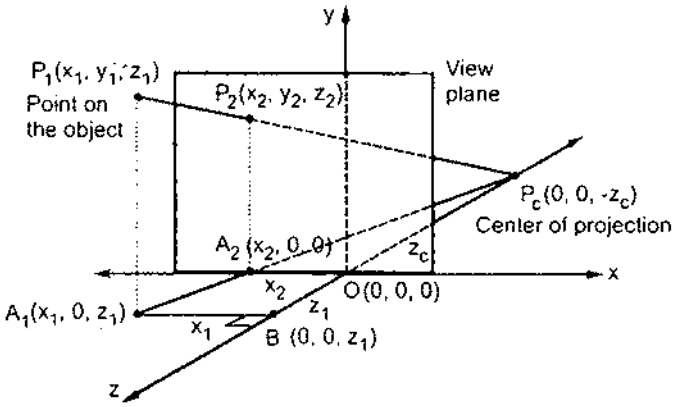
The above equations can be represented in the homogeneous matrix form as given below :

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1]\begin{bmatrix} -z_c & 0 & 0 & 0 \\ 0 & -z_c & 0 & 0 \\ x_c & y_c & 0 & 1 \\ 0 & 0 & 0 & -z_c \end{bmatrix}$$

Here, we have taken the center of projection as $P_c(x_c, y_c, z_c)$. If we take the center of projection on the negative z-axis such that

$$x = 0$$
$$y = 0$$
$$z = -z_c$$

i.e. $P_c(0, 0, -z_c)$ then we have



$\Delta P_c O A_2 \sim \Delta P_c B A_1$

$\therefore x_2 = \dfrac{x_1 z_c}{z_1 + z_c}$

Similarly, $y_2 = \dfrac{y_1 z_c}{z_1 + z_c}$

Fig. 7.20

$$x_2 = \frac{z_c x_1}{z_c + z_1}$$

$$y_2 = \frac{z_c y_1}{z_c + z_1}$$

$$z_2 = 0$$

Thus we get the homogeneous perspective transformation matrix as

$$[x_2 \quad y_2 \quad z_2 \quad 1] = [x_1 \quad y_1 \quad z_1 \quad 1] \begin{bmatrix} z_c & 0 & 0 & 0 \\ 0 & z_c & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & z_c \end{bmatrix}$$

7.6 Three Dimensional Clipping

In chapter 6, we have seen the concept of window, which served as clipping boundary in two-dimensional space. In three dimensional space the concept can be extended to a **clipping volume** or **view volume**. The two common three dimensional clipping volumes are a rectangular parallelepiped, i.e. a box, used for parallel or axonometric projections, and a truncated pyramidal volume, used for perspective projections. Fig. 7.21 shows these volumes. These volumes are six sided with sides : left, right, top, bottom, hither (near), and yon (far).

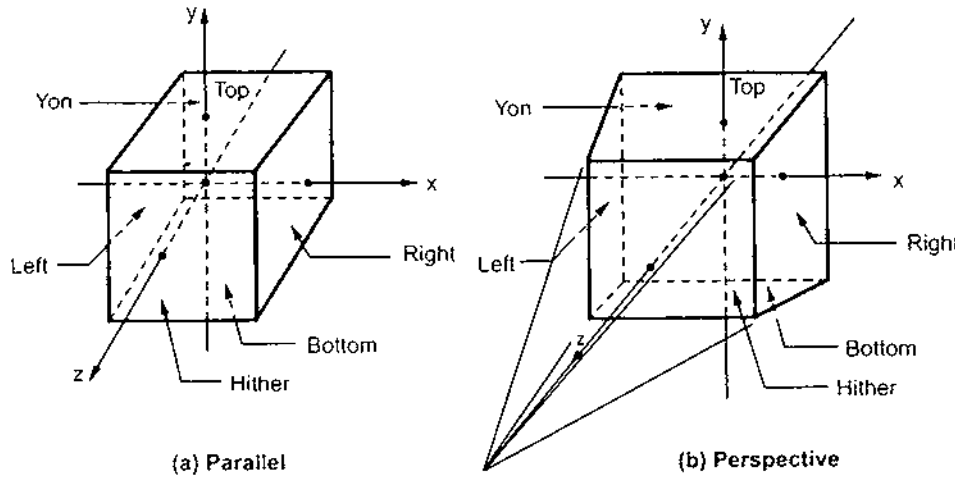


Fig. 7.21

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

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

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

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

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

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

Bit 6 = 1, if the end point is behind the volume

Otherwise, the bit is set to zero. As an example, a region code of 101000 identifies a point as above and behind the view volume, and the region code 000000 indicates a point within the view volume.

A line segment can be immediately identified as completely within the view volume if both endpoints have a region code of 000000. If either endpoint of a line segment does not have a region code of 000000, we perform the logical AND operation on the two endpoint codes. If the result of this AND operation is nonzero then both endpoints are outside the view volume and line segment is completely invisible. On the other hand, if the result of AND operation is zero then line segment may be partially visible. In this case, it is necessary to determine the intersection of the line and the clipping volume.

We have seen that determining the end point codes for a rectangular parallelepiped clipping volume is a straight forward extension of the two dimensional algorithm. However, the perspective clipping volume shown in Fig. 7.21(b) requires some additional processing. As shown in the Fig. 7.21(b), the line connecting the center of projection and the center of the perspective clipping volume coincides with the z axis in a right handed coordinate system.

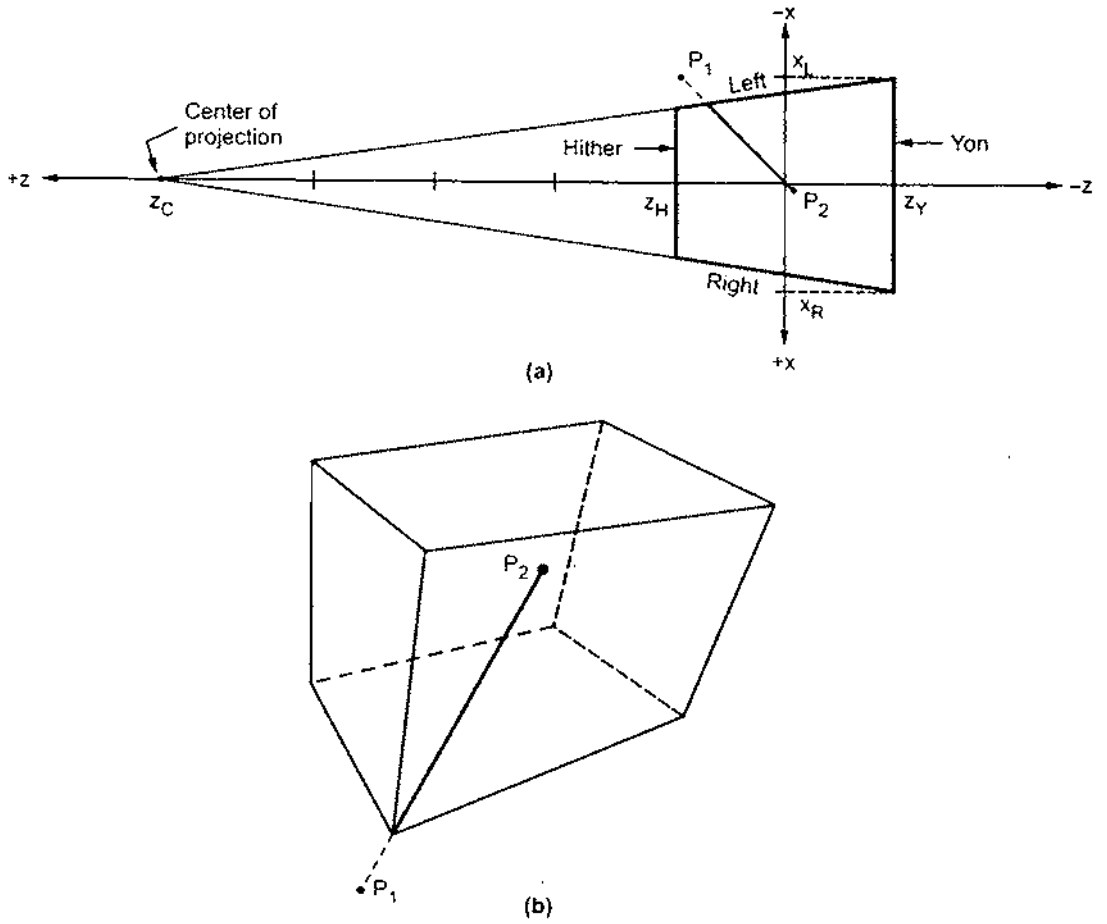


Fig. 7.22

Fig. 7.22 shows a top view of the perspective clipping volume. The equation of the line which represents the right hand plane in this view can be given as

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

where

$$\alpha_1 = \frac{x_R}{z_Y - z_C} \quad \text{and } \alpha_2 = -\alpha_1 z_C$$

This equation of right hand plane can be used to determine whether a point is to the right, on or to the left of the plane, i.e., outside the volume, on the right hand plane, or inside the volume. Substituting the x and y coordinates of a point P into $x - z\alpha_1 - \alpha_2$ gives the following results

$f_R = x - z\alpha_1 - \alpha_2$	> 0	if P is to the right of the right plane
	$= 0$	if P is on the right plane
	< 0	if P is to the left of the right plane

Similarly, we can derive the test functions for the left, top bottom, hither and yon planes. Table 7.1 shows these test functions.

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

Table 7.1 Test functions for six planes of clipping volume

7.7 Three - Dimensional Midpoint Subdivision Algorithm

In the previous section, we have seen how to identify location of the end points of line segments with respect to clipping volume. Once this process is over we can determine which line segments are completely visible, which are completely invisible and which are partially visible. For partially visible segment we have to determine the intersection with clipping volume. This can be achieved with the help of three - dimensional midpoint subdivision algorithm. It is an extension of 2D midpoint subdivision algorithm discussed in section 6.3.2.2.

Algorithm :

1. Find the locations of endpoints (endpoint codes) of line segments with respect to clipping volume (using test functions in case of perspective clipping volume)
2. Check visibility of each line segment
   a) If codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 4.
   b) If codes for endpoints are not zero and the logical ANDing of them is also nonzero the line is completely invisible, so reject the line and go to step 4.
   c) If codes for two endpoints do not satisfy the conditions in 2 a) and 2 b) the line is partially visible.
3. Divide the partially visible line segments in equal parts and repeat steps 1 and 2 for subdivided line segments until you get completely visible and completely invisible line segments. Draw the visible line segment and discard the invisible one.
4. Stop.

Solved Examples

Ex. 7.1 : *Under the standard perspective, what is the projected image of*

a) a point in the plane $z = -z_c$

b) the line segment joining P_1 $(1, -1, -3z_c)$ *to* P_2 $(3, 1, 0)$

Sol. :

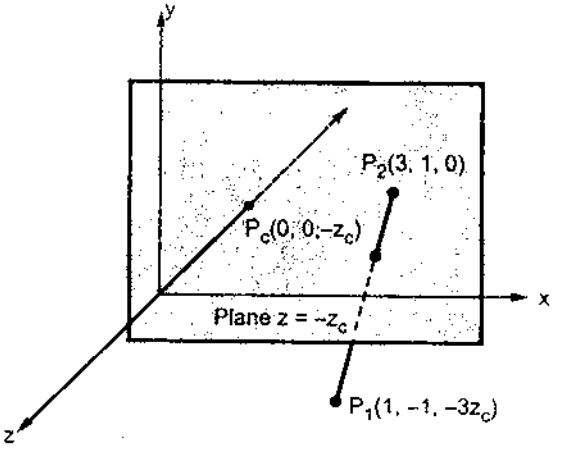


Fig. 7.23

a) The plane $z = -z_c$ is the plane parallel to xy view plane and is at a distance of z_c units from it. The centre of projection $P_c(0, 0, -z_c)$ lies on the plane. If $P(x, y, -z_c)$ is any point in this plane, the line of projection P_cP does not intersect the xy view plane. Thus $P(x, y, -z_c)$ is said to be projected out to infinity (∞).

b) The given line P_1P_2 passes through the plane $z = -z_c$. The equation of the line is given by

$$x = 1 + 2t \quad y = -1 + 2t \quad z = -3z_c + 3z_ct$$

Applying the standard projection to the equation of the line, we get

$$[1+2t \quad -1+2t \quad -3z_c+3z_ct \quad 1]\begin{bmatrix} z_c & 0 & 0 & 0 \\ 0 & z_c & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & z_c \end{bmatrix} = [z_c+2z_ct \quad -z_c+2z_ct \quad 0 \quad -2z_c+3z_ct]$$

Changing to 3D co-ordinates, the equations of the projected line segment are

$$x = \frac{z_c + 2z_ct}{-2z_c + 3z_ct} = \frac{1+2t}{-2+3t}$$

$$y = \frac{-z_c + 2z_ct}{-2z_c + 3z_ct} = \frac{-1+2t}{-2+3t}$$

$$z = 0$$

Ex. 7.2 : *Using the origin as the centre of projection, derive the perspective transformation onto the plane passing through the point P_o (x_o, y_o, z_o) and having the normal vector $N = n_1I + n_2J + n_3K$.*

Sol. : Let P_1 (x_1, y_1, z_1) be projected onto P_2 (x_2, y_2, z_2). From Fig. 7.24 we see that the line segments P_1O and P_2O are along the same line. Hence there exists a constant 'u' such that $P_2O = uP_1O$. Comparing the corresponding components, we get

$$\left.\begin{aligned} x_2 &= ux_1 \\ y_2 &= uy_1 \\ z_2 &= uz_1 \end{aligned}\right\} \quad \ldots (1)$$

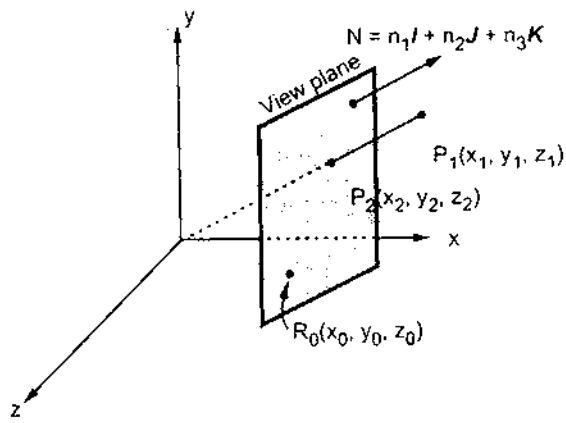


Fig. 7.24

We now find the value of u. As any point $P_2(x_2, y_2, z_2)$ lying on the plane satisfies the equation

$$n_1x_2 + n_2y_2 + n_3z_2 = d_0$$

where $d_0 = n_1x_0 + n_2y_0 + n_3 z_0$.

Putting values of x_2, y_2 and z_2 into the above equation, we have

$$n_1(u\, x_1) + n_2 (uy_1) + n_3 (uz_1) = d_0$$

$$\therefore \quad u = \frac{d_0}{n_1x_1 + n_2 y_1 + n_3 z_1}$$

This projection transformation can be represented using homogeneous coordinate representation for 3D points as follows

$$Per_{N, R_0} = \begin{bmatrix} d_0 & 0 & 0 & n_1 \\ 0 & d_0 & 0 & n_2 \\ 0 & 0 & d_0 & n_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Application of this matrix to homogeneous representation $P_1(x_1, y_1, z_1, 1)$ of point P_1 gives $P_2(d_0 x_1, d_0 y_1, d_0 z_1, n_1x_1 + n_2 y_1 + n_3 z_1)$, which is homogeneous representation of $P_2(x_2, y_2, z_2)$ found above i.e. in matrix form,

$$[x_1 \quad y_1 \quad z_1 \quad 1] \begin{bmatrix} d_0 & 0 & 0 & n_1 \\ 0 & d_0 & 0 & n_2 \\ 0 & 0 & d_0 & n_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} = [d_0 x_1 \quad d_0 y_1 \quad d_0 z_1 \quad n_1 x_1 + n_2 y_1 + n_3 z_1]$$

Ex. 7.3 : *Find the perspective projection onto the view plane $z = z_c$ where the centre of projection is the origin 0(0, 0, 0).*

Sol : The plane $z = z_c$ is parallel to xy plane and is z_c units away from it. Thus view plane normal N is same as normal vector k to the xy plane. i.e. N = K. Choosing the view reference point as $R_0(0, 0, z_c)$, then from the problem 7.2 we get the parameters as

$$N(n_1, n_2, n_3) = (0, 0, 1)$$

and $$R_0(x_0, y_0, z_0) = (0, 0, z_c)$$

So $$d_0 = n_1x_0 + n_2y_0 + n_3z_0 = z_c$$

∴The projection matrix is

$$Per_{K, R_0} = \begin{bmatrix} z_c & 0 & 0 & 0 \\ 0 & z_c & 0 & 0 \\ 0 & 0 & z_c & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Ex. 7.4 : *Derive the general perspective transformation onto a plane with reference point $R_0(x_0, y_0, z_0)$, normal vector $N = n_1I + n_2J + n_3K$, and using C (a, b, c) as the centre of projection.*

Sol.: From Fig. 7.25 we see that the line segments CP_2 and CP_1 are along the same line. Hence there exists a constant 'u' such that $CP_2 = uCP_1$. Comparing the corresponding components we have

$$x_2 = a + (x_1 - a)\,u$$
$$y_2 = b + (y_1 - b)\,u$$
$$z_2 = c + (z_1 - c)\,u$$

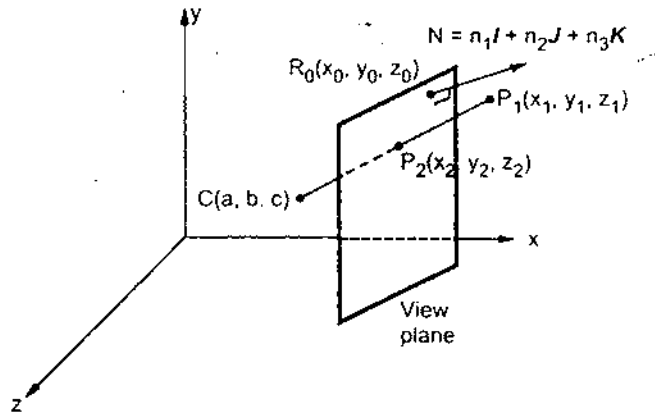


Fig. 7.25

Now $P_2\,(x_2, y_2, z_2)$ lies in the view plane,

$$\therefore \quad n_1 x_2 + n_2 y_2 + n_3 z_2 = d_0$$

where $\quad d_0 = n_1 x_0 + n_2 y_0 + n_3 z_0$

$$\therefore\ n_1\,[a + (x_1 - a)\,u] + n_2\,[b + (y_1 - b)\,u] + n_3\,[c + (z_1 - c)\,u] = d_0$$

$$\therefore \quad u = \frac{d}{n_1(x_1 - a) + n_2(y_1 - b) + n_3(z_1 - c)}$$

where $\quad d = d_0 - d_1$

i.e. $\quad d = d_0 - (n_1 a + n_2 b + n_3 c)$

To find the homogeneous coordinate matrix representation we proceed as follows

1. Translate the centre of projection C (a, b, c) so that it lies at the origin. Hence $R_0'(x_0 - a, y_0 - b, z_0 - c)$ becomes the view reference point of the translated plane. The normal vector remains the same.
2. Project onto the translated plane using origin as centre of projection (Refer problem 7.2)
3. Translate back to the original position.

$$\therefore \quad Per_{N,\,R_0,\,C} = T \cdot Per_{N,\,R_0'} \cdot T^{-1}$$

where R_0' is used as reference point in constructing projection $Per_{N,\,R_0'}$

$$\therefore \quad Per_{N, R_0, C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -a & -b & -c & 1 \end{bmatrix} \begin{bmatrix} d & 0 & 0 & n_1 \\ 0 & d & 0 & n_2 \\ 0 & 0 & d & n_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ a & b & c & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d + an_1 & bn_1 & cn_1 & n_1 \\ an_2 & d + bn_2 & cn_2 & n_2 \\ an_3 & bn_3 & d + cn_3 & n_3 \\ -ad_0 & -bd_0 & -cd_0 & -d_1 \end{bmatrix}$$

This is the general perspective transformation onto a plane with the reference point $R_0(x_0, y_0, z_0)$, normal vector N and centre of projection C(a, b, c).

Ex. 7.5 : *Find the transformation for*

a) Cavalier projection with $\theta = 45°$

b) Cabinet projection with $\theta = 30°$

Sol. : a) A cavalier projection is an oblique projection where there is **no foreshortening** of lines perpendicular to the xy plane.

$$\therefore \quad f = 1$$

$$\text{and} \quad \theta = 45° \qquad \text{... given}$$

$$\therefore \quad Par_V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is the cavalier projection transformation for $\theta = 45°$.

b) A cabinet projection is an oblique projection with f = ½. For $\theta = 30°$

we have,

$$Par_V = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{3}}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is the cabinet projection transformation for $\theta = 30°$.

c) Now to draw the projections, we first represent the vertices of the unit cube by a matrix whose rows are homogeneous coordinates of the vertices

$$V = \begin{bmatrix} A_1 \\ B_1 \\ C_1 \\ D_1 \\ E_1 \\ F_1 \\ G_1 \\ H_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

To draw the cavalier projection, we find the image coordinates by applying the transformation matrix Par_V to the coordinate matrix V.

$$\therefore \begin{bmatrix} A_2 \\ B_2 \\ C_2 \\ D_2 \\ E_2 \\ F_2 \\ G_2 \\ H_2 \end{bmatrix} = V \cdot Par_V$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1/\sqrt{2} & 1+1/\sqrt{2} & 0 & 1 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 & 1 \\ 1+1/\sqrt{2} & 1/\sqrt{2} & 0 & 1 \\ 1+1/\sqrt{2} & 1+1/\sqrt{2} & 0 & 1 \end{bmatrix}$$

Hence the image coordinates are

$$A_2 = (0, 0, 0) \qquad E_2 = (1/\sqrt{2}, 1 + 1/\sqrt{2}, 0)$$
$$B_2 = (1, 0, 0) \qquad F_2 = (1/\sqrt{2}, 1/\sqrt{2}, 0)$$
$$C_2 = (1, 1, 0) \qquad G_2 = (1 + 1/\sqrt{2}, 1/\sqrt{2}, 0)$$
$$D_2 = (0, 1, 0) \qquad H_2 = (1 + 1/\sqrt{2}, 1 + 1/\sqrt{2}, 0)$$

To draw the cabinet projection, we find the image coordinates by applying the transformation matrix Par_V to the coordinate matrix V.

$$\therefore \begin{bmatrix} A_2 \\ B_2 \\ C_2 \\ D_2 \\ E_2 \\ F_2 \\ G_2 \\ H_2 \end{bmatrix} = V \cdot Par_V$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ \sqrt{3}/4 & 1+1/4 & 0 & 1 \\ \sqrt{3}/4 & 1/4 & 0 & 1 \\ 1+\sqrt{3}/4 & 1/4 & 0 & 1 \\ 1+\sqrt{3}/4 & 1+1/4 & 0 & 1 \end{bmatrix}$$

Hence the image coordinates are

$A_2 = (0, 0, 0)$ $\quad E_2 = (\sqrt{3}/4, 1 + 1/4, 0)$

$B_2 = (1, 0, 0)$ $\quad F_2 = (\sqrt{3}/4, 1/4, 0)$

$C_2 = (1, 1, 0)$ $\quad G_2 = (1 + \sqrt{3}/4, 1/4, 0)$

$D_2 = (0, 1, 0)$ $\quad H_2 = (1 + \sqrt{3}/4, 1+1/4, 0)$

Review Questions

1. Explain the 3D viewing process.
2. Explain various 3D viewing parameters.
3. Derive the 3D transformation matrix to transform world coordinates to viewing coordinates.
4. Write a short note on parallel projection.
5. Write a short note on perspective projection.
6. Explain various types of parallel projections.
7. Explain various types of perspective projections.
8. Derive the transformation matrix for general parallel projection.
9. Derive the transformation matrix for perspective projection.
10. What is the necessity of 3D clipping algorithm ?
11. Explain midpoint subdivision algorithm for 3D clipping.

University Questions

1. Give a mathematical description of the perspective projection. **(Dec-96)**
2. Consider a cube of side 5 units placed such that the co-ordinate axes s, y and z are along the cube edges and the origin is at one corner of the cube. Assume that the centre of projection (COP) is at (0, 0 – d) and the view plane is the x y plane. Draw the projected image using perspective transformations for d = 5 and d 25. **(Dec-96)**
3. Write a detailed note on perspective projections **(May-97)**
4. With suitable examples and appropriate mathematical models, explain various perspective projections. **(Dec-97, May-2000, Dec-2000)**
5. Consider a cube of side 10 units placed such that the co-ordinate axes x, y, and z are parallel along the cube edges and the origin is at the centre of the cube.

   Assume that the centre of projection (COP) is at (0, 0,- d) and the view plane is in the x-y plane. Evaluate and draw the projected image using perspective transformation for d = 10 and d = 30. **(Dec-97)**
6. Differentiate between parallel and perspective projection. Generate a homogeneous matrix representation for oblique projection of co-ordinate position (x, y, z) to position (x_1, y_1) on the view plane. **(May-98)**
7. Write a detailed note on three dimensional clipping w.r.t. view volumes. **(May-98, May-2000)**
8. What are different types of projection ? Derive an matrix representation for perspective transformation ? What are different perspective anomalies ? **(Dec-98)**
9. For an standard perspective projection with COP at (0, 0, – d), what is the projected image of –

   i) A point in the plane z = – d

   ii) The line segment joining P_1 (– 1, 1, – 2d) to P_2(2, – 2, 0) **(Dec-98)**
10. How normalised view volume is converted into regular shaped form ? What advantage we will have, if the conversion takes place before clipping ? **(Dec-98)**
11. 3-dimensional view port ? How line clipping is done against it ? **(Dec-98)**
12. A tetrahedron of size 10 units is placed on xy plane with one edge along X-axis (+ ve) and one vertex at origin. Assuming the tetrahedron to be opaque evaluate and draw projected image if centre of projection is (10, 0, 0) **(May-99)**
13. Compare parallel and perspective projections with reference to practical use only. **(May-99)**
14. Identify the building blocks that implement the 3D - viewing as shown below : **(Dec-99)**
15. Compare and contrast parallel and perspective projection techniques. **(Dec-99)**
16. A tetrahedron of size 10 units is placed on the x-y plane with one edge along the x-axis and the origin at the centre of the object. Assuming the COP at (5, 0, 0), draw the projected image using perspective projection transformation. **(May-2000)**

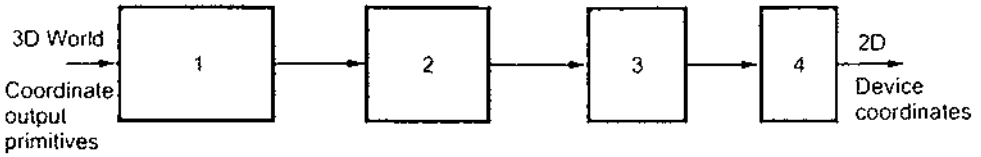


Fig. 7.26

17. Consider a cube of side 10 units placed such that the c-ordinate axes x, y and z are along the cube edges and the origin is at one corner of the cube. Assume that the centre of projection (COP) is at (0, 0 – d) and the view plane is the x y plane. Draw the projected image using perspective transformations for d = 10 and d = 30 **(Dec-2000)**

18. Identify the building blocks that implement the 3D-viewing as shown below. **(May-2001)**

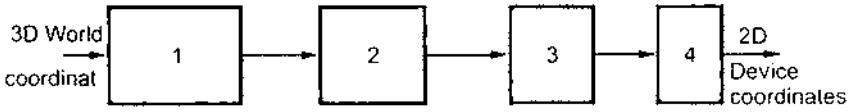


Fig. 7.27

19. What are different types of projections ? Generate a homogeneous matrix representation for orthogonal projection and oblique projection of co-ordinate position P(x, y, z) on to the xy plane. **(May-2001)**

20. Develop the perspective transformation of an object onto the xy plane with center of projection at cop (100, 100, – 100). What will be the projection of a line segment A (150, 250, 150) B(250, 350, 100) **(Dec-2001)**

21. What do you understand from the terms parallel projection, perspective projection, orthographic projection, oblique projection, Cavalier projection, cabinet projection, vanishing point, principal vanishing point ? **(May-2002)**

22. Define orthographic projection and oblique projection.

    Derive mathematical model and transformation matrix for parallel projection. **(May-2003)**

23. Define the following terms :

    Vanishing point, World coordinates, View volume, Homogeneous coordinates. **(May-2003)**

❑❑❑

8

Hidden Surface Elimination Methods

8.1 Introduction

For generation of realistic graphics displays, we have to identify those parts of a scene that are visible from a chosen viewing position. There are many algorithms called **visible surface algorithms** developed to solve this problem. In early days of computer graphics visible surface algorithms were called **hidden line** or **hidden surface algorithms**.

In a given set of 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as hidden surfaces or hidden line elimination, or visible surface determination. The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space. These algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called **object-space methods** or **object precision methods** and **image-space methods**, respectively.

Object-space Method : Object-space method is implemented in the **physical coordinate system** in which objects are described. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. Object-space methods are generally used in **line-display algorithms**.

Image-Space Method : Image space method is implemented in the **screen coordinate system** in which the objects are viewed. In an image-space algorithm, visibility is decided point by point at each pixel position on the view plane. Most **hidden line/surface algorithms** use image-space methods.

In this chapter we are going to study various visible surface detection or hidden line removal algorithms, algorithms for octrees, algorithms for curved surfaces, and visible-surface ray tracing.

8.2 Techniques for Efficient Visible-Surface Algorithms

We have seen that there are two basic approaches used for visible surface detection : Object precision algorithm and image precision algorithm. In both the algorithms we require to perform a number of potentially costly operations such as determination of projections of objects, whether or not they intersect and where they intersect, closest object in case of intersection and so on. To create and display picture in minimum time we have to perform visible surface algorithms more efficiently. The techniques to perform visible-surface algorithms efficiently are discussed in the following sections.

8.2.1 Coherence

The coherence is defined as the degree to which parts of an environment or its projection exhibit local similarities. Such as similarities in depth, colour, texture and so on. To make algorithms more efficient we can exploit these similarities (coherence) when we reuse calculations made for one part of the environment or a picture for other nearby parts, either without changes or with some incremental changes. Let us see different kinds of coherence we can use in visible surface algorithms.

- **Object coherence :** If one object is entirely separate from another, comparisons may need to be done only between the two objects, and not between their components faces or edges.
- **Face coherence :** Usually surface properties vary smoothly across a face. This allows the computations for one part of face to be used with incremental changes to the other parts of the face.
- **Edge coherence :** The visibility of edge may change only when it crosses a visible edge or penetrates a visible face.
- **Implied edge coherence :** If one planar face penetrates another their line of intersection can be determined from two points of intersection.
- **Area coherence :** A group of adjacent pixel is often belongs to the same visible face.
- **Span coherence :** It refers to a visibility of face over a span of adjacent pixels on a scan line. It is special case of area coherence.
- **Scan line coherence :** The set of visible object spans determined for one scan line of an image typically changes very little from the set on the previous line.
- **Depth coherence :** Adjacent parts of the same surface are typically same or very close depth. Therefore, once the depth at one point of the surface is determined the depth of the points on the rest of the surface can often be determined by at the most simple incremental calculation.
- **Frame Coherence :** Pictures of the same scene at two successive points in time are likely to be quite similar, except small changes in objects and view ports. Therefore, the calculations made for one picture can be reused for the next picture in a sequence.

8.2.2 Perspective Transformation

Visible-surface determination is done in a 3D space prior to the projection into 2D that destroys the depth information needed for depth comparisons, and depth comparisons are typically done after the normalizing transformation. Due to this projectors are parallel to the

Z axis in parallel projections or emanate from the origin in perspective projections. In parallel projection, when $x_1 = x_2$ and $y_1 = y_2$ we can say that points are on the same projector. However, in perspective projection we have to perform four divisions : $x_1 / z_1 = x_2 / z_2$ and $y_1 / z_1 = y_2 / z_2$ to determine whether the points are on the same projector. These divisions can be avoided by first transforming a 3D object into the 3D screen-coordinate system, so that the parallel projection of the transformed object is the same as the perspective projection of the untransformed object.

8.2.3 Extents and Bounding Volumes

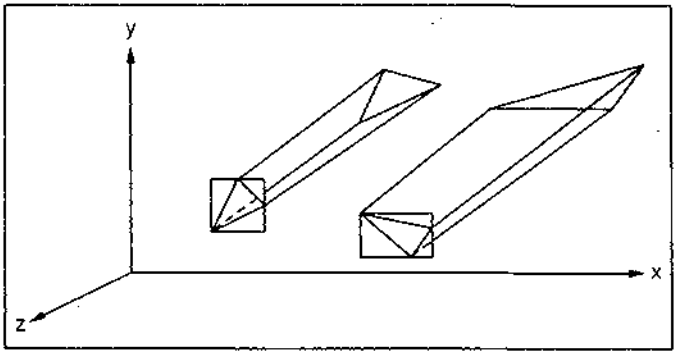


Fig. 8.1

The Fig. 8.1 shows two objects with their projections and the rectangular screen extents surrounding the projections.

It is easier to check the overlapping of extents than the projections, and we can say that when extends are not overlapping then projections are also not overlapping. Therefore, extents must be compared first and then the projections must be compared only if the extents are overlapped. This avoids unnecessary comparisons in checking the overlapping of projections if extents are not overlapped.

Extents can be used to surround the object themselves rather than their projections; in this case the extents become solid and are commonly known as **bounding volumes**. In this case extents can be used to check whether two objects are overlapped or not.

Extents and bounding volumes are used not only to compare two objects or their projections with each other, but also to determine whether or not a projector intersects an object.

8.2.4 Back-Face Culling

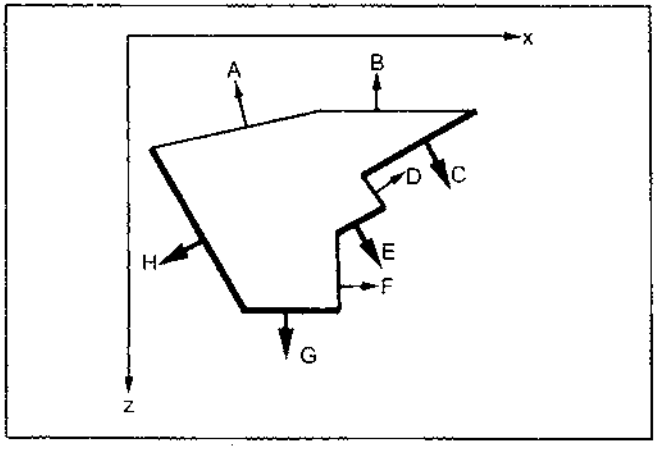


Fig. 8.2 Back face culling

When an object is approximated by a solid polyhedron, its polygonal faces completely enclose its volume. In such case, if none of the polyhedron's interior is exposed by the front clipping plane, then those polygons whose surface normals point away from the observer lie on a part of the polyhedron whose visibility is completely blocked by other closer polygons, as shown in Fig. 8.2.

As shown in the Fig. 8.2, polygons (A, B, D, F) shown in gray are eliminated, where as front-facing polygon (C, E, G, H) are retained.

These invisible back facing polygons can be eliminated from further processing. Such a technique to eliminate the back facing polygon from further processing is known as **Back-face Culling.**

8.2.5 Spatial Partitioning

In this technique, subdivision rule is applied to break down a large problem into a number of smaller ones. In this objects and their projections are assigned to spatically coherent groups as a preprocessing step. This partitioning speed up the process of determining which object intersect with a projector. Because now it is necessary to test only the objects lying within the partitions which intersect with projector.

When objects are unequally distributed in space the adaptive partitioning is more suitable. Because it allows variable size of each partition.

8.2.6 Hierarchy

In hierarchical structure different levels are assigned to the object and their is a parent-child relationship between the objects, as shown in the Fig. 8.3.

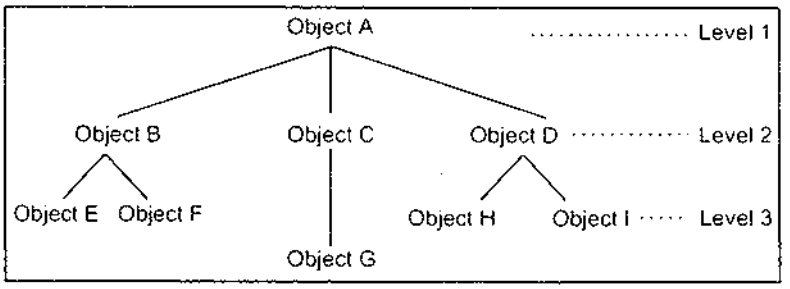


Fig. 8.3 Hierarchical structure of objects

In this structure, each child is considered as a part of its parent. This allows to restrict the number of object comparisons needed by a visible-surface algorithm. If the parent level object is fail to intersect, the lower level (child) objects belongs to that parent do not need to be tested for intersection.

8.3 Hidden Line Elimination Algorithms

In this section we see some algorithms for hidden line elimination.

8.3.1 Robert's Algorithm

The earliest visible-line algorithm was developed by Roberts. The primary requirement of this algorithm is that each edge be part of the face of a convex polyhedron. In the phase of this algorithm the all edges shared by a pair of polyhedron's back facing polygons are removed using back-face culling technique. In the next phase, each remaining edge is compared with each polyhedron that might obscure it. Because the polyhedra are convex, there is at most one contiguous group of points on any line that is blocked from the observer by any polyhedron. Thus each polyhedron either obscures the edge totally or causes one or two pieces of the remain. Then any remaining pieces of the edge are compared with the next polyhedron.

8.3.2 Appel's Algorithm

Arthur Appel introduced the quantitative invisibility of a line. He defined the quantitative invisibility as the number of potentially visible surfaces that lie between the line segment and the eyepoint. When a line passes behind a front facing polygon, its quantitative invisibility is incremented by 1 and when it passes out from behind that polygon, its quantitative invisibility is decremented by 1. A line is visible only when its quantitative invisibility is 0. This is illustrated in Fig. 8.4. Here, line AB is annotated with the quantitative invisibility of each of its segments.

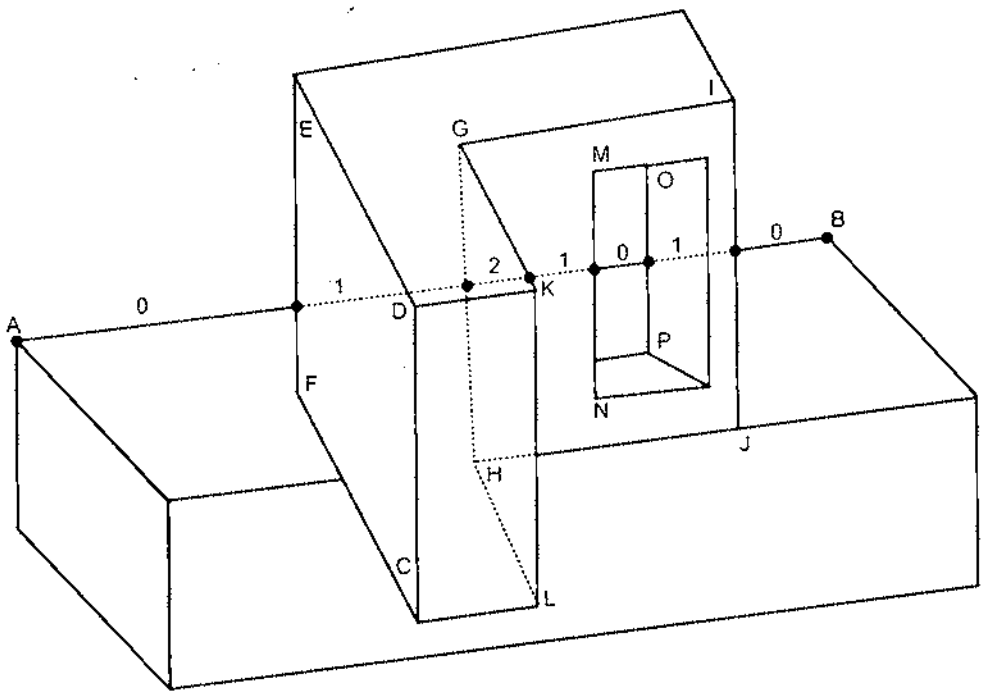


Fig. 8.4

Appel defined **contour line** as an edge shared by a front-facing and a back-facing polygon, or unshared edge of a front facing polygon that is not part of a closed polyhedron. An edge shared by two front-facing polygons causes no change in visibility and therefore is not a contour line. In Fig. 8.4, edges AB, EF, FC, GK and GH are contour lines, whereas edges ED, DC and GI are not. As shown in the Fig. 8.4 line AB begins with a quantitative invisibility of zero, passes behind contour line EF, where the quantitative invisibility increases to +1, then behind the contour line GH, where it increases to + 2. Passing behind contour line GK reduces the quantitative invisibility to +1. The line then emerges from behind contour line MN after which the quantitative invisibility is zero. It then passes behind contour line OP where the quantitative invisibility increases to +1. When it passes behind the contour IJ its quantitative invisibility is again zero. Therefore, the portions of line AB (portions from A to the contour line EF, from contour line MN to OP and from contour line IJ to B) with a quantitative invisibility of zero are visible.

8.3.3 Haloed Lines

In many applications the hidden lines are shown as dotted, dashed, lines with lower intensity or with some other rendering style supported by the display device. But in such application hidden lines are not totally suppressed. The haloed line algorithm described by Appel, Rohlf, and Stein can suppress hidden lines. The algorithm surrounds each line on both sides by a halo that obscures those parts of lines passing behind it. The Fig. 8.5 shows the haloed line segment. It has a symmetrically placed rectangle of width 2H with semicircular ends of radius H. If line A in Fig. 8.5 is closer to the viewpoint than line B, then the segment within the halo, labelled C, is not drawn.

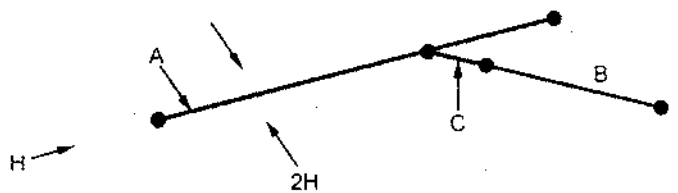


Fig. 8.5 Haloed line segment

This clears that the algorithm intersects each line with those passing in front of it, keeps track of those sections that are obscured by halos, and draws the visible sections of each line after the intersections have been calculated. If the halos are wider than the spacing between lines, then an effect similar to conventional hidden-line elimination is achieved.

Fig. 8.6 Hidden lines elimination by Haloed line algorithm

8.4 Hidden Surface Elimination Algorithms

In this section we see some algorithms for hidden surface elimination.

8.4.1 Painter's Algorithm (Depth Sort Algorithm)

The basic idea of the painter's algorithm developed by Newell and Sancha, is to paint the polygons into the frame buffer in order of decreasing distance from the viewpoint. This process involves following basic functions.

1. Sorting of polygons in order of decreasing depth.
2. Resolving any ambiguities this may cause when the polygon's z extents overlap, i.e., splitting polygons if necessary.
3. Scan conversion of polygons in order, starting with the polygon of greatest depth.

The algorithm gets its name from the manner in which an oil painting is created. The artist begins with the background. He then adds the most distant object and then the nearer object and so forth. There is no need to erase portions of background; the artist simply paints on top of them. The new paint covers the old so that only the newest layer of paint is visible. This is illustrated in Fig. 8.7.

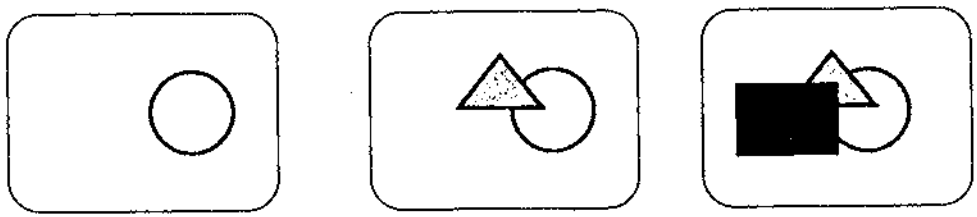


Fig. 8.7 Painter's algorithm

Using the similar technique, we first sort the polygons according to their distance from the view point. The intensity values for the farthest polygon are then entered into the frame buffer. Taking each polygon is succeeding polygon in turn (in decreasing depth order), polygon intensities are painted on the frame buffer over the intensities of the previously processed polygons. This process is continued as long as no overlaps occur. If depth overlap is detected by any point in the sorted list, we have to make some additional comparisons to determine whether any of the polygon should be reordered. We can check whether any polygon Q does not obscure polygon P by performing following steps :

1. The z-extents of P and Q do not overlap, i.e. $z_{Q\,max} < z_{P\,min}$ (see Fig. 8.8 (a))
2. The y-extents of P and Q do not overlap (see Fig. 8.8 (b))
3. The x-extents of P and Q do not overlap
4. Polygon P lying entirely on the opposite side of Q's plane from the view port. (see Fig. 8.8 (c))
5. Polygon Q lying entirely on the same side of P's plane as the viewport. (see Fig. 8.8 (d)).
6. The projections of the polygons P and Q onto the xy screen do not overlap.

If all these five tests fail, we assume for the moment that P actually obscures Q, and therefore test whether Q can be scan-converted before P. Here, we have to repeat tests 4 and 5 for Q. If these tests also fail then we can say that there is no order in which P and Q can be scan converted correctly and we have to split either P or Q into two polygons. The idea behind the splitting is that the splitted polygons may not obscure other polygon.

Algorithm

1. Sort all polygons in order of decreasing depth.
2. Determine all polygons Q (preceding P) in the polygon list whose z-extents overlap that of P.
3. Perform test 2 through 6 for each Q

a) If every Q passes the tests, scan convert the polygon P.

b) If test fails for some Q, swap P and Q in the list, and make the indication that Q is swapped. If Q has already been swapped, use the plane containing polygon P to divide polygon Q into two polygons, Q_1 and Q_2. Replace Q with Q_1 and Q_2. Repeat step 3.

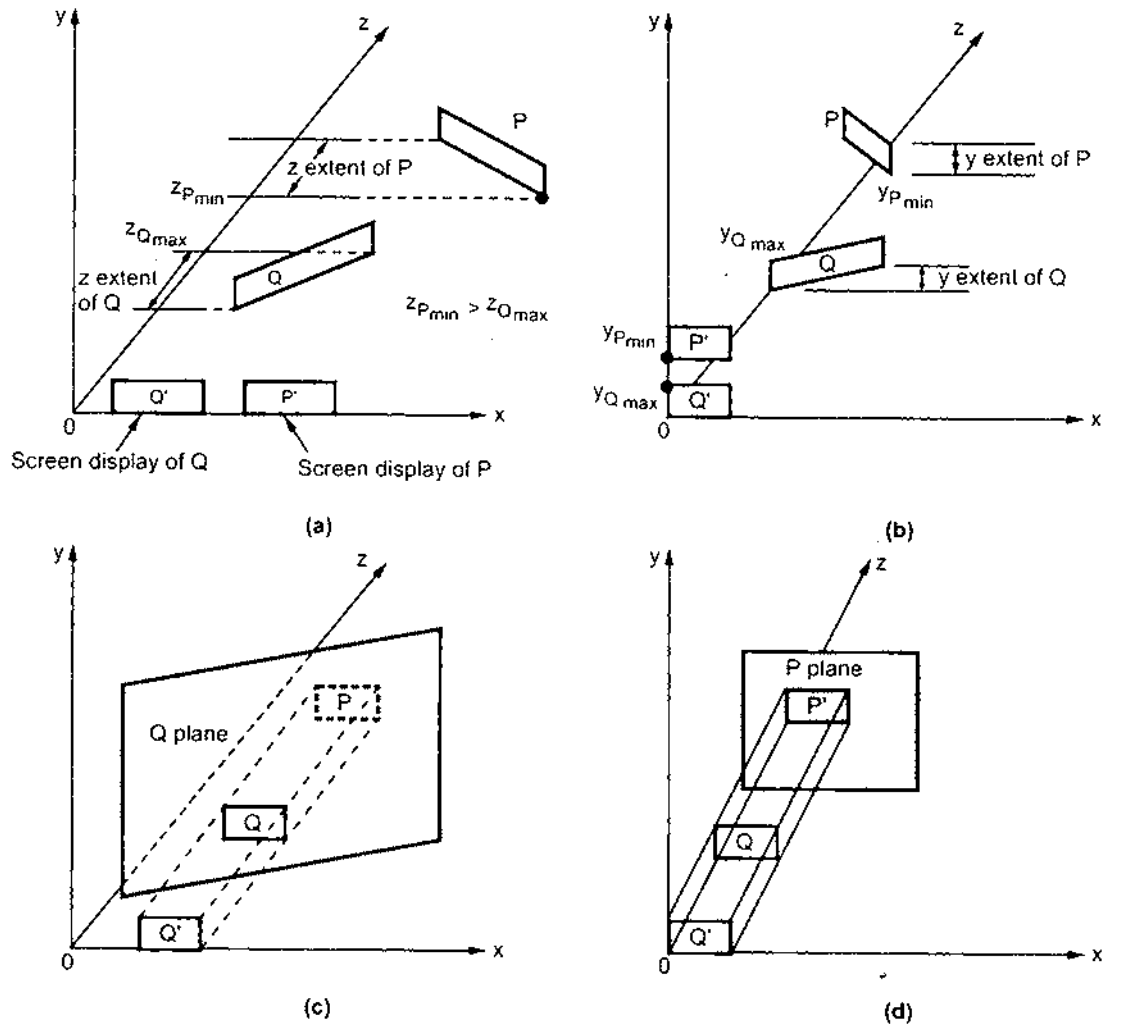


Fig. 8.8

8.4.2 Scan Line Algorithm

A scan line method of hidden surface removal is an another approach of image space method. It is an extension of the scan line algorithm for filling polygon interiors. Here, the algorithm deals with more than one surfaces. As each scan line is processed, it examines all polygon surfaces intersecting that line to determine which are visible. It then does the depth calculation and finds which polygon is nearest to the view plane. Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.

know that scan line algorithm maintains the active edge list. This active edge list ns only edges that cross the current scan line, sorted in order of increasing x. The scan line method of hidden surface removal also stores a flag for each surface that is set on or off to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned ON; and at the rightmost boundary, it is turned OFF.

The Fig. 8.9 illustrates the scan line method for hidden surface removal. As shown in the Fig. 8.9, the active edge list for scan line 1 contains the information for edges AD, BC, EH and FG. For the positions along this scan line between edges AD and BC, only the flag for surface S_1 is ON. Therefore, no depth calculations are necessary, and intensity information for surface S_1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is ON and during that portion of scan line the intensity information for surface S_2 is entered into the frame buffer.

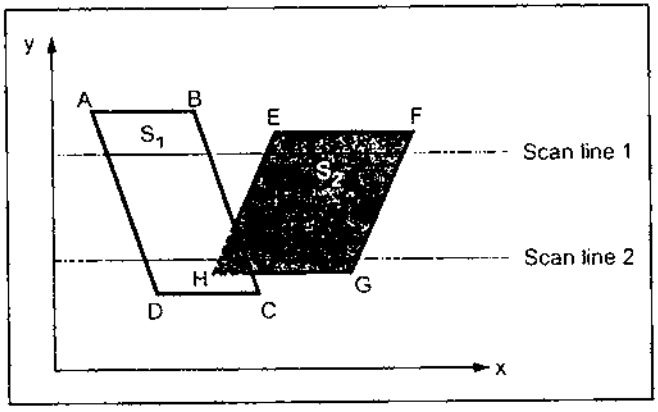


Fig. 8.9 Illustration of scan line method of hidden surface removal

For scan line 2 in the Fig. 8.9, the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S_1 is ON. However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations are necessary. Here we have assumed that the depth of S_1 is less than the depth of S_2 and hence the intensities of surface S_1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S_2 are entered into the frame buffer because during that portion only flag for S_2 is ON.

8.4.3 Z-Buffer Algorithm

One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer** or **depth buffer** algorithm. It is developed by Catmull. This algorithm compares surface depths at each pixel position on the projection plane. The surface depth is measured from the view plane along the z axis of a viewing system. When object description is converted to projection coordinates (x, y, z), each pixel position on the view plane is specified by x and y coordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z- values.

The Z-buffer algorithm is usually implemented in the normalized coordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane. The implementation requires another buffer memory called Z-buffer along with the frame buffer memory required for raster display devices. A Z-buffer is used to store depth values for each

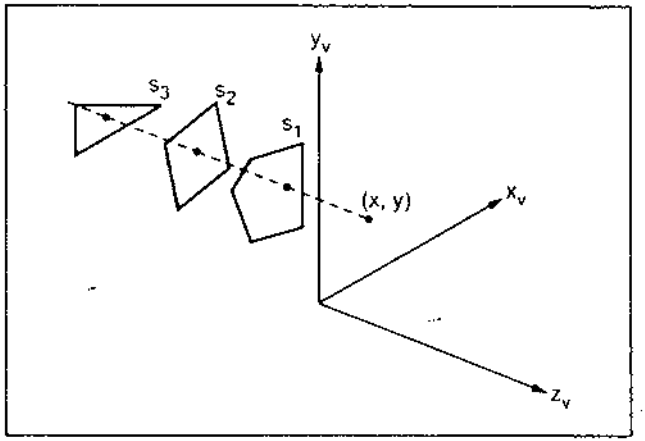


Fig. 8.10

(x, y) position as surfaces are processed, and the frame buffer stores the intensity values for each position. At the beginning Z-buffer is initialized to zero, representing the z-value at the back clipping plane, and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time, calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position. If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored, and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.

For example, in Fig. 8.10 among three surfaces, surface S_1 has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

Z-buffer Algorithm

1. Initialize the Z-buffer and frame buffer so that for all buffer positions

   Z-buffer (x, y) = 0 and frame-buffer (x, y) = $I_{background}$
2. During scan conversion process, for each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

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

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

   Z-buffer (x, y) = z, frame-buffer (x, y) = $I_{surface}$ (x, y)
3. Stop

Note that, $I_{background}$ is the value for the background intensity, and $I_{surface}$ is the projected intensity value for the surface at pixel position (x, y). After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.

To calculate z-values, the plane equation

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

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

Therefore, we can write

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

Note, if at (x, y) the above equation evaluates to z_1, then at (x + Δx, y) the value of z, is

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

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

Advantages

1. It is easy to implement.
2. It can be implemented in hardware to overcome the speed problem.
3. Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages

1. It requires an additional buffer and hence the large memory.
2. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

8.4.4 Warnock's Algorithm (Area Subdivision Algorithm)

An interesting approach to the hidden-surface problem was developed by Warnock. He developed area subdivision algorithm which subdivides each area into four equal squares. At each stage in the recursive-subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships :

1. Surrounding Polygon - One that completely encloses the (shaded) area of interest (see Fig. 8.11 (a))
2. Overlapping or Intersecting Polygon - One that is partly inside and partly outside the area (see Fig. 8.11 (b))
3. Inside or Contained Polygon - One that is completely inside the area (see Fig. 8.11 (c)).
4. Outside or Disjoint Polygon - One that is completely outside the area (see Fig. 8.11 (d)).

(a) Surrounding (b) Overlapping (c) Inside or Contained (d) Outside or Disjoint

Fig. 8.11 Possible relationships with polygon surfaces and the area of interest

After checking four relationships we can handle each relationship as follows :

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
2. If there is only one intersecting or only one contained polygon, then the area is first filled with the background colour, and then the part of the polygon contained in the area is filled with colour of polygon.
3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.
4. If there are more than one polygon intersecting, contained in, or surrounding the area then we have to do some more processing.

See Fig. 8.12. In Fig. 8.12 (a), the four intersections of surrounding polygon are all closer to the viewpoint than any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

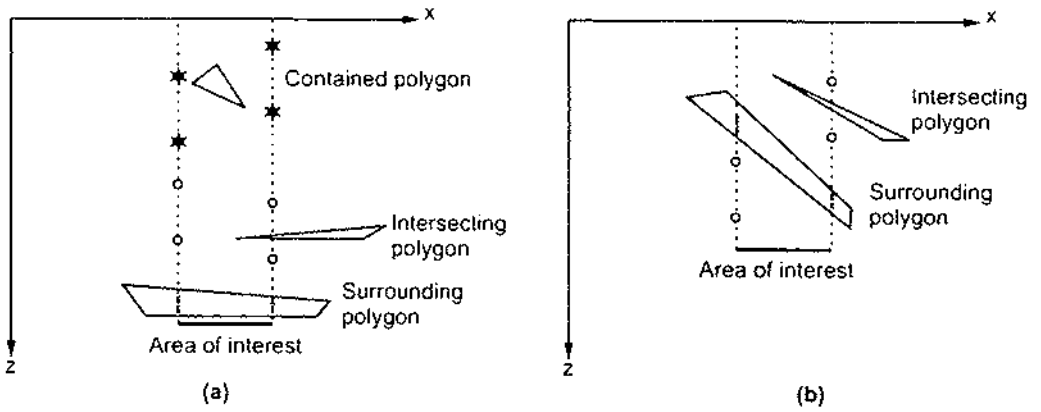


Fig. 8.12

However, Fig. 8.12 (b) shows that surrounding polygon is not completely in front of the intersecting polygon. In such case we cannot make any decision and hence Warnock's algorithm subdivides the area to simplify the problem. This is illustrated in Fig. 8.13. As shown in the Fig. 8.13 (a) we can not make any decision about which polygon is in front of the other. But after dividing area of interest polygon 1 is ahead of the polygon 2 in left area and polygon 2 is ahead of polygon 1 in the right area. Now we can fill these two areas with corresponding colours of the polygons.

The Warnock's algorithm stops subdivision of area only when the problem is simplified or when area is only a single pixel.

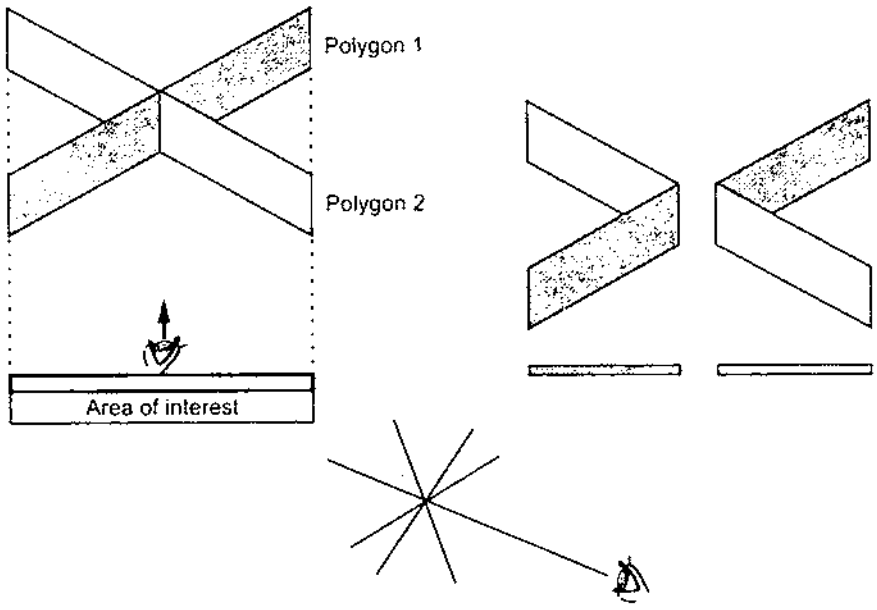


Fig. 8.13

Algorithm

1. Initialize the area to be the whole screen.
2. Create the list of polygons by sorting them with their z-values of vertices. Don't include disjoint polygons in the list because they are not visible.
3. Find the relationship of each polygon.
4. Perform the visibility decision test
   a) If all the polygons are disjoint from the area, then fill area with background colour.
   b) If there is only one intersecting or only one contained polygon then first fill entire area with background colour and then fill the part of the polygon contained in the area with the colour of polygon.
   c) If there is a single surrounding polygon, but no intersecting or contained polygons, then fill the area with the colour of the surrounding polygon.
   d) If surrounding polygon is closer to the viewpoint than all other polygons, so that all other polygons are hidden by it, fill the area with the colour of the surrounding polygon.
   e) If the **area is the pixel** (x, y), and neither a,b,c, nor d applies, compute the z coordinate at pixel (x, y) of all polygons in the list. The pixel is then set to colour of the polygon which is closer to the viewpoint.
5. If none of the above tests are true then subdivide the area and go to step 2.

Advantages

1. It follows the divide-and-conquer strategy, therefore, parallel computers can be used to speed up the process.

2. Extra memory buffer is not required.

8.4.5 Back-Face Removal Algorithm

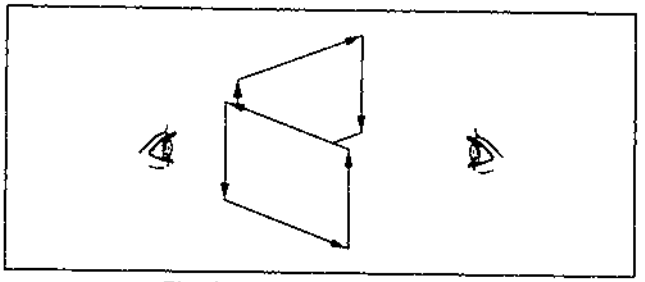


Fig. 8.14 Drawing directions

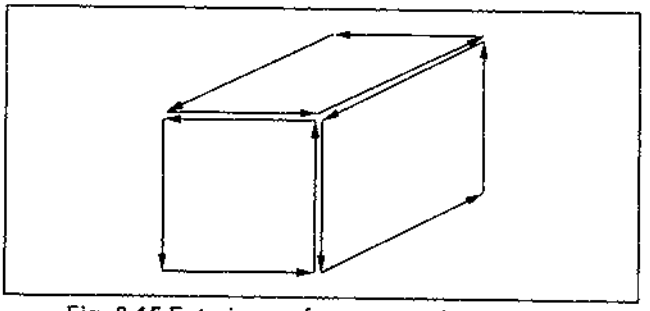


Fig. 8.15 Exterior surfaces are coloured light and drawn counter clockwise

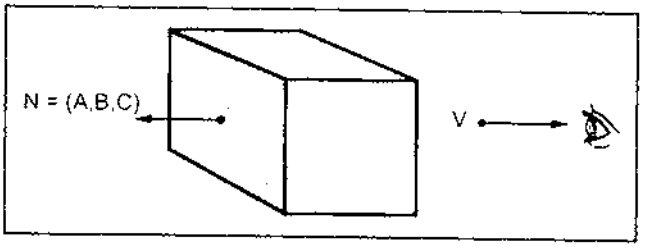


Fig. 8.16

We know that a polygon has two surfaces, a front and a back, just as a piece of paper does. We might picture our polygons with one side painted light and the other painted dark. But the question is "how to find which surface is light or dark". When we are looking at the light surface, the polygon will appear to be drawn with counter clockwise pen motions, and when we are looking at the dark surface the polygon will appear to be drawn with clockwise pen motions, as shown in the Fig. 8.14.

Let us assume that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to the air; the dark faces meet the material inside the object. This means that when we look at an object face from the outside, it will appear to be drawn counterclockwise, as shown in the Fig. 8.15.

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

The direction of the light face can be identified by examining the result

$$N \cdot V > 0$$

where

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

V : A vector in the viewing direction from the eye (or "camera") position (Refer Fig. 8.16)

We know that, the dot product of two vector gives the product of the lengths of the two vectors times the cosine of the angle between them. This cosine factor is important to us because if the vectors are in the same direction ($0 \le \theta < \pi/2$), then the cosine is positive and the overall dot product is positive. However, if the directions are opposite ($\pi/2 < \theta \le \pi$), then the cosine and the overall dot product is negative (Refer Fig. 8.17).

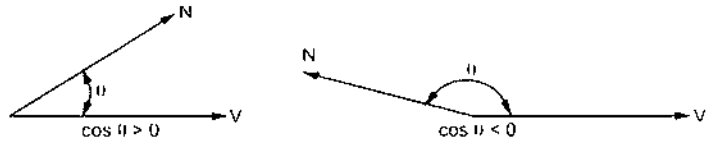


Fig. 8.17 Cosine angles between two vectors

If the dot product is positive, we can say that the polygon faces towards the viewer; otherwise it faces away and should be removed.

In case, if object description has been converted to projection coordinates and our viewing direction is parallel to the viewing z_v axis, then $V = (0, 0, V_z)$ and

$$V \cdot N = V_z C$$

So that we only have to consider the sign of C, the Z component of the normal vector N. Now, if the z component is positive, then the polygon faces towards the viewer, if negative, it faces away.

Review Questions

1. Explain the two approaches used to determine hidden surfaces.
2. Discuss the techniques for efficient visible-surface algorithms.
3. What is coherence ? Discuss various types of coherence that can be used to make visible surface algorithms more efficient.
4. Write a short note on
   a) Perspective transformation
   b) Extents and bounding volumes
   c) Back-face culling
5. Explain the Robert's visible line algorithm.
6. Explain the Appel's visible line algorithm.
7. Explain the Haloed line algorithm.
8. Explain the painter's algorithm for hidden surface removal.
9. Explain the scanline algorithm for hidden surface removal.
10. Explain the Z-buffer algorithm for hidden surface removal.
11. List the advantages and disadvantages of Z-buffer algorithm.
12. Explain any one area subdivision algorithm for visible surface detection.
13. Describe the back face removal algorithm.

University Questions

1. What do you understand by hidden line and hidden surface elimination techniques ? Develop an algorithm for any hidden surface algorithm of your choice. Clearly state its advantages and drawbacks over others. (Dec-96)
2. Develop an algorithm for removal of hidden surfaces. Illustrate this on the object shown below :- (Dec-96, Dec-2000)

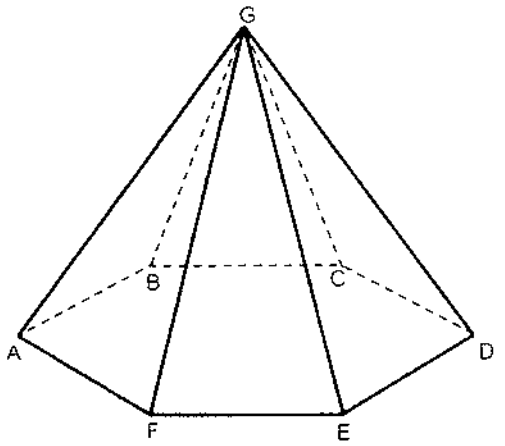


Fig. 8.18

Comment on the advantages and disadvantages of your method with respect to any other method you have studied.

3. Apply any hidden surface/hidden line algorithm on the above wire-frame model and draw the resultant, assuming that the view point is at (0, 2h, 0) (May-97, Dec-97)
4. Differentiate between image space and object space. Explain painters algorithm for hidden surface removal. (May-98)
5. Explain scan-line method for hidden surface removal and explain how it works for figure shown below : (May-98)

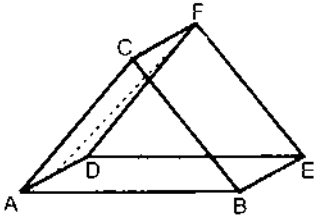


Fig. 8.19

6. Write a detailed note on z-buffer method (May-98)
7. Explain how area subdivision method is used for hidden surface elimination. (Dec-98, May-2001)
8. Explain a-buffer algorithm, state its advantages over z-buffer algorithm. (Dec-98)
9. Explain z-buffer method. (May-99)

10. Compare and contrast hidden surface removal algorithms based on object space and Image space approaches. (Dec-99)

11. Develop an appropriate hidden surface algorithm which is suitable for the above object. (May-2000)

12. Develop a program for the z-buffer technique. (May-2000)

13. Differentiate between image space and object space methods of hidden surface removal. Explain z buffer method of hidden surface removal with reference to the folowing object. (Dec-2001)

14. Explain depth buffer method of hidden surface removal in detail. (May-2001)

15. Apply scanline method of hidden surface removal on the following figure and do the following. (May-2001)

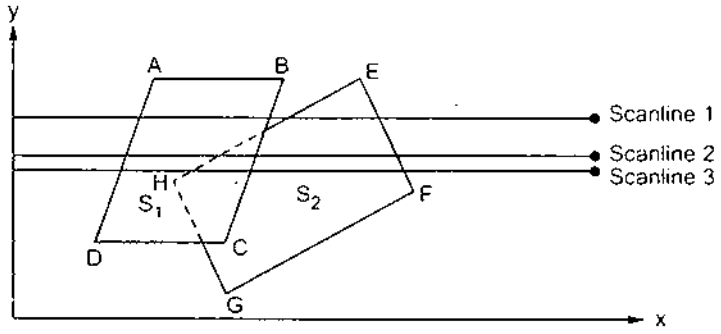


Fig. 8.20

i) Give the active edge list for each scanline.

ii) Mention the possible intensities of each scanline.

16. Differentiate between : image-space method and object-space method. (May-2003)

□□□

9

Curves

9.1 Introduction

In chapters 2 and 3 we have seen line, circle and polygon generation algorithms. In later chapters, we learned how to represent and manipulate line segments and polygons. We have also seen transformations and clipping of them. However, many real world objects are inherently smooth and involve curves to represent them. Some natural objects are neither perfectly flat nor smoothly curved but often have rough, jagged contours.

In this chapter, we see the methods for generating curved lines. In the later part of the chapter we discuss the procedures to draw fractal curves, lines and surfaces.

9.2 Generation of Curves

We can use two approaches to draw curved lines. One approach is to use a curve generation algorithm such as DDA. In this approach a true curve is created. In the second approach the curve is approximated by a number of small straight line segments. This can be achieved with the help of interpolation techniques.

9.2.1 Circular Arc Generation Using DDA Algorithm

Digital differential analyzer algorithm uses the differential equation of the curve. The differential equations for simple curve such as circle is fairly easy to solve and we have already discussed it in the chapter 2. Let us see the DDA algorithm for generating circular arcs. The equation for an arc in the angle parameters can be given as

$$x = R\cos\theta + x_0$$
$$y = R\sin\theta + y_0 \qquad \ldots (9.1)$$

where (x_0, y_0) is the center of curvature, and R is the radius of arc. (see Fig. 9.1)

Differentiating equation 9.1 we get

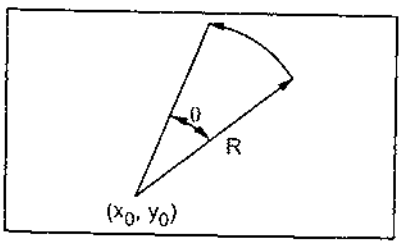


Fig. 9.1

$$dx = -R\sin\theta\, d\theta$$
$$dy = R\cos\theta\, d\theta \qquad \ldots (9.2)$$

From equation 9.1 we can solve for R cosθ and R sinθ as follows.

$$x = R\cos\theta + x_0$$
$$\therefore \quad R\cos\theta = x - x_0 \text{ and}$$

$$R \sin\theta = y - y_0 \quad \ldots (9.3)$$

Substituting values of $R \cos\theta$ and $R \sin\theta$ from equation 9.3 in equation 9.2 we get,

$$dx = -(y - y_0)\, d\theta \text{ and}$$

$$dy = (x - x_0)\, d\theta \quad \ldots (9.4)$$

The values of dx and dy indicate the increment in x and y increment, respectively, to be added in the current point on the arc to get the next point on the arc. Therefore, we can write

$$x_2 = x_1 + dx = x_1 - (y_1 - y_0)\, d\theta$$

$$y_2 = y_1 + dy = y_1 + (x_2 - x_0)\, d\theta \quad \ldots (9.5)$$

The equation 9.5 forms the basis for arc generation algorithm. From equation 9.5 we can see that the next point on the arc is the function of $d\theta$. To have a smooth curve, the neighbouring points on the arc should be close to each other. To achieve this, the value of $d\theta$ should be small enough not to leave gaps in the arc. Usually, the value of $d\theta$ can be determined from the following equation.

$$d\theta = \text{Min}\,(0.01,\ 1/(3.2 \times (|x - x_0| + |y - y_0|)))$$

Algorithm

1. Read the center of a curvature, say (x_0, y_0)
2. Read the arc angle, say θ
3. Read the starting point of the arc, say (x, y)
4. Calculate $d\theta$

   $d\theta = \text{Min}\,(0.01,\ 1 / (3.2 \times (|x - x_0| + |y - y_0|)))$
5. Initialize Angle = 0
6. While (Angle < θ)

   do

   {

   Plot (x, y)

   $x = x - (y - y_0) \times d\theta$

   $y = y + (x - x_0) \times d\theta$

   Angle = Angle + $d\theta$

   }
7. Stop.

Problems in True-Curve Generation Approach

1. To specify a curve, we need more information than just its endpoints.
2. It is difficult to apply transformations. For example, a circle when scaled in only one direction becomes an ellipse. If our algorithm supports only circular arc generation then ability to scale pictures is limited.
3. New clipping algorithm is required to clip arcs.
4. The curve generation algorithms for curves other than circular or elliptical such as airplane wings or cars or human faces, are complex.

9.2.2 Interpolation

In the last section we have seen limitations of true curve generation approach. Furthermore in practice we have to deal with some complex curves for which no direct mathematical function is available. Such curves can be drawn using approximation methods. If we have set of sample points which lie on the required curve, then we can draw the required curve by filling portions of the curve with the pieces of known curves which pass through nearby sample points. The gap between the sample points can be filled by finding the co-ordinates of the points along the known approximating curve and connecting these points with line segments as shown in the Fig. 9.2.

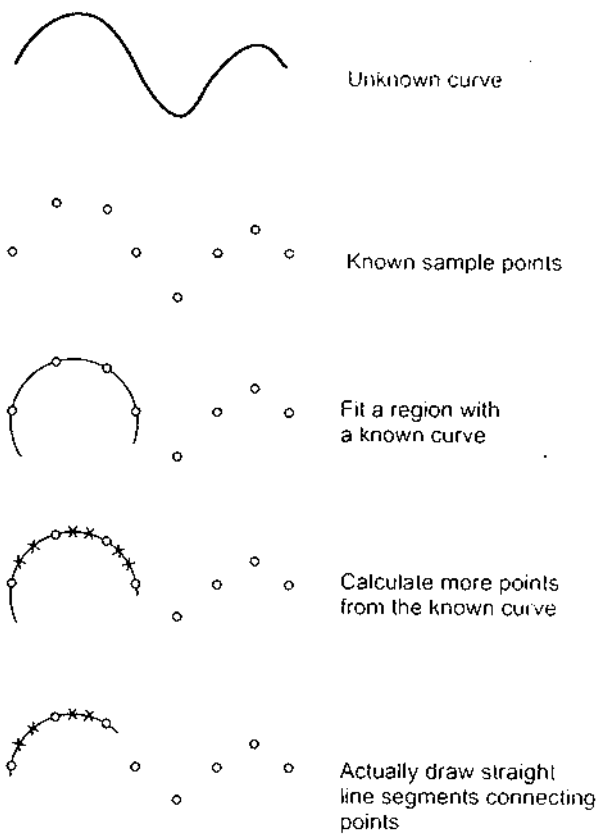


Fig. 9.2 The interpolation process

The main task in this process is to find the suitable mathematical expression for the known curve. There are polynomial, trigonometric, exponential and other classes of functions that can be used to approximate the curve. Usually polynomial functions in the

parametric form are preferred. The polynomial functions in the parametric form can be given as

$$x = f_x(u)$$
$$y = f_y(u)$$
$$z = f_z(u)$$

We can realise from above equations that the difference between 2 and 3 dimensions is just the addition of the third equation for z. Furthermore the parametric form treats all the three dimensions equally and allows multiple values (several values of y or z for a given x value). Due to these multiple values curves can double back or even cross themselves, as shown in Fig. 9.3.

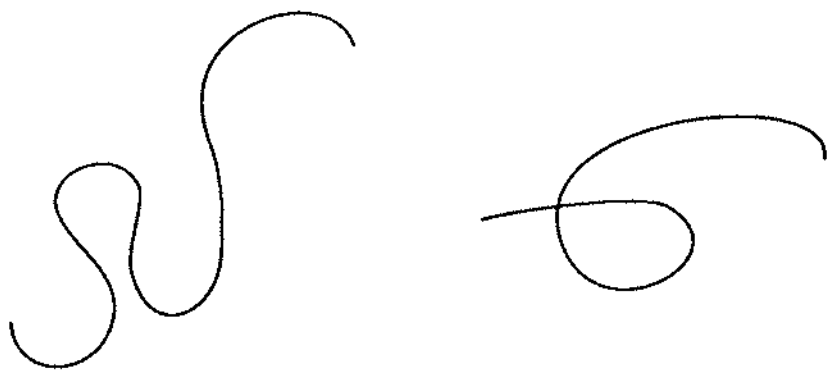


Fig. 9.3 Representation of curves with double back or crossing themselves

We have seen that, we have to draw the curve by determining the intermediate points between the known sample points. This can be achieved using interpolation techniques. Let's see the interpolation process.

Suppose we want a polynomial curve that will pass through n sample points.

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

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

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

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

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

The function B_i (u) is called '**blending function**'. For each value of parameter u, the blending function determines how much the ith sample point affects the position of the

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

$$\begin{aligned}
&\text{when} \quad u = -1 \Rightarrow B_1(u) = 1 \text{ and } 0 \text{ for } u = 0, 1, 2, \ldots, n-2 \\
&\text{when} \quad u = 0 \Rightarrow B_2(u) = 1 \text{ and } 0 \text{ for } u = -1, 1, \ldots, n-2 \\
&\vdots \\
&\text{when} \quad u = (n-2) \Rightarrow B_n(u) = 1 \text{ and } 0 \text{ for } u = -1, 0, \ldots, (n-1)
\end{aligned}$$

To get $B_1(u) = 1$ at $u = -1$ and 0 for $u = 0, 1, 2, \ldots, n-2$, the expression for $B_1(u)$ can be given as

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

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

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

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

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

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

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

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

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

$$\begin{aligned}
x &= x_1 B_1(u) + x_2 B_2(u) + x_3 B_3(u) + x_4 B_4(u) \\
y &= y_1 B_1(u) + y_2 B_2(u) + y_3 B_3(u) + y_4 B_4(u) \\
z &= z_1 B_1(u) + z_2 B_2(u) + z_3 B_3(u) + z_4 B_4(u)
\end{aligned}$$

It is possible to get intermediate points between two sampling points by taking values of u between the values of u related to the two sample points under consideration. For

example, we can find the intermediate points between second and third sample points for which values of u are 0 and 1, respectively; by taking values of u between 0 and 1. This is shown in Fig. 9.4.

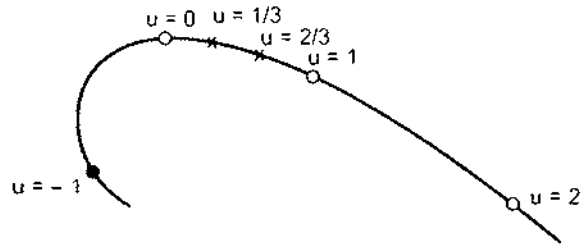


Fig. 9.4 Determining intermediate points for approximation of curve

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

Initially, sample points (1, 2, 3, 4) are considered and intermediate points between (2, 3) are obtained. Then sample point at one end is discarded and sample point at the other end is added to get new sample points (2, 3, 4, 5). Now the curve between sample points (3, 4) is approximated. The subsequent intermediate points can be obtained by repeating the same procedure. The initial and final portions of the curve require special treatment. For the first four points (1, 2, 3, 4) we have to draw region between points (1, 2) with u values between – 1 and 0.

Similarly the blending function for very last step of the curve should be evaluated with u values between 1 and 2.

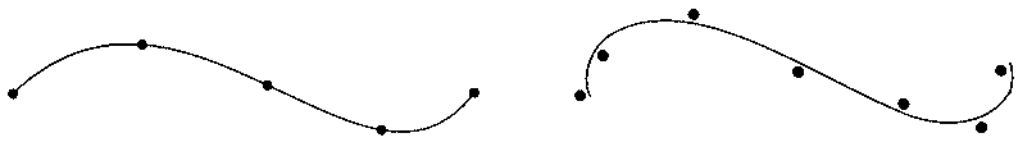
Interpolating Algorithm

1. Get the sample points.
2. Get intermediate values of u to determine intermediate points.
3. Calculate blending function values for middle section of the curve.
4. Calculate blending function values for first section of the curve.
5. Calculate blending function values for the last section of the curve.
6. Multiply the sample points by blending functions to give points on approximation curve.
7. Connect the neighbouring points using straight line segments
8. Stop.

9.3 Spline Representation

To produce a smooth curve through a designated set of points, a flexible strip called **spline** is used. Such a spline curve can be mathematically described with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various

curve sections. We can specify a spline curve by giving a set of coordinate positions, called **control points**, which indicates the general shape of the curve. When polynomial sections are fitted so that the curve passes through all control points, as shown in the Fig. 9.5 (a), the resulting curve is said to **interpolate** the set of control points. On the other hand, when the polynomials are fitted to the path which is not necessarily passing through all control points, the resulting curve is said to approximate the set of control points. This is illustrated in the Fig. 9.5 (b).



(a) Interpolation spline (b) Approximation spline

Fig. 9.5

9.3.1 Geometric and Parametric Continuity

To ensure a smooth transition from one section of a piecewise parametric curve to the next, we can impose various continuity conditions at the connection points. We see parametric continuity and geometric continuity conditions.

In geometric continuity we require parametric derivatives of two sections to be proportional to each other at their common boundary instead of equal to each other. Parametric continuity is set by matching the parametric derivatives of adjoining two curve sections at their common boundary. In zero order parametric continuity, given as C^0, it means simply the curve meet and same is for zero order geometric continuity. In first order parametric continuity called as C^1 means that first parametric derivatives of the coordinate functions for two successive curve sections are equal at the joining point and geometric first order continuity means the parametric first derivative are proportional at the intersection of two successive sections. Second order parametric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are same at the intersection and for second order geometric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under C^2 continuity curvature of the two curve sections match at the joining positions.

Two curves

$$r(t) = (t^2 - 2t, t)$$
$$n(t) = (t^2 + 1, t + 1)$$

C^1 and G^1 are continuous at $r(1) = n(0)$

Derivative

$$r(t) = 2t - 2, 1$$
$$r(1) = 2 - 2, 1$$
$$= 0, 1$$

Derivative $n(t) = 2t, 1$

$n(0) = 0, 1$

$\therefore$ $r(1) = n(0)$, two curves are continuous.

Ex. 9.1 : *Show that two curves $n(t) = (t^2 + 2t - 2, t^2)$ and $r(t) = (t^2 + 2t + 1, t + 1)$ are both C^0 and G^0 continuous where they join at $n(1) = r(0)$. Do they meet C^1 and G^1 continuity.*

Sol. :

$$n(t) = (t^2 + 2t - 2, t^2)$$

$$r(t) = (t^2 + 2t + 1, t + 1)$$

Zero order parametric continuity, described as C^0 continuity, means simply that the curves meet. That is, the values of x, y and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x, y and z evaluated at u_1 for the next curve section. The **zero-order geometric continuity** described as G^0 continuity, is the same as zero-order parametric continuity.

We have,

$$n(1) = (1^2 + 2 - 2, 1^2)$$

$$= (1, 1)$$

$$r(0) = (0^2 + 0 + 1, 0 + 1)$$

$$= (1, 1)$$

Therefore, we can say that both curves are C^0 and G^0 continuous at n(1) and r(0). To check for C^1 and G^1 continuity we have to take first derivative of both the curves :

$$\text{Derivative } n(t) = (2t + 2, 2t)$$

$$\text{Derivative } r(t) = (2t + 2, 1)$$

$$\therefore \quad n(1) = (2 + 2, 2)$$

$$= (4, 2)$$

$$r(0) = (2, 1)$$

Since $n(1) \neq r(0)$, the two curves are not C^1 and G^1 continuous at n(1) and r(0).

9.3.2 Spline Specifications

There are three basic ways of specifying spline curve :

- We can state the set of boundary conditions that are imposed on the spline
- We can state the matrix that characteristics the spline or
- We can state the set of blending functions that calculate the positions along the curve path by specifying combination of geometric constraints on the curve.

Why to use cubic polynomials ?

Polylines and polygons are first-degree, piecewise linear approximation to curves and surfaces, respectively. But this lower degree polynomials give too little flexibility in controlling the shape of the curve. The higher-degree polynomials give reasonable design flexibility, but introduce unwanted wiggles and also require more computation. For this reason the third-degree polynomials are most often used for representation of curves. These polynomials are commonly known as cubic polynomials.

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

$$x(u) = ax\,u^3 + bx\,u^2 + cx\,u + dx$$
$$y(u) = ay\,u^3 + by\,u^2 + cy\,u + dy$$
$$z(u) = az\,u^3 + bz\,u^2 + cz\,u + dz \qquad (0 \le u \le 1) \qquad \ldots (9.6)$$

For each of these three equations, we need to determine the values of the four coefficients a, b, c and d in the polynomial representation for each of the n curve sections between the n + 1 control points. We do this by setting enough boundary conditions at the joints between curve sections so that we can obtain numerical values for all the coefficient. Let us see the common methods for setting the boundary conditions for cubic interpolation splines.

9.4 Bezier Curves

Bezier curve is an another approach for the construction of the curve. A Bezier curve is determined by a defining polygon. Bezier curves have a number of properties that make them highly useful and convenient for curve and surface design. They are also easy to implement. Therefore Bezier curves are widely available in various CAD systems and in general graphic packages. In this section we will discuss the cubic Bezier curve. The reason for choosing cubic Bezier curve is that they provide reasonable design flexibility and also avoid the large number of calculations.

Properties of Bezier curve

1. The basis functions are real.
2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
4. The curve generally follows the shape of the defining polygon.
5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
6. The curve lies entirely within the convex hull formed by four control points.
7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more often than the defining polygon.
9. The curve is invariant under an affine transformation.

In cubic Bezier curve four control points are used to specify complete curve. Unlike the B-spline curve, we do not add intermediate points and smoothly extend Bezier curve, but we pick four more points and construct a second curve which can be attached to the first.

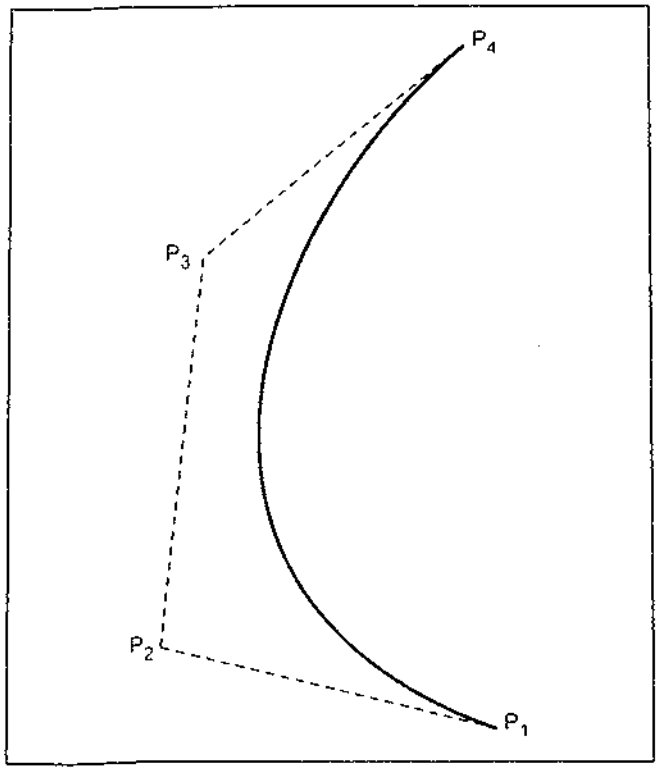


Fig. 9.6 A cubic Bezier spline

The second curve can be attached to the first curve smoothly by selecting appropriate control points.

Fig. 9.6 shows the Bezier curve and its four control points. As shown in the Fig. 9.6, Bezier curve begins at the first control point and ends at the fourth control point. This means that if we want to connect two Bezier curves, we have to make the first control point of the second Bezier curve match the last control point of the first curve. We can also observe that at the start of the curve, the curve is tangent to the line connecting first and second control points. Similarly at the end of curve, the curve is tangent to the line connecting the third and fourth control point. This means that, to join two Bezier curves smoothly we have to place the third and the fourth control points of the first curve on the same line specified by the first and the second control points of the second curve.

The Bezier matrix for periodic cubic polynomial is

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\therefore \quad P(u) = U \cdot M_B \cdot G_B$$

$$\text{where} \quad G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

and the product $P(u) = U.M_B.G_B$ is

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

Ex. 9.2 *Construct the Bezier curve of order 3 and with 4 polygon vertices A(1, 1), B(2, 3), C(4, 3) and D(6, 4).*

Sol. : The equation for the Bezier curve is given as

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

for $0 \le u \le 1$

where P(u) is the point on the curve P_1, P_2, P_3, P_4

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$

$$\therefore \quad P(0) = P_1 = (1, 1)$$

$$\therefore \quad P\left(\frac{1}{4}\right) = \left(1-\frac{1}{4}\right)^3 P_1 + 3\frac{1}{4}\left(1-\frac{1}{4}\right)^2 P_2 + 3\left(\frac{1}{4}\right)^2\left(1-\frac{1}{4}\right)P_3 + \left(\frac{1}{4}\right)^3 P_4$$

$$= \frac{27}{64}(1,1) + \frac{27}{64}(2,3) + \frac{9}{64}(4,3) + \frac{1}{64}(6,4)$$

$$= \left[\frac{27}{64}\times 1 + \frac{27}{64}\times 2 + \frac{9}{64}\times 4 + \frac{1}{64}\times 6,\ \frac{27}{64}\times 1 + \frac{27}{64}\times 3 + \frac{9}{64}\times 3 + \frac{1}{64}\times 4\right]$$

$$= \left[\frac{123}{64}, \frac{139}{64}\right]$$

$$= (1.9218, 2.1718)$$

$$\therefore \quad P\left(\frac{1}{2}\right) = \left(1-\frac{1}{2}\right)^3 P_1 + 3\frac{1}{2}\left(1-\frac{1}{2}\right)^2 P_2 + 3\left(\frac{1}{2}\right)^2\left(1-\frac{1}{2}\right)P_3 + \left(\frac{1}{2}\right)^3 P_4$$

$$= \frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,3) + \frac{1}{8}(6,4)$$

$$= \left[\frac{1}{8}\times 1 + \frac{3}{8}\times 2 + \frac{3}{8}\times 4 + \frac{1}{8}\times 6,\ \frac{1}{8}\times 1 + \frac{3}{8}\times 3 + \frac{3}{8}\times 3 + \frac{1}{8}\times 4\right]$$

$$= \left[\frac{25}{8}, \frac{23}{8}\right]$$

$$= (3.125,\ 2.875)$$

$$\therefore \quad P\left(\frac{3}{4}\right) = \left(1-\frac{3}{4}\right)^3 P_1 + 3\frac{3}{4}\left(1-\frac{3}{4}\right)^2 P_2 + 3\left(\frac{3}{4}\right)^2\left(1-\frac{3}{4}\right)P_3 + \left(\frac{3}{4}\right)^3 P_4$$

$$= \frac{1}{64}P_1 + \frac{9}{64}P_2 + \frac{27}{64}P_3 + \frac{27}{64}P_4$$

$$= \frac{1}{64}(1,1) + \frac{9}{64}(2,3) + \frac{27}{64}(4,3) + \frac{27}{64}(6,4)$$

$$= \left[\frac{1}{64}\times 1 + \frac{9}{64}\times 2 + \frac{27}{64}\times 4 + \frac{27}{64}\times 6,\ \frac{1}{64}\times 1 + \frac{9}{64}\times 3 + \frac{27}{64}\times 3 + \frac{27}{64}\times 4\right]$$

$$= \left[\frac{289}{64}, \frac{217}{64}\right]$$

$$= (4.5156,\ 3.375)$$

$$P(1) = P_3 = (6, 4)$$

The Fig. 9.7 shows the calculated points of the Bezier curve and curve passing through it

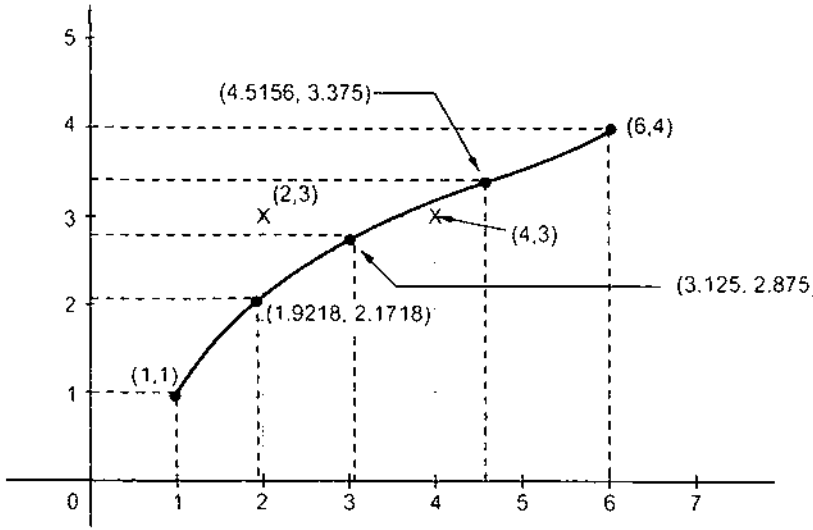


Fig. 9.7 Plotted Bezier curved

Another approach to construct the Bezier curve is called midpoint approach. In this approach the Bezier curve can be constructed simply by taking midpoints. In midpoint approach midpoints of the lines connecting four control points (A, B, C, D) are determined (AB, BC, CD). These midpoints are connected by line segments and their midpoints ABC and BCD are determined. Finally these two midpoints are connected by line segments and its midpoint ABCD is determined. This is illustrated in Fig. 9.8.

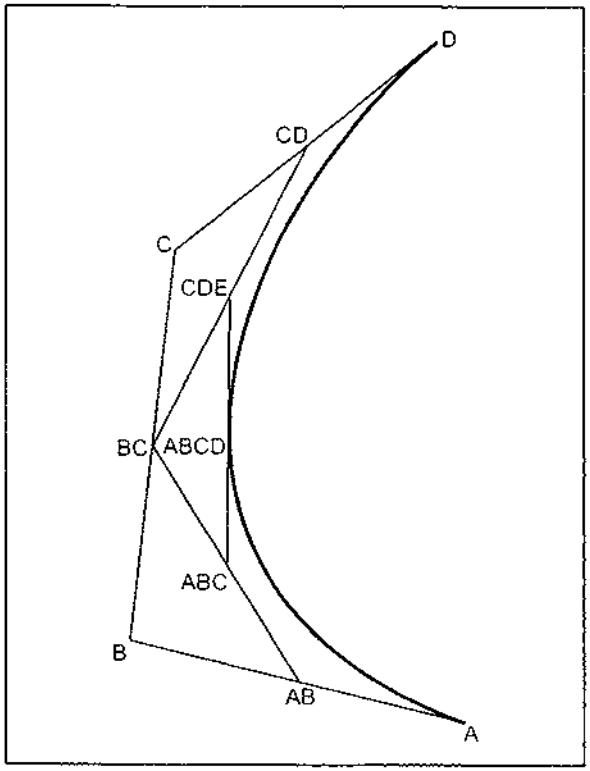


Fig. 9.8 Subdivision of a Bezier spline

The point ABCD on the Bezier curve divides the original curve into two sections. This makes the points A, AB, ABC and ABCD are the control points for the first section and the points ABCD, BCD, CD and D are the control points for the second section. By considering two sections separately we can get two more sections for each separate section i.e. the original Bezier curve gets divided into four different curves. This process can be repeated to split the curve into smaller sections until we have sections so short that they can be replaced by straight lines or even until the sections are not bigger than individual pixels.

Algorithm

1. Get four control points say A (x_A, y_A), B (x_B, y_B), C (x_C, y_C), D (x_D, y_D)
2. Divide the curve represented by points A, B, C and D in two sections

   $x_{AB} = (x_A + x_B) / 2$

   $y_{AB} = (y_A + y_B) / 2$

   $x_{BC} = (x_B + x_C) / 2$

   $y_{BC} = (y_B + y_C) / 2$

   $x_{CD} = (x_C + x_D) / 2$

   $y_{CD} = (y_C + y_D) / 2$

   $x_{ABC} = (x_{AB} + x_{BC}) / 2$

   $y_{ABC} = (y_{AB} + y_{BC}) / 2$

   $x_{BCD} = (x_{BC} + x_{CD}) / 2$

   $y_{BCD} = (y_{BC} + y_{CD}) / 2$

   $x_{ABCD} = (x_{ABC} + x_{BCD}) / 2$

   $y_{ABCD} = (y_{ABC} + y_{BCD}) / 2$
3. Repeat the step 2 for section A, AB, ABC and ABCD and section ABCD, BCD, CD and D
4. Repeat step 3 until we have sections so short that they can be replaced by straight lines.
5. Replace small sections by straight lines.
6. Stop

'C' code for Drawing Bezier Curve

(Softcopy of this program is available at vtubooks.com)

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
int gd,gm,maxx,maxy;
float xxx[4][2];

/* Function to draw line from relative position
   specified in array xxx-----------------------*/

void line1(float x2,float y2)
{
line(xxx[0][0],xxx[0][1],x2,y2);
xxx[0][0]=x2;
xxx[0][1]=y2;
}
```

```
/* Bezier function
-------------------- */
bezier(float xb,float yb,float xc,float yc,float xd,float yd,int n)
{
float xab,yab,xbc,ybc,xcd,ycd;
float xabc,yabc,xbcd,ybcd;
float xabcd,yabcd;
if (n==0)
        {
         line1(xb,yb);
         line1(xc,yc);
         line1(xd,yd);
        }
else
        {
         xab = (xxx[0][0]+xb)/2;
         yab = (xxx[0][1]+yb)/2;
         xbc = (xb+xc)/2;
         ybc = (yb+yc)/2;
         xcd = (xc+xd)/2;
         ycd = (yc+yd)/2;
         xabc = (xab+xbc)/2;
         yabc = (yab+ybc)/2;
         xbcd = (xbc+xcd)/2;
         ybcd = (ybc+ycd)/2;
         xabcd = (xabc+xbcd)/2;
         yabcd = (yabc+ybcd)/2;
         n=n-1;
         bezier(xab,yab,xabc,yabc,xabcd,yabcd,n);
         bezier(xbcd,ybcd,xcd,ycd,xd,yd,n);
        }
}

/* Function to initialise graphics
---------------------------------- */
void igraph()
        {
        detectgraph(&gd,&gm);
```

```
        if(gd<0)
                {
                puts("CANNOT DETECT A GRAPHICS CARD");
                exit(1);
                }
        initgraph(&gd,&gm,"f:\\tc");
        }
main()
        {
        int i;
        float temp1,temp2;
        igraph();

        /* Read two end points and two control points of the curve
        ------------------------------------------------------------ */
        for(i=0;i<4;i++)
        {
        printf("Enter (x,y) coordinates of point%d : ",i+1);
        scanf("%f,%f",&temp1,&temp2);
        xxx[i][0] = temp1;
        xxx[i][1] = temp2;
        }
        bezier(xxx[1][0],xxx[1][1],xxx[2][0],xxx[2][1],xxx[3][0],xxx
        [3][1],8);
        getch();
        closegraph();
}
```

9.5 B-Spline Curves

We have seen that, a curve generated by using the vertices of a defining polygon is dependent on some interpolation or approximation scheme to establish the relationship between the curve and the polygon. This scheme is provided by the choice of basis function. The Bezier curve produced by the Bernstein basis function has a limited flexibility. First the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve. For example, polygon with four vertices results a cubic polynomial curve. The only way to reduce the degree of the curve is to reduce the number of vertices, and conversely the only way to increase the degree of the curve is to increase the number of vertices. The second limiting characteristics is that the value of the blending function is nonzero for all parameter values over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change with in a curve.

There is another basis function, called the B-spline basis, which contains the Bernstein basis as a special case. The B-spline basis is nonglobal. It is nonglobal because each vertex B_i is associated with a unique basis function. Thus, each vertex affects the shape of the curve only over a range of parameter values where its associated basis function is nonzero. The B-spline basis also allows the order of the basis function and hence the degree of the resulting curve is independent on the number of vertices. It is possible to change the degree of the resulting curve without changing the number of vertices of the defining polygon.

If P(u) be the position vectors along the curve as a function of the parameter u, a B-spline curve is given by

$$P(u) = \sum_{i=1}^{n+1} B_i N_{i,k}(u) \quad u_{min} \le u < u_{max}, \quad 2 \le k \le n+1 \qquad ...(9.7)$$

where the B_i are the position vectors of the n + 1 defining polygon vertices and the $N_{i,k}$ are the normalized B-spline basis functions. For the ith normalized B-spline basis function of order k, the basis function $N_{i,k}(u)$ are defined as

$$N_{i,1}(u) = \begin{cases} 1 & \text{if} \quad x_i \le u < x_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

and

$$N_{i,k}(u) = \frac{(u - x_i) N_{i,k-1}(u)}{x_{i+k-1} - x_i} + \frac{(x_{i+k} - u) N_{i+1,k-1}(u)}{x_{i+k} - x_{i+1}} \quad ...(9.8)$$

The values of x_i are the elements of a knot vector satisfying the relation $x_i \le x_{i+1}$. The parameter u varies from u_{min} to u_{max} along the curve P(u). The choice of knot vector has a significant influence on the B-spline basis functions $N_{i,k}(u)$ and hence on the resulting B-spline curve. There are three types of knot vector : uniform, open uniform and nonuniform.

In a uniform knot vector, individual knot values are evenly spaced. For example,

[0 1 2 3 4]

For a given order k, uniform knot vectors give periodic uniform basis functions for which

$$N_{i,k}(u) = N_{i-1,k}(u-1) = N_{i+1,k}(u+1)$$

An open uniform knot vector has multiplicity of knot values at the ends equal to the order k of the B-spline basis function. Internal knot values are evenly spaced. Examples are,

$$k = 2 \; [0\,0\,1\,2\,3\,3]$$
$$k = 3 \; [0\,0\,0\,1\,2\,3\,3\,3]$$
$$k = 4 \; [0\,0\,0\,0\,1\,2\,2\,2\,2]$$

Generally, an open uniform knot vector is given by,

$$x_i = 0 \qquad 1 \le i \le k$$
$$x_i = i - k \qquad k+1 \le i \le n+1$$
$$x_i = n - k + 2 \qquad n+2 \le i \le n+k+1 \qquad ...(9.9)$$

The curves resulted by the use of open uniform basis function are nearly like Bezier curves. In fact, when the number of defining polygon vertices is equal to the order of the B-spline basis and an open uniform knot vector is used, the B-spline basis reduces to the Bernstein basis. Hence, the resulting B-spline curve is a Bezier curve.

Ex. 9.3 *Calculate the four third-order basis function $N_{i,3}(u)$, $i = 1, 2, 3, 4$ with an open uniform knot vector.*

Sol. : We have to calculate four basis functions, therefore n = (4 – 1) = 3 and it is of order three, therefore k = 3. From equation 9.9 the open uniform knot vector is given as

$$[X] = [0\,0\,0\,1\,2\,2\,2]$$

Now, from equation 9.8, the basis functions for various parameters are as follows :

$0 \le u < 1$

$N_{3,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \ne 3$

$N_{2,2}(u) = 1 - u;$ $\quad N_{3,2}(u) = u, \quad N_{i,2}(u) = 0 \quad i \ne 2, 3$

$N_{1,3}(u) = (1-u)^2;$ $\quad N_{2,3}(u) = u(1-u) + \dfrac{(2-u)}{2}u$

$N_{3,3}(u) = \dfrac{u^2}{2};$ $\quad N_{i,3}(u) = 0; \quad i \ne 1, 2, 3$

$1 \le u < 2$

$N_{4,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \ne 4$

$N_{3,2}(u) = (2-4);$ $\quad N_{4,2}(u) = (u-1); \quad N_{i,2}(u) = 0, i \ne 3, 4$

$N_{2,3}(u) = \dfrac{(2-u)^2}{2};$ $\quad N_{3,3}(t) = \dfrac{u(2-u)}{2} + (2-u)(u-1);$

$N_{4,3}(u) = (u-1)^2;$ $\quad N_{i,3}(u) = 0; \quad i \ne 2, 3, 4$

Ex. 9.4 *Construct the B-spline curve of order 4 and with 4 polygon vertices A(1, 1), B(2, 3) C(4, 3) and D(6, 2).*

Sol. : Here n = 3 and k = 4 from equation 9.9 we have open uniform knot vector as

x = [0 0 0 0 1 1 1 1] and from equation 9.8 we have basis functions are

$0 \le u < 1$

$N_{4,1}(u) = 1;$ $\quad N_{i,1}(u) = 0, \quad i \ne 4$

$N_{3,2}(u) = (1-u);$ $\quad N_{4,2}(u) = u, \quad N_{i,2}(u) = 0, \quad i \ne 3, 4$

$N_{2,3}(u) = (1-u)^2;$ $\quad N_{3,3}(u) = 24(1-u);$

$N_{4,3}(u) = u^2;$ $\quad N_{i,3}(u) = 0; \quad i \ne 2, 3, 4$

$N_{1,4}(u) = (1-t)^3;$ $\quad N_{2,4}(u) = u(1-u)^2 + 2u(1-u)^2 = 3u(1-u)^2;$

$N_{3,4}(u) = 2u^2(1-u) + (1-u)u^2 = 3u^2(1-u);\ N_{4,4}(u) = u^3$

Using equation 1 the parametric B-spline is

$$P(u) = AN_{1,4}(u) + BN_{2,4}(u) + CN_{3,4}(u) + DN_{4,4}(u)$$

$$\therefore\ P(u) = (1-u)^3 A + 3u(1-u)^2 B + 3u^2(1-u)C + u^3 D$$

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$

$$\therefore P(0) = A = [1, 1]$$

$$\therefore\ P\left(\frac{1}{4}\right) = \left(1-\frac{1}{4}\right)^3 A + 3\frac{1}{4}\left(1-\frac{1}{4}\right)^2 B + 3\left(\frac{1}{4}\right)^2\left(1-\frac{1}{4}\right)C + \left(\frac{1}{4}\right)^3 D$$

$$= \left(\frac{27}{64}\right)A + \left(\frac{27}{64}\right)B + \left(\frac{9}{64}\right)C + \left(\frac{1}{64}\right)D$$

$$= \left[\frac{27}{64}(1,1) + \frac{27}{64}(2,3) + \frac{9}{64}(4,3) + \frac{1}{64}(6,2)\right]$$

$$= \left[\frac{27}{64}\times 1 + \frac{27}{64}\times 2 + \frac{9}{64}\times 4 + \frac{1}{64}\times 6,\ \frac{27}{64}\times 1 + \frac{27}{64}\times 3 + \frac{9}{64}\times 3 + \frac{1}{64}\times 2\right]$$

$$= \left[\frac{123}{64}, \frac{137}{64}\right]$$

$$= [1.9218,\ 2.14]$$

$$\therefore\ P\left(\frac{1}{2}\right) = \left(1-\frac{1}{2}\right)^3 A + 3\frac{1}{2}\left(1-\frac{1}{2}\right)^2 B + 3\left(\frac{1}{2}\right)^2\left(1-\frac{1}{2}\right)C + \left(\frac{1}{2}\right)^3 D$$

$$= \frac{1}{8}A + \frac{3}{8}B + \frac{3}{8}C + \frac{1}{8}D$$

$$= \left[\frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,3) + \frac{1}{8}(6,2)\right]$$

$$= \left[\frac{1}{8}\times 1 + \frac{3}{8}\times 2 + \frac{3}{8}\times 4 + \frac{1}{8}\times 6,\ \frac{1}{8}\times 1 + \frac{3}{8}\times 3 + \frac{3}{8}\times 3 + \frac{1}{8}\times 2\right]$$

$$= \left[\frac{25}{8}, \frac{21}{8}\right]$$

$$= [3.125,\ 2.625]$$

$$\therefore\ P\left(\frac{3}{4}\right) = \left(1-\frac{3}{4}\right)^3 A + 3\frac{3}{4}\left(1-\frac{3}{4}\right)^2 B + 3\left(\frac{3}{4}\right)^2\left(1-\frac{3}{4}\right)C + \left(\frac{3}{4}\right)^3 D$$

$$= \frac{1}{64}A + \frac{9}{64}B + \frac{27}{64}C + \frac{27}{64}D$$

$$= \frac{1}{64}(1,1) + \frac{9}{64}(2,3) + \frac{27}{64}(4,3) + \frac{27}{64}(6,2)$$

$$= \left[\frac{1}{64}\times 1 + \frac{9}{64}\times 2 + \frac{27}{64}\times 4 + \frac{27}{64}\times 6,\ \frac{1}{64}\times 1 + \frac{9}{64}\times 3 + \frac{27}{64}\times 3 + \frac{27}{64}\times 2\right]$$

$$= \left[\frac{289}{64}, \frac{163}{64}\right]$$

$= [4.5156, 2.5468]$

$\therefore \; P(1) = D = [6, 2]$

The Fig. 9.9 shows the calculated points of the B-spline curve and curve passing through it.

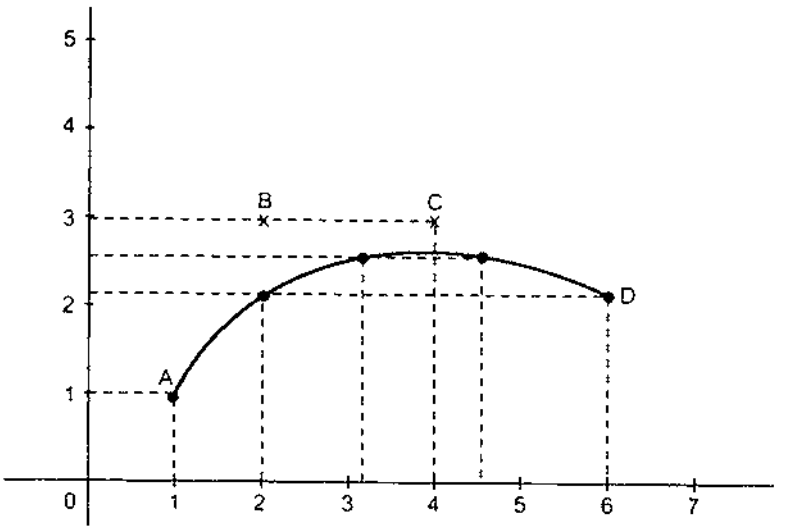


Fig. 9.9 Plotted B-spline curve

Properties of B-spline curve

- The sum of the B-spline basis functions for any parameter value u is 1.

  i.e. $\sum_{i=1}^{n+1} N_{i,k}(u) \equiv 1$

- Each basis function is positive or zero for all parameter values, i.e., $N_{i,k} \geq 0$.
- Except for k = 1 each basis function has precisely one maximum value.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon (with certain limitations).
- B-spline allows local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property. Thus the curve does not oscillate about any straight line move often than its defining polygon.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

9.6 Parametric Bicubic Surfaces

Parametric bicubic surfaces are a generalization of parametric cubic curves. In section 9.3 we have seen the general form of parametric cubic curve

$$P(u) = U \cdot M \cdot G$$

If we now allow the points in G to vary in 3D along some path that is parameterized on t, we have

$$P(u, t) = U \cdot M \cdot G(t) = U \cdot M \cdot \begin{bmatrix} G_1(t) \\ G_2(t) \\ G_3(t) \\ G_4(t) \end{bmatrix} \quad \ldots(9.10)$$

Now, different values of t between 0 to 1 we get different curves. For slight different values of t we get slightly different curves. The set of all such curves arbitrarily close to each other for values of t between 0 and 1 defines a surface. If the $G_i(t)$ are themselves cubics, the surface is said to be a parametric bicubic surface, and $G_i(t)$ can be represented as

$$G_i(t) = U \cdot M \cdot G_i \quad \ldots (9.11)$$

where $G_i = [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}]^T$ and

g_{i1} is the first element of the geometry vector for curve $G_i(t)$.

The transpose of equation (9.11) can be given as

$$G_i(t)^T = G_i^T \cdot M^T \cdot U^T \quad \because (A \cdot B \cdot C)^T = C^T \cdot B^T \cdot A^T$$

Substituting the above result in equation (9.10)

We have

$$\begin{aligned} P(u, t) &= U \cdot M \cdot G_i^T \cdot M^T \cdot U^T \\ &= U \cdot M \cdot [g_{i1}\ g_{i2}\ g_{i3}\ g_{i4}] \cdot M^T \cdot U^T \\ &= U \cdot M \cdot \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M^T \cdot U^T \quad \ldots (9.12) \\ &= U \cdot M \cdot G \cdot M^T \cdot U^T \quad \text{where } 0 \le u, t \le 1 \quad \ldots (9.13) \end{aligned}$$

In terms of x, y, z separately the above equation can be written as

$$\begin{aligned} x(u, t) &= U \cdot M \cdot G_x \cdot M^T \cdot U^T \\ y(u, t) &= U \cdot M \cdot G_y \cdot M^T \cdot U^T \\ z(u, t) &= U \cdot M \cdot G_z \cdot M^T \cdot U^T \quad \ldots (9.14) \end{aligned}$$

9.6.1 Hermite Surfaces

The parametric bicubic equation for Hermite surface can be given as

$$P(u, t) = U \cdot M_H \cdot G_H(t) = U \cdot M_H \cdot \begin{bmatrix} P_1(t) \\ P_4(t) \\ DP_1(t) \\ DP_4(t) \end{bmatrix} \quad \ldots (9.15)$$

where

$$P_1(t) = U \cdot M_H \cdot \begin{bmatrix} g_{11} \\ g_{12} \\ g_{13} \\ g_{14} \end{bmatrix}$$

$$P_4(t) = U \cdot M_H \cdot \begin{bmatrix} g_{21} \\ g_{22} \\ g_{23} \\ g_{24} \end{bmatrix}$$

$$DP_1(t) = U \cdot M_H \cdot \begin{bmatrix} g_{31} \\ g_{32} \\ g_{33} \\ g_{34} \end{bmatrix}$$

$$DP_4(t) = U \cdot M_H \cdot \begin{bmatrix} g_{41} \\ g_{42} \\ g_{43} \\ g_{44} \end{bmatrix}$$

The above four equations can be rewritten together as

$$[\, P_1(t) \;\; P_4(t) \;\; DP_1(t) \;\; DP_4(t)] = U \cdot M_H \cdot G_H^T \quad \ldots (9.16)$$

where,

$$G_H = \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix}$$

Taking transpose of both sides we have

$$\begin{bmatrix} P_1(t) \\ P_4(t) \\ DP_1(t) \\ DP_4(t) \end{bmatrix} \begin{bmatrix} g_{11} & g_{12} & g_{13} & g_{14} \\ g_{21} & g_{22} & g_{23} & g_{24} \\ g_{31} & g_{32} & g_{33} & g_{34} \\ g_{41} & g_{42} & g_{43} & g_{44} \end{bmatrix} \cdot M_H^T \cdot U^T = G_H \cdot M_H^T \cdot U^T \quad \ldots (9.17)$$

Substituting the equation (9.17) in equation (9.15) we have,

$$P(u, t) = U \cdot M_H \cdot G_H \cdot M_H^T \cdot U^T \quad \ldots (9.18)$$

In terms of x, y, z separately the above equation can be written as

$$x(u, t) = U \cdot M_H \cdot G_{Hx} \cdot M_H^T \cdot U^T$$
$$y(u, t) = U \cdot M_H \cdot G_{Hy} \cdot M_H^T \cdot U^T$$
$$z(u, t) = U \cdot M_H \cdot G_{Hz} \cdot M_H^T \cdot U^T \quad \ldots (9.19)$$

9.6.2 B-Spline Surfaces

Applying similar procedure as that of Hermite surface we can represent B-spline surface as

$$x(u, t) = U \cdot M_{BS} \cdot G_{BSx} \cdot M_{BS}^T \cdot U^T$$
$$y(u, t) = U \cdot M_{BS} \cdot G_{BSy} \cdot M_{BS}^T \cdot U^T$$
$$z(u, t) = U \cdot M_{BS} \cdot G_{BSz} \cdot M_{BS}^T \cdot U^T \quad \ldots (9.20)$$

9.6.3 Bezier Surface

Applying similar procedure as that of Hermite surface we can represent Bezier surface as

$$x(u, t) = U \cdot M_B \cdot G_{Bx} \cdot M_B^T \cdot U^T$$
$$y(u, t) = U \cdot M_B \cdot G_{By} \cdot M_B^T \cdot U^T$$
$$z(u, t) = U \cdot M_B \cdot G_{Bz} \cdot M_B^T \cdot U^T \quad \ldots (9.21)$$

Review Questions

1. Explain the true curve generation algorithm.
2. List the problems in true curve generation algorithm.
3. What is interpolation ? Explain Lagrangian interpolation method.
4. What is spline ?
5. Differentiate between interpolation spline and approximation spline.
6. Give the various methods for specifying spline curve.
7. Why to use cubic polynominals ?
8. Write a short note on B-spline curve.
9. List the properties of B-spline curve.
10. Write a short note on Bezier curve.
11. Explain the properties of Bezier curve.

University Questions

1. Write detailed note on cubic B-splines **(Dec-96, May-97, May-2001)**
2. What do you understand by cubic B-splines? Discuss with suitable mathematical models. **(Dec-97)**

3. Temperature recorded at hourly intervals over a 24 hour period are stored in an array "temp". Write a program to display this as a "smooth" curve. (Dec-97)

4. What are the properties associated with curves ? Explain significance of each of them. (May-98)

5. Give the mathematical representation for Bezier curve ? Specify highlights and drawbacks of Bezier curve. If the Bezier curve is to be generated for P_1 (0, 0), P_2 (1, 3), P_3 (4, 2) and P_4 (2, 1) using six intervals of parameter u, find out the co-ordinates positions for every value of u. (May-98)

6. Give mathematical representation for B-spline curves. What are the properties of B-spline curves ? (Dec-98)

7. How the description for curved surface is obtained from the equation of Bezier curve ? State the properties of Bezier curves. (Dec-98)

8. Write a short note on B-spline curves (May-99)

9. Show that two curves $\gamma(t) = (t^2 - 2t, t)$ and $\eta(t) = (t^2 + 1, t + 1)$ are both C^1 and G^1 continuous where they join at $\gamma(1) = \eta(0)$. Note that C^n represents parametric continuity and G^n geometric continuity. (Dec-99)

10. What are the properties associated with curves ? Explain a mathematical representation for B-spline curves. (May-2000, Dec-2000)

11. Derive a mathematical representation for Bezier curves and state their properties. (May-2000, May-2002)

12. Explain how a curved surface can be obtained from the definition of a Bezier curve. (May-2000, May-2002)

13. Explain parametric continuity conditions and geometric continuity conditions. Show that two curves $n(t) = (t^2 + 2t - 2, t^2)$ and $r(t) = (t^2 + 2t + 1, t + 1)$ are both C^0 and G^0 continuous where they join at $n(1) = r(0)$. Do they meet C^1 and G^1 continuity ? (May-2001)

14. Write a program to display a two dimensional Bezier curve given a set of four control points in the xy plane. (Dec-2001)

15. Give important properties for designing curves and illustrate them. (May-2003)

16. Write a short note on Bezier curve. (May-2003)

❑❑❑

10 Light Shading

10.1 Introduction

So far we have seen how to construct three-dimensional objects, parallel and perspective projections of the objects, and removal of hidden surfaces and lines. In this chapter, we will see the shading of the three-dimensional objects and its model. The shading model is also called **illumination model** or **lighting model**. This model is used to calculate the intensity of light that we should see at a given point on the surface of an object.

Later part of this chapter gives the information about the colour models

10.2 Diffuse Illumination

An objects illumination is as important as its surface properties in computing its intensity. The object may be illuminated by light which does not come from any particular source but which comes from all directions. When such illumination is uniform from all directions, the illumination is called **diffuse illumination**. Usually, diffuse illumination is a background light which is reflected from walls, floor, and ceiling.

When we assume that going up, down, right and left is of same amount then we can say that the reflections are constant over each surface of the object and they are independent of the viewing direction. Such a reflection is called **diffuse reflection**. In practice, when object is illuminated, some part of light energy is absorbed by the surface of the object, while the rest is reflected. The ratio of the light reflected from the surface to the total incoming light to the surface is called **coefficient of reflection** or the **reflectivity**. It is denoted by R. The value of R varies from 0 to 1. It is closer to 1 for white surface and closer to 0 for black surface. This is because white surface reflects nearly all incident light whereas black surface absorbs most of the incident light. Reflection coefficient for gray shades is in between 0 to 1. In case of colour object reflection coefficient are various for different colour surfaces.

Lambert's Law

We have seen that, the diffuse reflections from the surface are scattered with equal intensity in all directions, independent of the viewing direction. Such surfaces are sometimes referred to as ideal diffuse reflectors. They are also called Lambertian reflector, since radiated light energy from any point on the surface is governed by **Lambert's cosine law**. This law states that the reflection of light from a perfectly diffusing surface varies as the

cosine of the angle between the normal to the surface and the direction of the reflected ray. This is illustrated in Fig. 10.1.

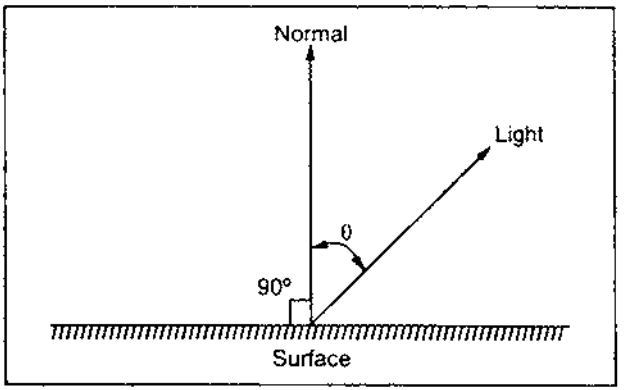


Fig. 10.1 The direction of light is measured from the surface normal

Thus if the incident light from the source is perpendicular to the surface at a perpendicular point, that point is fully illuminated. On the otherhand, as the angle of illumination moves away from the surface normal, the brightness of the point drops off; but the points appear to be squeezed closer together, and the net effect is that the brightness of the surface is unchanged. This is illustrated in Fig. 10.2. In other words we can say that the reduction in brightness due to cosine of angle gets cancelled by increase in the number of light-emitting points within the area of view.

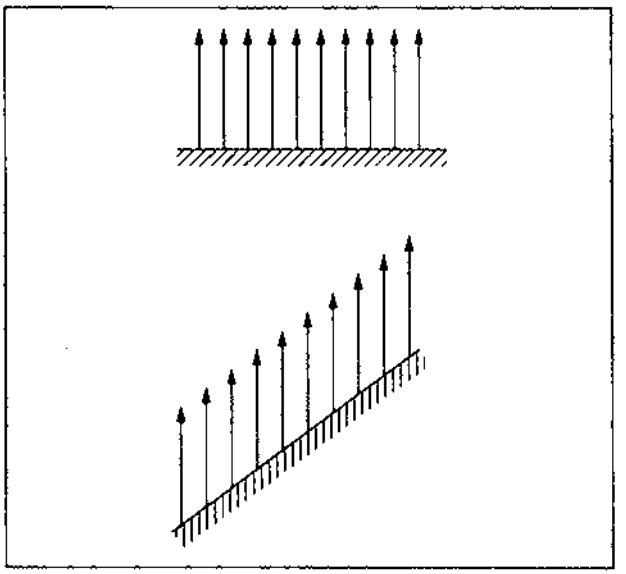


Fig. 10.2 Surface brightness

A similar cancellation effect can be observed as the surface is moved farther from the view point. As we move farther from the view port, the light coming from the surface spreads over a large area. This area increases by the square of distance, thus the amount of light reaching the eye decreases by the same factor. This factor is compensated by the size of the object. When object is moved farther from the viewport, it appears smaller. Therefore, eventhough there is less light, it is applied to a smaller area on the retina and hence the brightness of the surface remains unchanged.

The expression for the brightness of an object illuminated by diffuse ambient or background light can be given as

$$I_{ambdiff} = k_a I_a,$$

where I_a is the intensity of the ambient light or background light, k_a is the ambient reflection coefficient and $I_{ambdiff}$ is the intensity of diffuse reflection at any point on the surface which is exposed only to ambient light. Using above equation it is possible, to create light or dark scenes or gray shaded objects. But in this simple model, every plane on a particular object will be shaded equally. The real shaded object does not look like this. For more realistic shading model we also have to consider the point sources of illumination.

10.3 Point-Source Illumination

Point sources emits rays from a single point and they can approximate real world sources such as a small incandescent bulbs or candles. A point source is a direction source, whose all the rays come from the same direction, therefore, it can be used to represent the distant sun by approximating it as an infinitely distant point source.

The modelling of point sources requires additional work because their effect depends on the surface's orientation. If the surface is normal (perpendicular) to the incident light rays, it is brightly illuminated. The surfaces turned away from the light source (oblique surfaces) are less brightly illuminated. This is illustrated in Fig. 10.3.

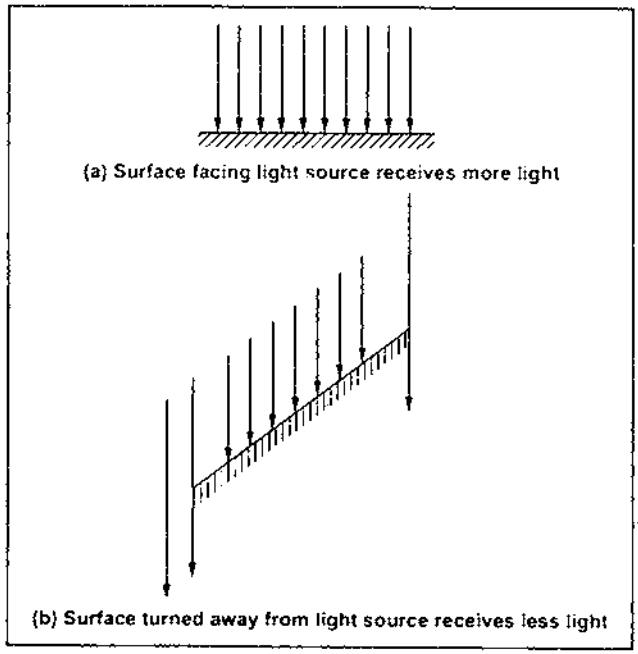


Fig. 10.3

For oblique surfaces, the illumination decreases by a factor of cos I, where I is the angle between the direction of the light and the direction normal to the surface plane. The angle I is know as angle of incidence. (See Fig. 10.4)

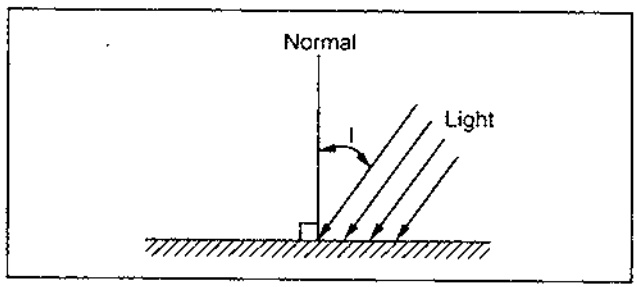


Fig. 10.4 The angle of incidence

The factor cos I is given as

$$\cos I = N \cdot L$$

where L is the vector of length 1 units pointing towards the light source and N is the vector of length 1 in the direction normal to the surface plane.

Considering both diffuse illumination and point source illumination, the shade of the visible surface of an object is given as

$$I_{ldiff} = k_a I_a + k_d I_l (\cos I)$$
$$= k_a I_a + k_d I_l (N \cdot L)$$

where $k_a I_a$ is the intensity of light coming from visible surface due to diffuse illumination,

I_l is the intensity of light comes from the point source, k_d is the diffuse reflectivity coefficient and vector dot product (L.N) gives the cosine of the angle of incidence.

10.4 Specular Reflection

When we illuminate a shiny surface such as polished metal or an apple with a bright light, we observe highlight or bright spot on the shiny surface. This phenomenon of reflection of incident light in a concentrated region around the specular reflection angle is called **specular reflection**. Due to specular reflection, at the highlight, the surface appears to be not in its original colour, but white, the colour of incident light.

The Fig. 10.5 shows the specular reflection direction at a point on the illuminated surface. The specular reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N. As shown in the Fig. 10.5, R is the unit vector in the direction of ideal specular reflection, L is the unit vector directed toward the point light source and V is the unit vector pointing to the viewer from the surface position.

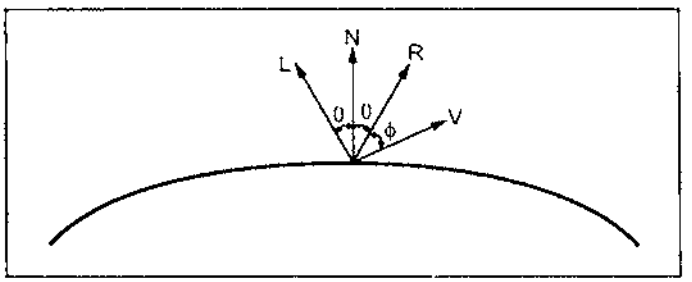


Fig. 10.5 Specular reflection

The angle ϕ between vector R and vector V is called **viewing angle**. For an ideal reflector (perfect mirror), incident light is reflected only in the specular reflection direction. In such case, we can see reflected light only when vector V and R coincide, i.e., $\phi = 0$.

10.4.1 The Phong Illumination Model

Phong Bui-Tuong developed a popular illumination model for nonperfect reflectors. It assumes that maximum specular reflection occurs when ϕ is zero and falls off sharply as ϕ increases. This rapid fall-off is approximated by $\cos^n \phi$, where n is the specular reflection parameter determined by the type of surface. The values of n typically vary from 1 to several hundred, depending on the surface material. The larger values (say, 100 or more) of n are used for very shiny surface and smaller values are used for dull surfaces. For a perfect reflector, n is infinite. For rough surface, such as chalk, n would be near to 1. Fig. 10.6 and Fig 10.7 show the effect of n on the angular range of specular reflection.

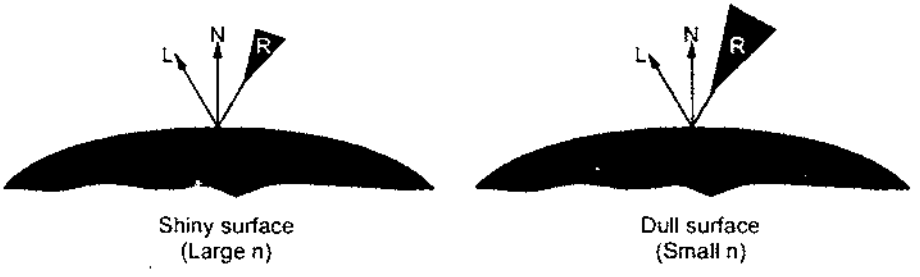


Fig. 10.6 Effect of n on the angular range of specular reflection

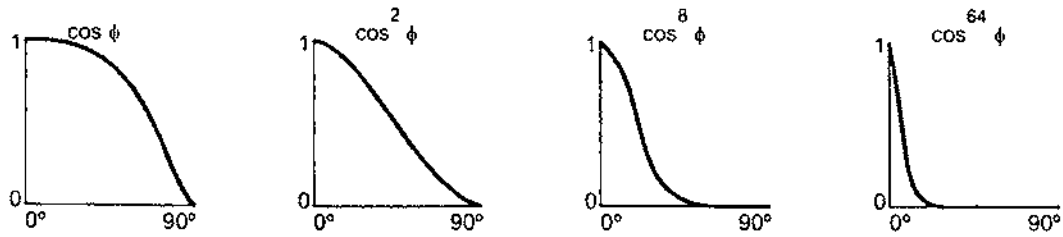


Fig. 10.7 Different values of $\cos^n \phi$ used in the Phong illumination model

The amount of incident light specularly reflected depends on the angle of incidence θ, material properties of surface, polarization and colour of the incident light. The model is approximated for monochromatic specular intensity variations using a specular-reflection coefficient, $W(\theta)$, for each surface. We can write the equation for Phong specular reflection model as

$$I_{spec} = W(\theta)\, I_l \cos^n\phi$$

where I_l is the intensity of the light source and ϕ is the angle between viewing vector and specular reflection vector R.

$W(\theta)$ is typically set to a constant k_s, the material's specular-reflection coefficient, which ranges from between 0 to 1. The value of k_s is selected experimentally to produce aesthetically pleasing results. Note that V and R are the unit vectors in the viewing and specular-reflection directions, respectively. Therefore, we can calculate the value of $\cos\phi$ with the dot product $V \cdot R$. Considering above changes we can rewrite the equation for intensity of the specular reflection as

$$I_{spec} = k_s I_l\, (V \cdot R)^n$$

The vector R in the above equation can be calculated in terms of vector L and N. This calculation requires mirroring L about N. As shown in Fig. 10.8, this can be accomplished with some simple geometry. Since N and L are normalized, the projection of L onto N is $N \cos\theta$. Note that $R = N\cos\theta + S$, where $|S|$ is $\sin\theta$. But, by vector subtraction and congruent triangles, S is just $N\cos\theta - L$. Therefore,

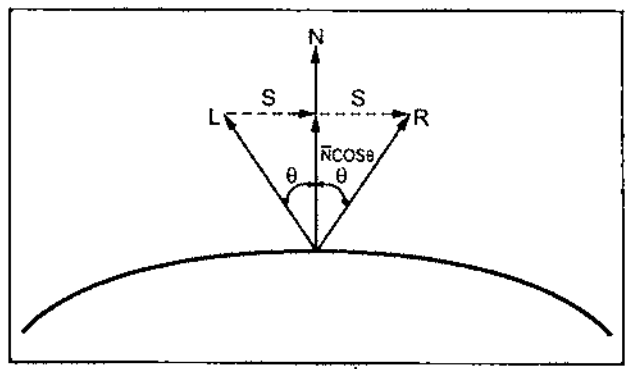


Fig. 10.8 Calculating the reflection vector

$$R = N\cos\theta + N\cos\theta - L$$
$$= 2N\cos\theta - L$$

Substituting $N \cdot L$ for $\cos\theta$ we have,

$$R = 2N\,(N \cdot L) - L$$

10.4.2 The Halfway Vector

More simplified way of formulation of Phong's illumination model is the use of halfway vector H. It is called halfway vector because its direction is halfway between the directions of the light source and the viewer as shown in the Fig. 10.9.

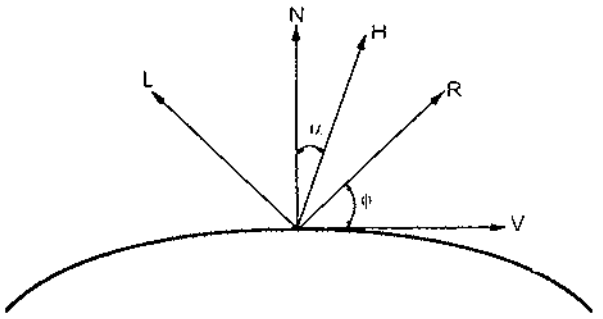


Fig. 10.9 Halfway vector H

If we replace V · R in the Phong model with the dot product N · H, this simply replaces the empirical cos φ calculation with the empirical cos α calculation (Refer Fig. 10.9). The halfway vector is given as

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

When the light source and the viewer are both at infinity, then the use of N · H offers a computational advantage, since H is constant for all surface points. Substituting N · H. in place of V · R the intensity for specular reflection is given as

$$I_{spec} = k_s I_l (N \cdot H)^n$$

For given light-source and viewer positions, vector H gives the orientation direction for the surface that would produce maximum specular reflection in the viewing direction. Thus, H is also referred to as the surface orientation direction for maximum highlights.

10.5 Combined Diffuse and Specular Reflections

For a single point light source, the combined diffuse and specular reflections from any point on the illuminated surface is given as

$$\begin{aligned} I &= I_{diff} + I_{spec} \\ &= k_a I_a + k_d I_l (N \cdot L) + k_s I_l (N \cdot H)^n \end{aligned}$$

For a multiple point light source the above equation can be modified as

$$I = k_a I_a + \sum_{i=1}^{M} I_{l_i} [k_d (N \cdot L_i) + k_s (N \cdot H_i)^n]$$

Therefore, in case of multiple point light sources the light reflected at any surface point is given by summing the contributions from the individual sources.

hading Algorithms

From the previous discussion it is clear that we can shade any surface by calculating the surface normal at each visible point and applying the desired illumination model at that point. Unfortunately, this shading method is expensive. In this section, we discuss more efficient shading methods for surfaces defined by polygons. Each polygon can be drawn with a single intensity, or with different intensity obtained at each point on the surface. Let us see various shading methods.

10.6.1 Constant-Intensity Shading

The fast and simplest method for shading polygon is constant shading, also known as **faceted shading** or **flat shading**. In this method, illumination model is applied only once for each polygon to determine single intensity value. The entire polygon is then displayed with the single intensity value.

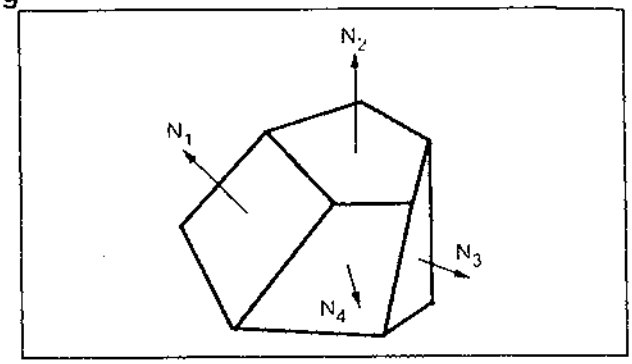


Fig. 10.10 Polygons and their surface normals

This method is valid for the following assumptions :

1. The light source is at infinity, so N · L is constant across the polygon face.
2. The viewer is at infinity, so V · R is constant over the surface.
3. The polygon represents the actual surface being modeled, and is not an approximation to a curved surface.

If either of the first two assumptions are not true still we can use constant intensity shading approach; however, we require some method to determine a single value for each of L and V vectors.

10.6.2 Gouraud Shading

In this method, the intensity interpolation technique developed by Gouraud is used, hence the name. The polygon surface is displayed by linearly interpolating intensity values across the surface. Here, intensity values for each polygon are matched with the values of adjacent polygons along the common edges. This eliminates the intensity discontinuities that can occur in flat shading.

By performing following calculations we can display polygon surface with Gouraud shading.

1. Determine the average unit normal vector at each polygon vertex.
2. Apply an illumination model to each polygon vertex to determine the vertex intensity.
3. Linearly interpolate the vertex intensities over the surface of the polygon.

We can obtain a normal vector at each polygon vertex by averaging the surface normals of all polygons sharing that vertex. This is illustrated in Fig. 10.11.

As shown in the Fig. 10.11, there are three surface normals N_1, N_2 and N_3 of polygon sharing vertex V. Therefore, normal vector at vertex V is given as

$$N_v = \frac{N_1 + N_2 + N_3}{|N_1 + N_2 + N_3|}$$

In general, for any vertex position V, we can obtain the unit vertex normal by equation

$$N_V = \frac{\sum_{i=1}^{n} N_i}{\left|\sum_{i=1}^{n} N_i\right|}$$

where n is the number of surface normals of polygons sharing that vertex.

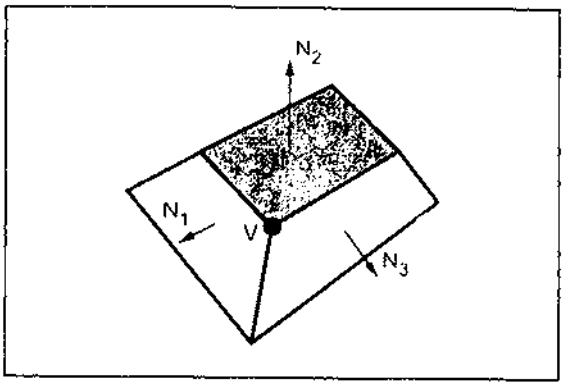


Fig. 10.11 Calculation of normal vector at polygon vertex V

The next step in Gouraud shading is to find vertex intensities. Once we have the vertex normals, their vertex intensities can be determined by applying illumination model to each polygon vertex. Finally, each polygon is shaded by linear interpolating of vertex intensities along each edge and then between edges along each scan line. This is illustrated in Fig. 10.12.

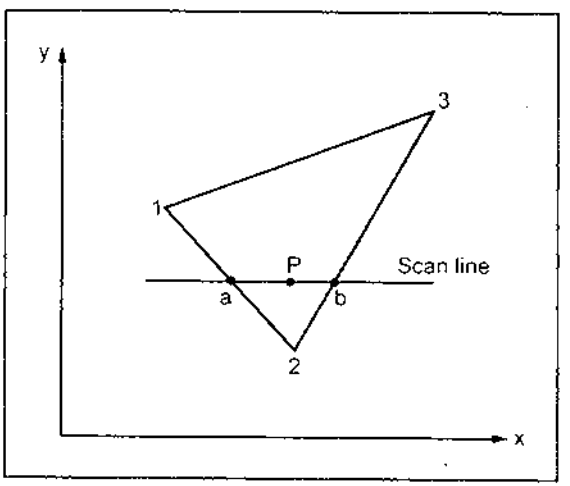


Fig. 10.12

For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints. For example, in Fig. 10.12, the polygon edge with endpoint vertices 1 and 2 is intersected by the scan line at point 'a'. The intensity at point 'a' can be interpolated from intensities I_1 and I_2 as

$$I_a = \frac{y_a - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_a}{y_1 - y_2} I_2$$

Similarly, we can interpolate the intensity value for right intersection (point b) from intensity values I_2 and I_3 as

$$I_b = \frac{y_a - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_b}{y_3 - y_2} I_2$$

Once the intensities of intersection points a and b are calculated for a scan line, the intensity of an interior point (such as P in Fig. 10.12) can be determined as

$$I_p = \frac{x_b - x_p}{x_b - x_a} I_a + \frac{x_p - x_a}{x_b - x_a} I_b$$

During the scan conversion process, usually incremental calculations are used to obtain the successive edge intensity values between the scan lines and to obtain successive intensity along a scan line. this eliminates the repeatative calculations.

If the intensity at edge position (x, y) is interpolated as

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

then we can obtain the intensity along this edge for the next scan line, y – 1 as (see Fig. 10.13)

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

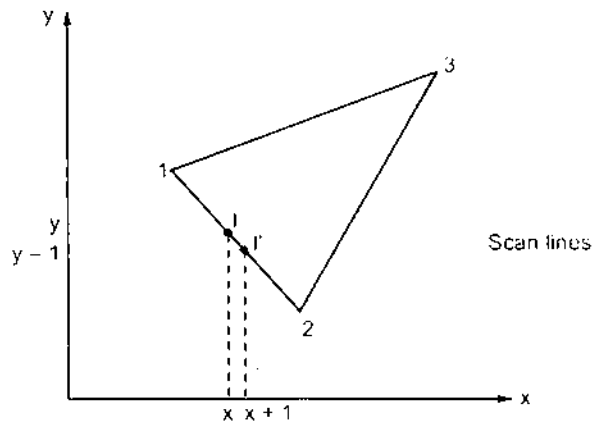


Fig. 10.13 Calculation of incremental interpolation of intensity values along a polygon edge for successive scan lines

Similarly, we can obtain intensities at successive horizontal pixel positions along each scan line (see Fig. 10.14) as

$$I' = I + \frac{I_b - I_a}{x_b - x_a}$$

Advantages

1. It removes the intensity discontinuities exists in constant shading model.
2. It can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.

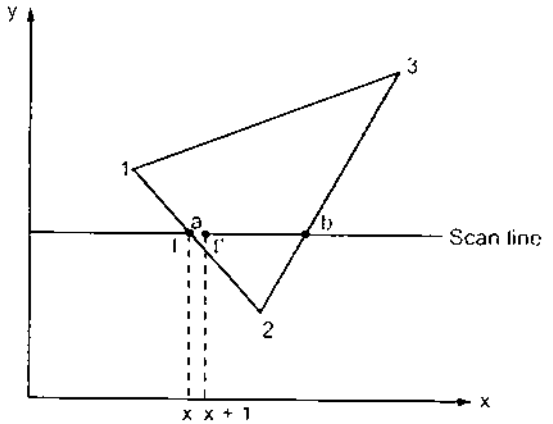


Fig. 10.14 Calculation of incremental interpolation of intensity values along a scan line

Disadvantages

1. Highlights on the surface are sometimes displayed with anomalous shapes.
2. The linear intensity interpolation can result bright or dark intensity streaks to appear on the surface. These bright or dark intensity streaks, are called **Mach bands**. The mach band effect can be reduced by breaking the surface into a greater number of smaller polygons.
3. Sharp drop of intensity values on the polygon surface can not be displayed.

10.6.3 Phong Shading

Phong shading, also known as **normal-vector interpolation shading**, interpolates the surface normal vector N, instead of the intensity. By performing following steps we can display polygon surface using Phong shading.

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

The first steps in the Phong shading is same as first step in the Gouraud shading. In the second step the vertex normals are linearly interpolated over the surface of the polygon. This is illustrated in Fig. 10.15. As shown in the Fig. 10.15, the normal vector N for the scan line intersection point along the edge between vertices 1 and 2 can be obtained by vertically interpolating between edge endpoint normals :

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Like, Gouraud shading, here also we can use incremental methods to evaluate normals between scan lines and along each individual scan line. Once the surface normals are evaluated the surface intensity at that point is determined by applying the illumination model.

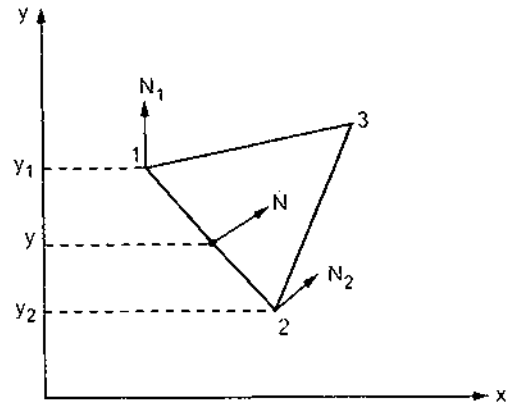


Fig. 10.15 Calculation of interpolation of surface normals along a polygon edge

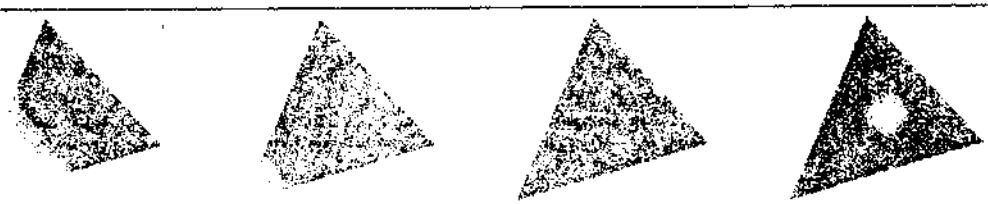
Advantages

1. It displays more realistic highlights on a surface. (See Fig. 10.16 d)
2. It greatly reduces the Mach-band effect.
3. It gives more accurate results.

Disadvantage

1. It requires more calculations and greatly increases the cost of shading steeply.

Fig. 10.16 shows the improvement in display of polygon surface using Phong shading over Gouraud shading.



(a) Gouraud shading (b) Phong shading (c) Gouraud shading (d) Phong shading

Fig. 10.16

Method of Speeding Up Phong Shading Technique

Phong shading is applied by determining the average unit normal vector at each polygon vertex and then linearly interpolating the vertex normals over the surface of the polygon. Then apply an illumination model along each scan line to calculate projected pixel intensities for the surface points. Phong shading can be speeded up by the intensity calculations using a Taylor- Series expansion and triangular surface patches. Since phong shading interpolates normal vectors from vertex normals, we can express the surface normal N at any point (x, y) over a triangle as

$$N = A_x + B_y + C$$

where A, B, C are determined from three vertex equations

$N_k = Ax_k + By_k + C \quad k = 1, 2, 3$ with (x_k, y_k) denoting a vertex positions. Discarding the reflectivity and attenuation parameters, the calculations for light source diffuse reflection from a surface point (x, y) as

$$I_{diff}(x, y) = \frac{L \cdot N}{|L|\ |N|} = \frac{L \cdot (A_x + B_y + C)}{|L|\ |A_x + B_y + C|}$$

$$= \frac{(L \cdot A)\, x + (L \cdot B)\, y + L \cdot C}{|L|\ |A_x + B_y + C|}$$

Now the expression can be rewritten in the form as

$$I_{diff}(x, y) = \frac{ax + by + c}{(dx^2 + exy + fy^2 + gx + hy + i)^{1/2}}$$

Where parameters a, b, c and d are used to represent the various dot products. We can express the denominator as a Taylor - series expansion and retain terms up to second degree in x and y.

$$I_{diff}(x, y) = T_5 x^2 + T_4 x y + T_3 y^2 + T_2 x + T_1 y + T_0$$

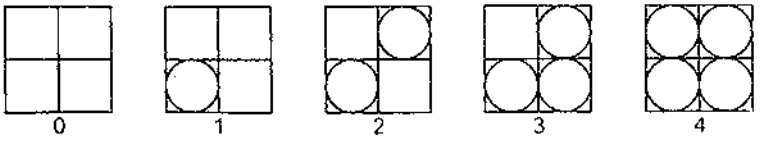
where each T_k is a functions of parameter a, b, c and so forth.

Using forward differences, we can evaluate above equation with only two additions for each pixel position (x, y) once the initial forward difference parameters have been evaluated. Thus the fast phong shading technique reduces the calculations and speed up the process.

10.6.4 Halftone Shading

Many displays and hardcopy devices are bilevel. They can only produce two intensity levels. In such displays or hardcopy devices we can create an apparent increase in the number of available intensities. This is achieved by incorporating multiple pixels positions into the display of each intensity value. When we view a very small area from a sufficiently large viewing distance, our eyes average fine details within the small area and record only the overall intensity of the area. This phenomenon of apparent increase in the number of available intensities by considering combine intensity of multiple pixels is known as **halftoning**. The halftoning is commonly used in printing black and white photographs in newspapers, magazines and books. The pictures produced by halftoning process are called **halftones**.

In computer graphics, halftone reproductions are approximated using rectangular pixel regions, say 2×2 pixels or 3×3 pixels. These regions are called halftone patterns or pixel patterns. Fig. 10.17 shown the halftone patterns to create number of intensity levels.



$0 \le I < 0.2$ $0.2 \le I < 0.4$ $0.4 \le I < 0.6$ $0.6 \le I < 0.8$ $0.8 \le I < 1.0$

Fig. 10.17 (a) 2×2 Pixel patterns for creating five intensity levels

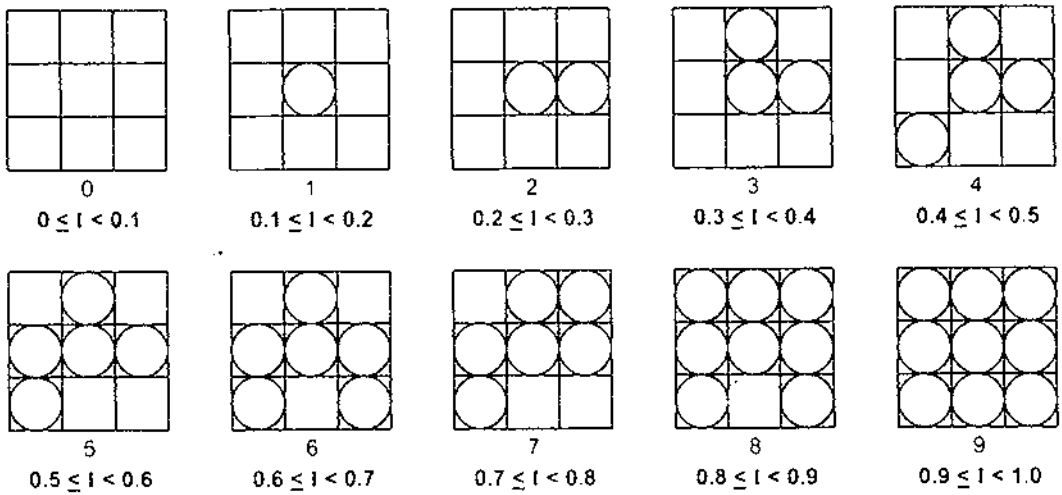


Fig. 10.17 (b) 3 × 3 Pixel patterns for creating ten intensity levels

10.6.5 Dithering Techniques

Dithering refers to techniques for approximating halftones without reducing resolution, as pixel grid patterns do. The term dithering is also applied to halftone approximation methods using pixel grids, and sometimes it is used to refer to colour halftone approximations only.

Random values added to pixel intensities to break up contours are often referred as **dither noise**. Number of methods are used to generate intensity variations. Ordered dither methods generate intensity variations with a one-to-one mapping of points in a scene to the display pixels. To obtain n^2 intensity levels, it is necessary to set up an $n \times n$ dither matrix D_n whose elements are district positive integers in the range of 0 to $n^2 - 1$. For e.g. it is possible to generate four intensity levels with

$$D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$ and it is possible to generate nine intensity levels with

$$D_3 = \begin{bmatrix} 7 & 2 & 6 \\ 4 & 0 & 1 \\ 3 & 8 & 5 \end{bmatrix}$$

The matrix elements for D_2 and D_3 are in the same order as the pixel mask for setting up 2×2 and 3×3 pixel grids respectively. For bilevel system, we have to determine display intensity values by comparing input intensities to the matrix elements. Each input intensity is first scaled to the range $0 \le I \le n^2$. If the intensity I is to be applied to screen position (x, y), we have to calculate row and column numbers for the either matrix as

$$i = (x \bmod n) + 1, \qquad j = (y \bmod n) + 1$$

If $I > D_n(i, j)$ the pixel at position (x, y) is turned on; otherwise the pixel is not turned on. Typically, the number of intensity levels is taken to be a multiple of 2. High order dither matrices can be obtained from lower order matrices with the recurrence relation.

$$D_n = \begin{bmatrix} 4\,D_{n/2} + D_2(1,\,1)\,u_{n/2} & 4\,D_{n/2} + D_2\,(1,2)\,u_{n/2} \\ 4\,D_{n/2} + D_2(2,\,1)\,u_{n/2} & 4\,D_{n/2} + D_2\,(2,2)\,u_{n/2} \end{bmatrix}$$

assuming $n \geq 4$. Parameter $u_{n/2}$ is the unity matrix.

Another method for mapping a picture with $m \times n$ points to a display area with $m \times n$ pixels is **error diffusion**. Here, the error between an input intensity value and the displayed pixel intensity level at a given position is dispersed, or diffused to pixel positions to the right and below the current pixel position.

10.7 Transparency

In the shading models we have not considered the transparent objects. A transparent surface, in general, produces both reflected and transmitted light. It has a transparency coefficient T as well as values for reflectivity and specular reflection. The coefficient of transparency depends on the thickness of the object because the transmission of light depends exponentially on the distance which the light ray must travel within the object. The expression for coefficient of transparency is given as

$$T = te^{-ad}$$

Where t is the coefficient of property of material which determines how much of the light is transmitted at the surface instead of reflected, a is the coefficient of property of material which tells how quickly the material absorbs or attenuates the light, d is the distance the light must travel in the object.

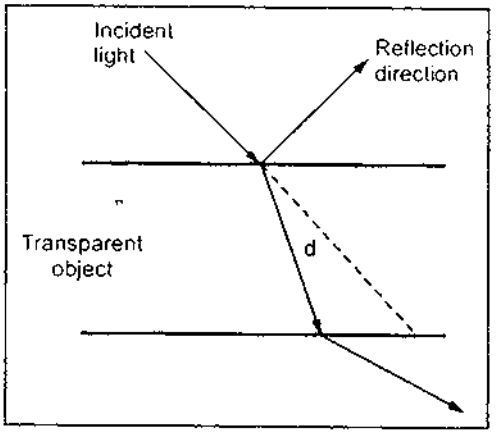


Fig. 10.18 Refraction

When light crosses the boundary between two media it changes the direction as shown in the Fig. 10.18. This effect is called **refraction**. The effect of refraction is observed because the speed of light is different in different materials resulting different path for refracted light from that of incident light. The direction of the refracted light is specified by the **angle of refraction** (θ_r). It is the function of the property materials called the **index of refraction** (n). The angle of refraction θ_r is calculated from the angle of incidence θ_i, the index of refraction n_i of the incident material (usually air), and the index of refraction n_r of the refracting material according to Snell's law :

$$\sin\theta_r = \frac{n_i}{n_r}\sin\theta_i$$

In practice, the index of refraction of a material is a function of the wave length of the incident light, so that the different colour components of a light ray refracts at different angles. The transparency and absorption coefficients are also depend on colour. Therefore, when we are dealing with colour objects we require three pairs of transparency and absorption coefficients.

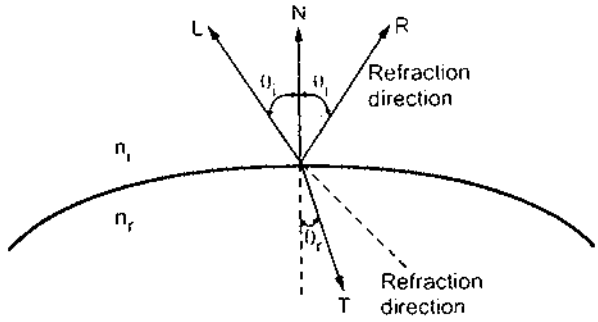


Fig. 10.19 Refraction direction and angle of refraction θ_i

For modeling of transparent surface we have to consider contributions from the light reflected from the surface and the light coming from behind the object. If we assume for a given surface that

- The transparency coefficient for the object is a constant
- Refraction effects are negligible and
- No light source can be seen directly through the object,

then the light coming through the object is given as,

$$v = v_r + t\, v_t$$

where v is the total amount of light,

v_r is the amount of light reflected from the surface,

t is the transparency coefficient and,

v_t is the light coming from behind the object.

To get more realistic images we have to consider the angular behavior of the reflection v_r, transmission at the surface and also the attenuation due to thickness. The simple approximation for this behavior can be given as

$$t = (t_{max} - t_{min})(N \cdot E)^{\alpha} + t_{min}$$

where $(N \cdot E)^{\alpha}$ is the cosine of the angle between the eye and the surface normal raised to the power. This angle decides the distance the light must travel through the object. When viewed straight on, angle is 0 i.e. cosine of angle is 1 (highest) and the distance travelled by light is minimum. When viewed at a glancing angle, cosine is less than 1 and the distance travelled by light is more. Therefore, we can say that cosine of angle is maximum when surface is viewed straight on and it drops off for glancing views. The power of angle represented by α enhances the effect. The values of α of 2 or 3 give reasonable effects.

10.8 Shadows

A shadowed object is one which is hidden from the light source. It is possible to use hidden surface algorithms to locate the areas where light sources produce shadows. In order to achieve this we have to repeat the hidden-surface calculation using light source as the viewpoint. This calculation divides the surfaces into shadowed and unshadowed groups. The surfaces that are visible from the light source are not in shadow; those that are not visible from the light source are in shadow. Surfaces which are visible and which are also

visible from the light source are shown with both the background illumination and the light-source illumination. Surfaces which are visible but which are hidden from the light source are displayed with only the background illumination, as shown in the Fig. 10.20.

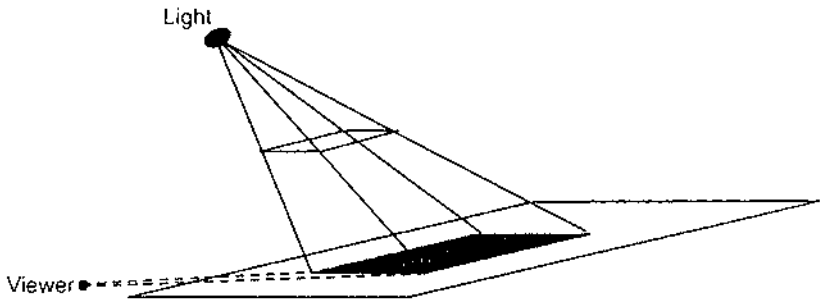


Fig. 10.20 Shadow

Another way to locate shadow areas is the use of **shadow volumes**. A shadow volume is defined by the light source and an object and is bounded by a set of invisible shadow polygons, as shown in the Fig. 10.21. This volume is also known as polygon's shadow volume. By comparing visible polygon with this volume we can identify the portions which lie inside of the volume and which are outside of the volume. The portions which lie inside of the volume are shadowed, and their intensity calculations do not include a term from the light source. the polygons or portions of polygons which lie outside the shadow volume are not shaded by this polygon, but might be shaded by some other polygon so they still must be checked against the other shadow volume.

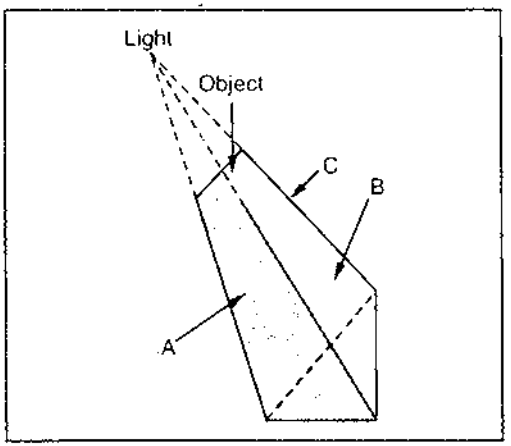


Fig. 10.21

10.9 Ray-Tracing

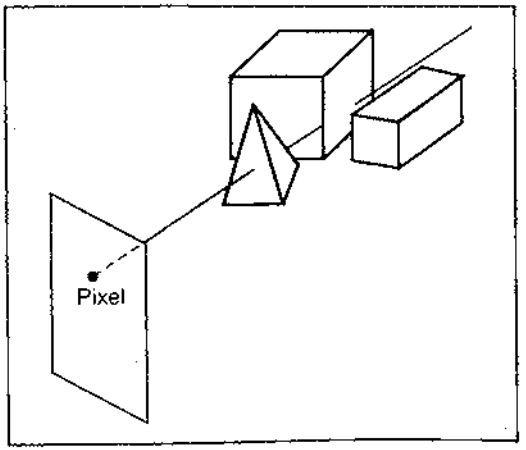


Fig. 10.22 A ray along the line of sight from a pixel position through a scene

If we consider the line of sight from a pixel position on the view plane through a scene, as in Fig. 10.22, we can determine which objects in the scene (if any) intersect this line. From the intersection points with different object, we can identify the visible surface as the one whose intersection point is closest to the pixel. **Ray tracing** is an extension of this basic idea. Here, instead of identifying for the visible surface for each pixel, we continue to bounce the ray around the picture. This is illustrated in Fig. 10.23. When the ray is bouncing from one surface to another surface, it contributes the

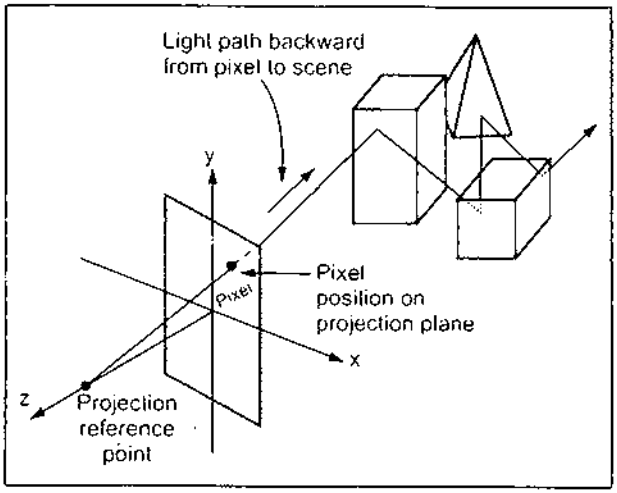


Fig. 10.23 Bouncing of ray around the scene

intensity for that surfaces. This is a simple and powerful rendering technique for obtaining global reflection and transmission effects.

As shown in the Fig. 10.23, usually pixel positions are designated in the xy plane and projection reference point lie on the z axis, i.e. the pixel screen area is centered on viewing coordinate origin. With this coordinate system the contributions to a pixel is determined by tracing a light path backward from the pixel to the picture.

For each pixel ray, each surface is tested in the picture to determine if it is intersected by the ray. If surface is intersected, the distance from the pixel to the surface intersection point is calculated. The smallest calculated intersection distance identifies the visible surface for that pixel. Once the visible surface is identified the ray is reflected off the visible surface along a specular path where the angle reflection equals angle of incidence. If the surface is transparent, the ray is passed through the surface in the refraction direction. The ray reflected from the visible surface or passed through the transparent surface in the refraction direction is called **secondary ray**. The ray after reflection or refraction strikes another visible surface. This process is repeated recursively to produce the next generations of reflection and refraction paths. These paths are represented by **ray tracing tree** as shown in the Fig. 10.24.

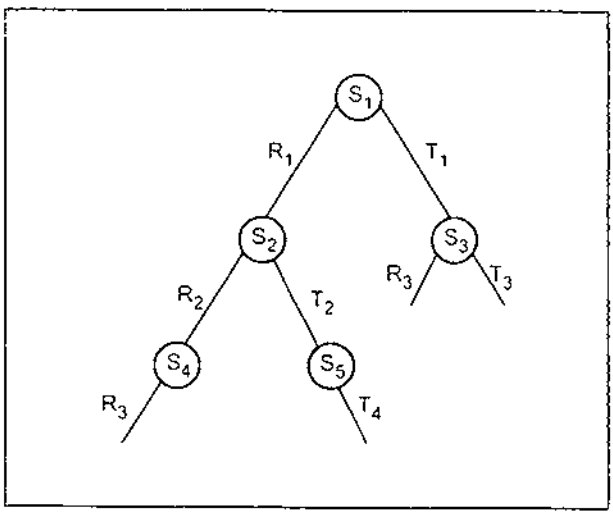


Fig. 10.24 Binary ray-tracing tree

As shown in the Fig. 10.24, the left branches in the binary ray tracing tree are used to represent reflection paths, and right branches are used to represent transmission paths. The recursion depth for ray tracing tree is determined by the amount of storage available, or by the user. The ray path is terminated when predetermined depth is reached or if ray strikes a light source. As we go from top to bottom of the tree, surface intensities are attenuated by distance from the parent surface. The surface intensities of all the nodes are added traversing the ray tree from bottom to top to determine the intensity of the pixel.

If pixel ray does not intersect to any surface then the intensity value of the background is assigned to the pixel. If a pixel ray intersects a nonreflecting light source, the pixel can be assigned the intensity of the source, although light sources are usually placed beyond the path of the initial rays.

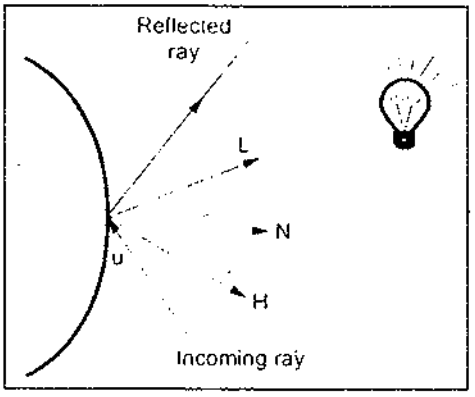


Fig. 10.25 Surface intersected by a ray and the unit vectors

The Fig. 10.25 shows a surface intersected by a ray and unit vectors needed for the reflected light-intensity calculations. Here, u is the unit vector in the direction of the ray path, N is the unit surface normal, R is the unit reflection vector, L is the unit vector pointing to the light source, and H is the unit vector halfway between V (viewer) and L (light source).

If any object intersects the path along L between the surface and the point light source, the surface is in shadow with respect to that source. Hence a path along L is referred to as **shadow ray**. Ambient light at the surface is given as $K_a I_a$, diffuse reflection due to the surface is proportional to K_d (N.L), and the specular-reflection component is proportional to K_s $(H.N)^{n_s}$. We know that, the specular reflection direction for R depends on the surface normal and the incoming ray direction. It is given as

$$R = u - (2u \cdot N) N$$

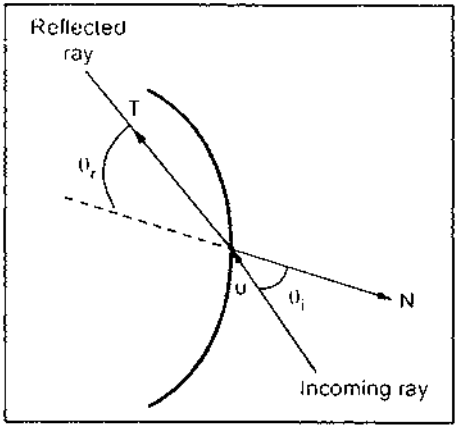


Fig. 10.26 Refracted ray through the transparent material

In a transparent material light passes through the material and we have to calculate intensity contributions from light transmitted through the material. Referring the Fig. 10.26, we can obtain the unit transmission vector from vectors u and N as

$$T = \frac{\eta_i}{\eta_r} u - (\cos\theta_r - \frac{\eta_i}{\eta_r} \cos\theta_i) N$$

where η_i and η_r are the indices of relfection in the incident material and the refracting meterial, respectively. The angle of refraction θ_r is given by Snell's law

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 (1 - \cos^2\theta_i)}$$

10.9.1 Ray Surface Intersection Calculations

The ray equation is given as

$$P = P_0 + su$$

Where P_0 is the initial position of ray, P is any point along the ray path at distance s from P_0 and u is the unit direction vector. The ray equation gives the coordinates of any point P along the ray path at a distance s from P_0. Initially, P_0 is set to the position of the pixel on the projection plane, or it is chosen as a projection reference point. Unit vector u is initially obtained from the position of the pixel through which the ray passes and the projection reference point

$$u = \frac{P_{pixel} - P_{prp}}{|P_{pixel} - P_{prp}|}$$

Vectors P_0 and u are updated for the secondary rays at the ray-surface intersection point at each intersected surface. For the secondary rays, reflection direction for u is R and the transmission direction is T. We can locate the surface intersections by simultaneously solving the ray equation and the surface equation for the individual objects in the scene.

The simplest object to ray trace is sphere, i.e. we can easily identify that whether the ray does intersect the sphere or not; and if it intersects we can easily obtain the surface intersection coordinates from the ray equation. Consider the sphere of radio r and center position P_c, as shown in Fig. 10.27. P is any point on the sphere which satisfies the sphere equation :

$$|P - P_c|^2 - r^2 = 0$$

Substituting the value of P from ray equation we can write above equation as

$$|P_0 + s_u - P_c|^2 - r^2 = 0$$

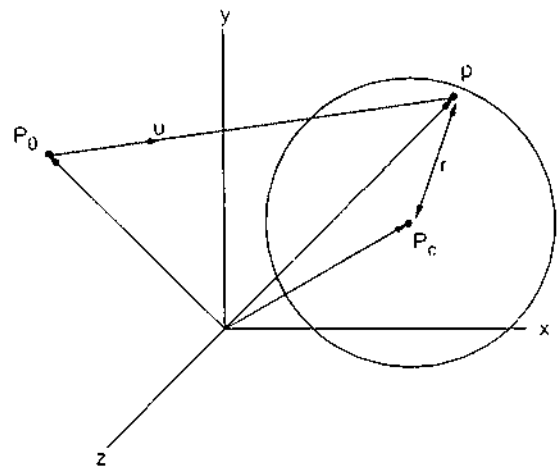


Fig. 10.27 A ray intersecting a sphere having radius r centered on position P_c

If we assume $\Delta P = P_c - P_0$ and expand the dot product, we get the quadratic equation

$$s^2 - 2\,(u \cdot \Delta P)\, s + (|\Delta P|^2 - r^2) = 0$$

By solving quadratic equation we get,

$$s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P|^2 + r^2}$$

In the above equation if the discriminant is negative we can say that the ray does not intersect the sphere; otherwise the surface intersection coordinates can be obtained from the ray equation.

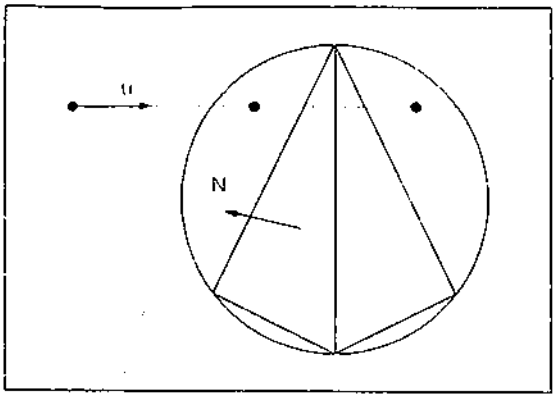


Fig. 10.28 Polyhedron bounded by a sphere

In case of polyhedra more processing is required to locate the surface intersections. For this reason instead of doing intersection calculations directly, first the intersection test is carried out on bounding volume, as shown in Fig. 10.28; and if the ray intersects the bounding volume further tests and intersection calculations are carried out. As shown in Fig. 10.28, the polyhedron is bounded by a sphere. If a ray does not intersect the sphere, we do not need to do any further testing on the polyhedron. But if ray does intersect the sphere, we have to locate front faces with the test

$$u \cdot N < 0$$

Where N is a surface normal. For each face of the polyhedron that satisfies inequality in above equation, we have to solve the plane equation as

$$N \cdot P = -D$$

for surface position P that also satisfies the ray equation. With these initial calculations we can say that the position P is both on the plane and on the ray path if

$$N \cdot (P_0 + Su) = -D$$

and the distance from the initial ray position to the plane is

$$s = -\frac{D + N \cdot P_0}{N \cdot u}$$

The above calculations gives us a position on the infinite plane that contain the polygon face, however they do not satisfy that the position is inside or outside the polygon boundaries. Therefore, to determine whether the ray intersected this face of the polyhedron, we have to perform an inside-outside test discussed in section 3.5.

In case of other objects, such as quadric or spline surfaces we have to follow the same procedure to calculate ray-surface intersection positions.

10.9.2 Reducing Object-Intersection Calculations

When scene contains more than one objects, most of the processing time for each ray is spent in checking objects that are not visible along the ray path. Therefore, as discussed earlier, adjacent objects are enclosed in groups within a bounding volume, such as a sphere or a box. We can then proceed for intersection calculations only when the ray intersects the bounding volume. This approach can be extended to include a hierarchy of bounding volumes. That is, we enclose several bounding volumes within a larger volume and carry out the intersection test hierarchically. In this, we first test the outer bounding volume and then if necessary test the smaller inner bounding volumes and so on.

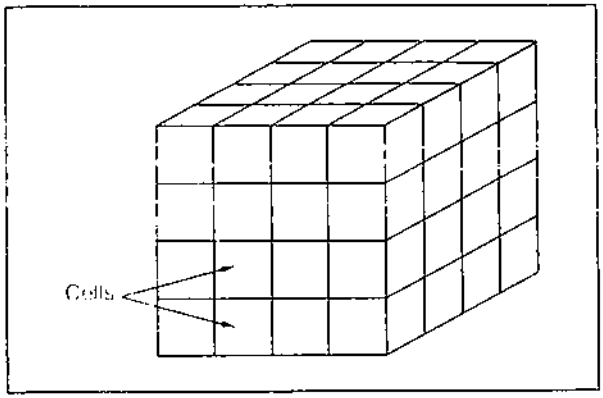


Fig. 10.29 Subdivision of cube into cells

Another method known as **space-subdivision method** is also used to reduce intersection calculations. In this method, the scene is enclosed within a cube and a cube then successively subdivided until each subregion (cell) contains no more than a preset maximum number of surfaces. For example, one surface per cell. We then trace rays through the individual cells of the cube, performing intersection tests only within those cells containing surfaces. There is a trade-off between the cell size and the number of surfaces per cell. If we set the maximum number of surfaces per cell too low, cell size can become too small and cell number can become too large increasing cell-traversal processing.

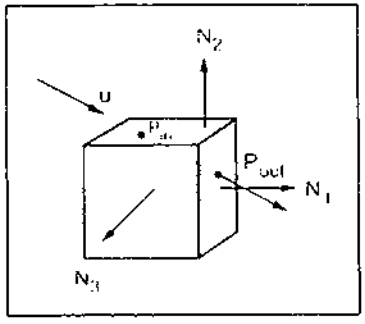


Fig. 10.30 Traversal of ray through a cell

Initially, we have to determine the intersection point on the front face of the cube. It can be determined by checking the intersection coordinates against the cell boundary positions. We then need to process the ray through the cells by determining the entry and exit points as shown in the Fig. 10.30, for each cell traversed by the ray until ray intersect and object surface or exit the cube.

If a ray direction is u and a ray entry point is P_{in} for a cell, the potential exit faces are those for which

$$u \cdot N_k > 0$$

where N_k are the normal vectors. If these vectors are aligned with coordinate axes,

then

$$N_k = \begin{cases} (\pm 1, 0, 0) \\ (0, \pm 1, 0) \\ (0, 0, \pm 1) \end{cases}$$

and to determine the three candidate exits plane we have to check only the sign of each component of u. The exit position on each candidate plane can be obtained from the ray equation as

$$P_{out,k} = P_{in} + S_k u$$

where S_k is the distance along the ray from P_{in} to $P_{out,k}$. Substituting the ray equation into the plane equation for each face of the cell we have,

$$N_k \cdot P_{out,k} = -D$$

Now, the ray distance to each candidate exit face can be given as

$$S_k = \frac{-D - N_k \cdot P_{in}}{N_k \cdot u}$$

We have to select smallest S_k. The above calculations are simple when the normal vectors are aligned with the candidate axes. For example, if a candidate normal vector is (0, 1, 0), then for that plane we have

$$S_k = \frac{x_k - x_0}{u_y}$$

where $u = (u_x, u_y, u_z)$, and x_k is the value of the right boundary face for the cell.

10.9.3 Antialiased Ray Tracing

Traditional ray tracing systems suffer from aliasing artifacts. The term aliasing in computer graphics is loosely defined. It can mean almost anything unwanted in the rendered image. Typically aliasing is used to describe jagged edges. In the real world things are not quite as perfect as in a computer generated world-edges and boundaries are not as sharp, reflections are not as perfect, and things can be in and out of focus. If a renderer is being used to approximate reality then these things must be taken into account.

Three basic techniques are used to perform antialiased ray tracing. These are super sampling, adaptive sampling and stochastic sampling. In these sampling methods, the pixel is treated as a finite square area instead of a single point.

10.9.3.1 Super Sampling

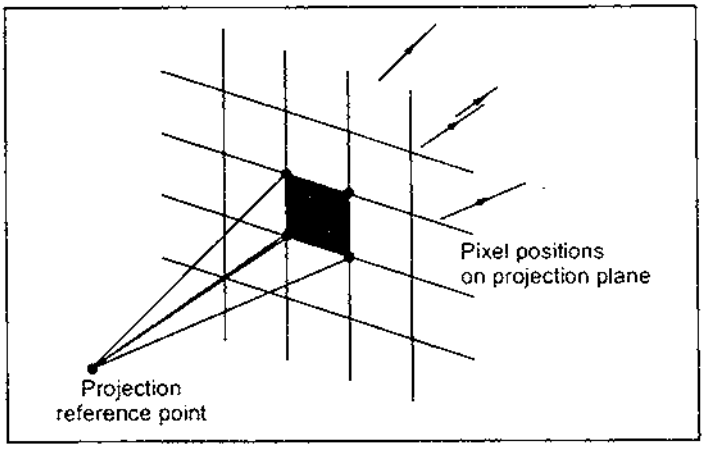


Fig. 10.31 Super sampling procedure

In supersampling, multiple, evenly spaced rays (samples) are taken over each pixel area. The Fig. 10.31 shows a simple supersampling procedure with four rays per pixel, one at each pixel corner. To determine the overall pixel intensity, the intensity of these pixel rays are averaged. However, if the intensities for the four rays are not appropriately equal, or if some small object lies between the four rays, we have to divide the pixel area into subpixels and repeat the process. This is illustrated in Fig. 10.32. Here, pixel area divided into nine subpixels using 16 rays, one at each subpixel corner.

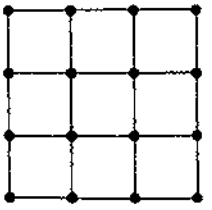


Fig. 10.32 Subdivision of pixel into nine subpixels

10.9.3.2 Adaptive Sampling

In adaptive sampling, multiple, unevenly spaced rays (samples) are taken in some regions of the pixel area. For example, more rays can be taken near object edges to obtain a [illegible] of the pixel intensities. Again, to determine the overall pixel intensity where multiple rays are used, the intensity of rays from subpixels are averaged. However, the subpixels that do not have nearly equal intensity rays are further subdivided until each subpixel has approximately equal intensity rays or an upper bound, say, 256, has been reached for the number of rays per pixel.

10.9.3.3 Stochastic Sampling / Distributed Ray Tracing

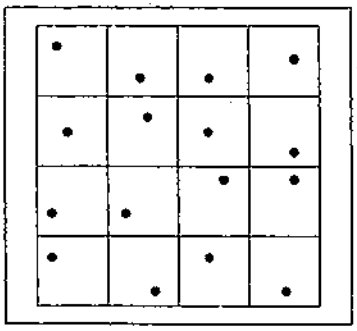


Fig. 10.33 The random distribution of rays

Distributed ray tracing is a stochastic sampling method. It is not ray tracing on a distributed system. It is a ray tracing method based on randomly distributed rays over the pixel area (Refer Fig. 10.33) used to reduce aliasing effect. In this method, the multiple samples are taken and averaged together. The location of where the sample is random so that the resulting average in an approximation of a finite area covered by the samples.

The random distribution of a number of rays over the pixel surface is achieved by the technque called **jittering**. In this technique, initially, pixel area is divided into the 16 subareas as shown in the Fig. 10.33. Then random ray positions are obtained by jittering the center coordinates of each subpixel area by small amounts say δx and δy, where both δx and δy are assigned values in the inverter (– 0.5, 0.5). Therefore, if center position of a cell is specified as (x, y) then the jitter position is $(x + \delta x, y + \delta y)$.

10.9.3.4 Advantages of Distributed Ray Tracing

The intensity of a point in a scene can be represented analytically by an integral over the illumination function and the reflectance function. The evaluation of this integral, while extremely accurate, is too expansive for most graphics applications. Traditional ray tracing makes assumptions about the components of this integral to simplify evaluation. For example, the Phong model assumes that diffuse light is reflected equally in all directions, and specular light is at full intensity in the reflected direction and falls off exponentially with the cosine of the angle away from this direction. In addition, light sources are modeled as single points, so the light that emanates from a source and hits a surface can be represented by a single ray.

Distributed ray tracing uses a slightly better approximation for the illumination and reflectance integrals. The idea is based in the theory of **oversampling**. Instead of approximating an integral by a single scalar value, the function is point sampled and these samples are used to define a more accurate scalar value. The practical benefits of this are :

- Gloss (fuzzy reflections)
- Translucency
- Soft shadows
- Depth of field
- Motion blur

Gloss

Traditional ray tracing is good at representing perfect reflecting surfaces, but poor at representing glossy or partially reflecting surfaces. Only when surfaces are perfect mirrors do the relfections look identical to the scene they are reflecting. More often surfaces are glossy and reflect a blurred image of the scene. This is due to the light scattering properties of the surface. Reflections in traditional ray tracing are always sharp, even partial reflections. Glossy surfaces are generated in distributed ray tracing by randomly distributing rays reflected by a surface. Instead of casting a single ray out in the reflecting direction, a packet of rays are sent out around the reflecting direction. The actual value of reflectance can be found by taking the statistical mean of the values returned by each of these rays.

Translucency

Traditional ray tracing is good at representing perfectly transparent surfaces, but poor at representing translucent surfaces. Real surfaces that are translucent generally transmit a blurred image of the scene behind them. Distirbuted ray tracing achieves this type of translucent surface by casting randomly distributed rays in the general direction of the transmitted ray from traditional ray tracing. The value computed from each of these rays is then averaged to form the true translucent component.

Soft Shadows

Shadows in traditional ray tracing are discrete. When shading a point, each light source is checked to see if it is visible. If the source is visible it has a contribution to the shading of the point, otherwise it does not. The light source itself is modeled by a single point, which is fairly accurate for sources that are a great distance away, but a poor representation for large sources or sources that are close. The result of this discrete decision making is that the edges of shadows are very sharp. There is a distinct transition from when points are visible to the light source to when they are not. Shadows in the real world are much softer. The transition from fully shadowed to partially shadowed is gradual. This is due to the finite area of real light sources, and scattering of light of other surfaces. Distributed ray tracing attempts to approximate soft shadows by modeling light sources as spheres. When a point is tested to see if it is in shadow, a set of rays are cast about the projected area of the light source. The amount of light transmitted from the source to the point can be approximated by the ratio of the number of rays that hit the source to the number of rays cast. This ratio can be used in the standard Phong lighting calculations to scale the amount of light that hits a surface.

Depth of Field

Both the human eye and cameras have a finite lens aperture, and therefore have a finite depth of field. Objects that are two far away or two close will appear unfocused and blurry. Almost all computer graphics rendering techniques use a **pinhole camera** model. In this model all objects are in perfect focus regardless of distance. In many ways this is advantageous, blurring due to lack of focus is often unwanted in images. However,

simulating depth of field can lead to more realistic looking images because it more accurately models true optical systems. Distributed ray tracing creates depth of field by placing an artificial lens in front of the view plane. Randomly distributed rays are used once again to simulate the blurring of depth of field. The first ray cast is not modified by the lens. It is assumed that the focal point of the lens is at a fixed distance along this ray. The rest of the rays sent out for the same pixel will be scattered about the surface of the lens. At the point of the lens they will be bent to pass through the focal point. Points in the scene that are close to the focal point of the lens will be in sharp focus. Points closer or further away will be blurred.

Motion Blur

Animation in computer graphics is produced by generating a sequence of still images and then playing them back in order. This is yet another sampling process, but it is temporal rather than spatial. In movie cameras, each frame represents an average of the scene during the time that the camera shutter is open. If objects in the scene are in motion relative to the camera, then they will appear blurred on the film. Distributed ray tracing can simulate this blurring by distributing rays temporally as well as spatially. Before each ray is cast, objects are translated or rotated to their correct position for that frame. The rays are then averaged afterwards to give the actual value. Objects with the most motion will have the most blurring in the rendered image.

10.10 Colour Models

A colour model is a specification of a 3D colour coordinate system and a visible subset in the coordinate system within which all colours in a particular colour range lie. For example, RGB colour model is the unit cube subset of the 3D Cartesian coordinate system. The colour model allows to give convenient specification of colours in the specific colour range or gamut. There are three hardware oriented colour models : RGB, used for colour CRT monitors, YIQ used for the broadcast TV colour system, and CMY (Cyan, Magenta, Yellow) used for some colour printing devices. However, these models are not easy to use because they does not relate directly to intuitive colour notions of hue, saturation, and brightness. Therefore, another class of colour model has been developed. These include HSV, HLS and HVC models. In this chapter we are going to study RGB, CMY, HSV and HLS models.

10.10.1 Properties of Light

A light source produced by a sun or electric bulb emits all frequencies within the visible range to give white light. When this light is incident upon an object, some frequencies are absorbed and some are reflected by the object. The combination of reflected frequencies decides the colour of the object. If the lower frequencies are predominant in the reflected frequencies, the object colour is red. In this case, we can say that the perceived light has a dominant frequency at the red end of the spectrum. Therefore, the dominant frequency decides the colour of the object. Due to this reason dominant frequency is also called the **hue** or simply the **colour**.

Apart from the frequency there are two more properties which describe various characteristics of light. These are : brightness and saturation (purity). The brightness refers to the intensity of the perceived light. The saturation describes the purity of the colour. Pastels and pale colours are described as less pure or less saturated. When the two

properties purity and dominant frequency are used collectively to describe the colour characteristics, are referred to as **chromaticity**.

We know that two different colour light sources with suitably chosen intensities can be used to produce a range of other colour. But when two colour sources are combined to produce white colour, they are referred to as **complementary colours**. Red and cyan, green and magenta, and blue and yellow are complementary colour pairs. Usually, the colour model use combination of three colours to produce wide range of colours, called the **colour gamut** for that model. The basic colours used to produce colour gamut in particular model are called **primary colours**.

10.10.2 CIE Chromaticity Diagram

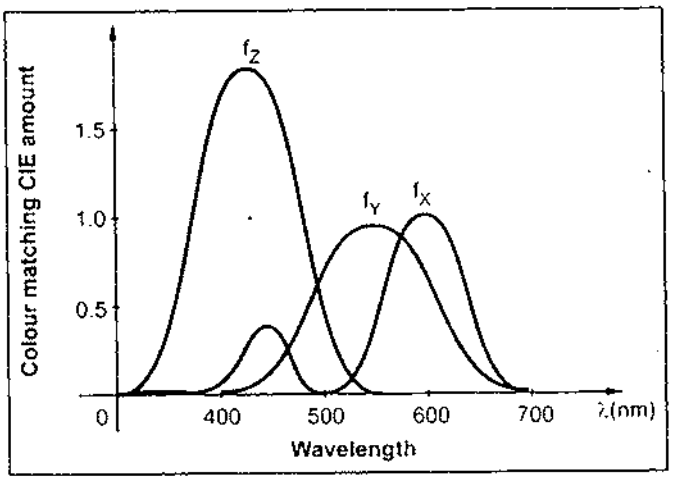


Fig. 10.34 Amounts of CIE primaries needed to display spectral colours

Matching and therefore defining a coloured light with a combination of three fixed primary colours is desirable approach to specify colour. In 1931, the Commission Internationale de l' Eclairage (CIE) defined three standard primaries, called X, Y and Z to replace red, green and blue. Here, X, Y and Z represent vectors in a three-dimensional, additive colour space. The three standard primaries are imaginary colours. They are defined mathematically with positive colour-matching functions, as shown in Fig. 10.34. They specify the amount of each primary needed to describe any spectral colour.

The advantage of using CIE primaries is that they eliminate matching of negative colour values and other problems associated with selecting a set of real primaries.

Any colour (C_λ) using CIE primaries can be expressed as

$$C_\lambda = X\,X + Y\,Y + Z\,Z$$

where X, Y and Z are the amounts of the standard primaries needed to match C_λ and X, Y and Z represent vectors in a three-dimensional, additive colour space.

With above expression we can define chromaticity values by normalizing against luminance (X + Y + Z). The normalizing amounts can be given as

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{X}{X+Y+Z}, \quad Z = \frac{Z}{X+Y+Z}$$

Notice that x + y + z = 1. That is, x, y and z are on the (X + Y + Z = 1) plane. The complete description of colour is typically given with the three values x, y and Y. The remaining values can be calculated as follows :

$$z = 1 - x - y, \qquad X = \frac{x}{y} Y, \qquad Z = \frac{z}{y} Y$$

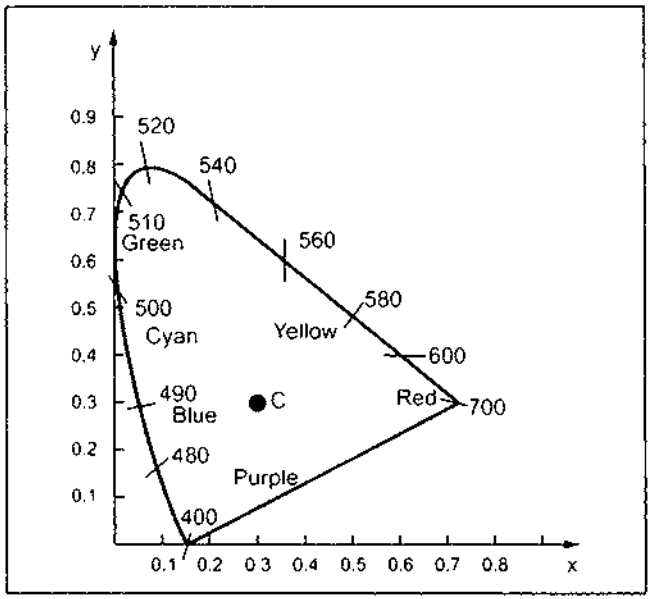


Fig. 10.35

Chromaticity values depend only on dominant wavelength and saturation and are independent of the amount of luminous energy. By plotting x and y for all visible colours, we obtain the CIE chromaticity diagram shown in Fig. 10.35, which is the projection onto the (X, Y) plane of the (X + Y + Z = 1) plane.

The interior and boundary of the tongue-shaped region represent all visible chromaticity values. The points on the boundary are the **pure** colours in the electromagnetic spectrum, labeled according to wavelength in nanometers from the red end to the violet end of the spectrum. A standard white light, is formally defind by a light source illuminant C, marked by the center dot. The line joining the red and violet spectral points is called the purple line, which is not the part of the spectrum.

The CIE chromaticity diagram is useful in many ways :

- It allows us to measure the dominant wavelength and the purity of any colour by matching the colour with a mixture of the three CIE primaries.
- It identifies the complementary colours.
- It allows to define colour gamuts or colour ranges, that show the effect of adding colours together.

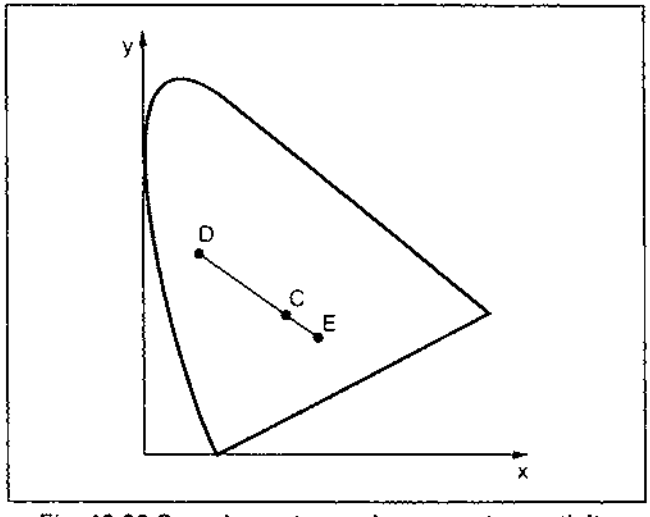


Fig. 10.36 Complementary colours on chromaticity diagram

The Fig. 10.36 represents the complementary colours on the chromaticity diagram. The straight line joining colours represented by points D and E passes through point C (represents white light). This means that when we mix proper amounts of the two colours D and E in Fig. 10.36, we can obtain white light. Therefore, colours D and E are complementary colours, and with point C on the chromaticity diagram we can identify the complement colour of the known colour.

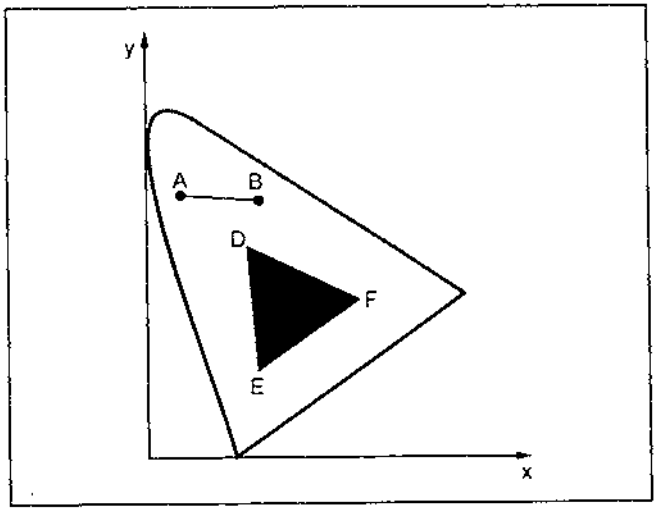


Fig. 10.37 Definition of colour gamuts on the chromaticity diagram

Colour gamuts are represented on the chromaticity diagram as a straight line or as a polygon. Any two colours say A and B can be added to produce any colour along their connecting line by mixing their appropriate amounts. The colour gamut for three points, such as D, E and F in Fig. 10.37, is a triangle with three colour points as vertices. The triangle DEF in Fig. 10.37, shows that three primaries can only generate colours inside or on the bounding edges of the triangle.

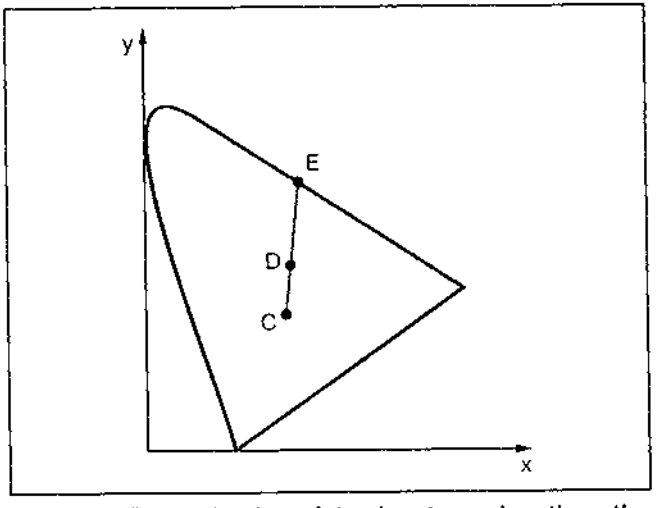


Fig. 10.38 Determination of dominant wavelength on the chromaticity diagram

The chromaticity diagram is also useful to determine the dominant wavelength of a colour. For colour point D in the Fig. 10.38, we can draw a straight line from C through D to intersect the spectral curve at point E. The colour D can then be represented as a combination of white light C and the spectral colour E. Thus, the dominant wavelength of D is E. This method for determining dominant wavelength will not work for colour points that are between C and the purple line because the purple line is not a part of spectrum.

10.10.3 RGB Colour Model

The red, green and blue (RGB) colour model used in colour CRT monitors and colour raster graphics employs a Cartesian coordinate system. In this model, the individual contribution of red, green and blue are added together to get the resultant colour.

We can represent this colour model with the unit cube defined on R, G, and B axes, as shown in the Fig. 10.39 (See Fig. 10.39 on next page).

The vertex of the cube on the axes represent the primary colours, and the remaining vertices represent the complementary colour for each of the primary colours. The main diagonal of the cube, with equal amounts of each primary represents the gray levels. The

end at the origin of the diagonal represents black (0, 0, 0) and other end represents white (1,1,1).

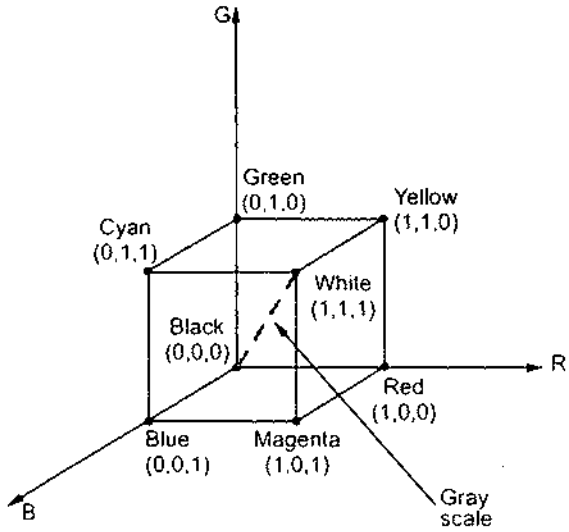


Fig. 10.39 The RGB cube

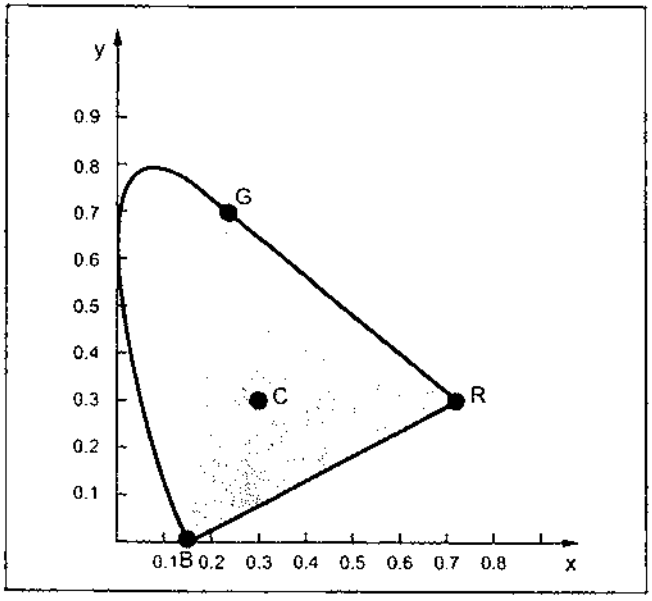


Fig. 10.40 Colour gamut for CIE standard RGB primaries

Each colour point within the bounds of the cube is represented as the triple (R, G, B), where value for R, G, B are assigned in the range from 0 to 1. As mentioned earlier, it is an additive model. Intensities of the primary colours are added to get the resultant colour. Thus, the resultant colour C_λ is expressed in RGB component as

$$C_\lambda = RR + GG + BB$$

The RGB chromaticity coordinates for the CIE RGB colour model as given as R (0.735, 0.265), G (0.274, 0.717), B (0.167, 0.009). The Fig. 10.40 shows the colour gamut for the CIE standard RGB primaries.

10.10.4 CMY Colour Model

In this model cyan, magenta and yellow colours are used as a primary colours. This model is used for describing colour output to hard-copy devices. Unlike video monitor, which produce a colour pattern by combining light from the screen phosphors, hard-copy devices such as plotters produce a colour picture by coating a paper with colour pigments.

The subset of the Cartesian coordinate system for the CMY model is the same as that for RGB except that white (full light) instead of black (no light) is at the origin. Colours are specified by what is removed or subtracted from white light, rather than by what is added to blackness. We know that, cyan can be formed by adding green and blue light. Therefore, when white light is reflected from cyan coloured ink, the reflected light does not have red component. That is, red light is absorbed or subtracted, by the ink. Similarly, magenta ink subtracts the green component from incident light, and yellow subtracts the blue component. Therefore, cyan, magenta, and yellow are said to be complements of red, green and blue respectively.

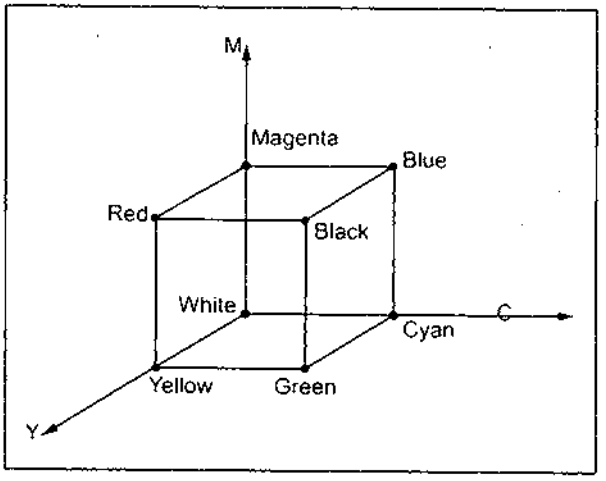


Fig. 10.41 The CMY cube

The Fig. 10.41 shows the cube representation for CMY model

As shown in the Fig. 10.41, point (1, 1, 1) represents black, because all components of the incident light are subtracted. The point (0, 0, 0), the origin represents white light. The main diagonal represents equal amount of primary colours, thus the gray colours. A combination of cyan and yellow produces green light, because the red and blue components of the incident light are absorbed. Other colour combinations are obtained by a similar subtractive process.

It is possible to get CMY representation from RGB representation as follows

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The unit column vector is the RGB representation for white and the CMY representation for black. The conversion from RGB to CMY is then can be given as

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

10.10.5 HSV Colour Model

We know that RGB and CMY models which we have seen are hardware oriented model. In contrast, HSV colour model is user oriented. It uses colour descriptions that have a more intuitive appeal to a user. The colour specification in HSV model can be given by selecting a spectral colour and the amounts of white and black that are to be added to obtain different shades, tints, and tones. This model uses three colour parameter : hue (H), saturation (S), and value (V). Hue distinguishes among colours such as red, green, purple and yellow. Saturation refers to how far colour is from a gray of equal intensity. For example, red is

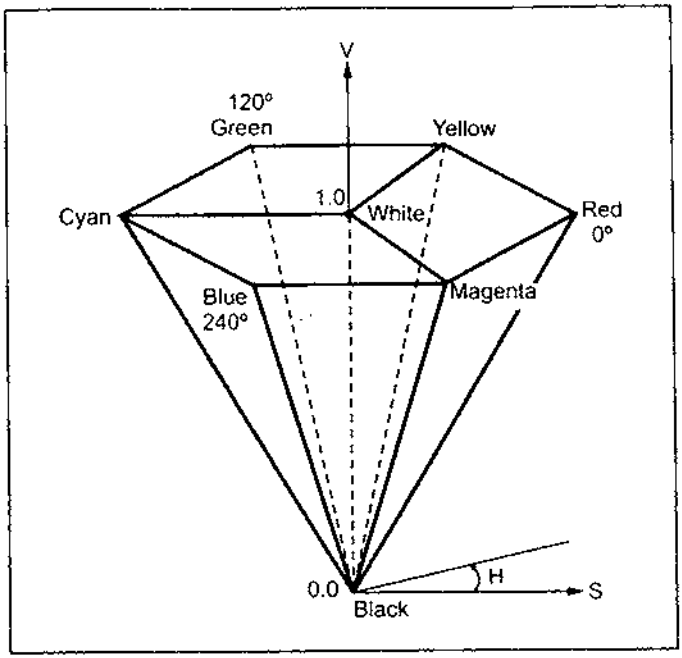


Fig. 10.42 The HSV hexcone

highly saturated whereas pink is relatively unsaturated. The value V indicates the level of brightness.

This model uses cylindrical coordinate system, and the subset of the space within which model is defined is a hexcone, or six-sided pyramid, as shown in the Fig. 10.42.

The top of the hexcone is derived from the RGB cube. If we imagine viewing the cube along the main diagonal from the white vertex to the origin (black), we see an outline of the cube that has the hexagon shape shown in Fig. 10.43. This boundary of cube is used as a top if hexcone and it represents various hues.

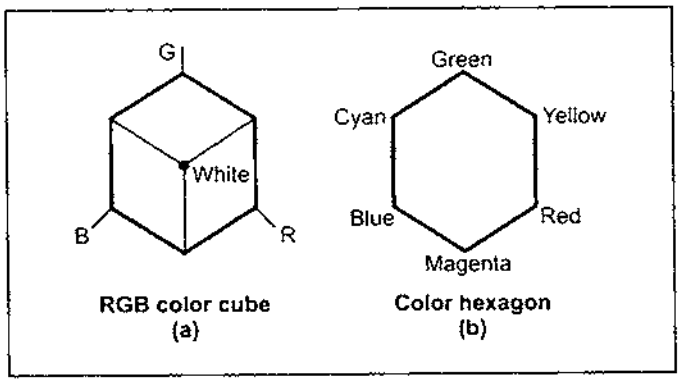


Fig. 10.43 Top of hexcone

Hue, or H, is measured by the angle around the vertical axis, with red at 0°, green at 120° and so on as shown in the Fig. 10.42. Complementary colours in the HSV hexcone are 180° apart saturation parameter varies from 0 to 1. Its value is the ratio ranging from 0 on the center line (V axis) to 1 on the triangular sides of the hexcone. The value V varies from 0 at the apex of the hexcone to 1 at the top. The apex represents black. At the top of the hexcone, colours have their maximum intensity. When V = 1 and S = 1, we have the pure hues. For example, pure red is at H = 0, V = 1 and S = 1, pure green is at H = 120, V = 1 and S = 1, pure blue is at H = 240, V = 1 and S = 1 and so on. The required colour can be obtained by adding either white or black to the pure hue. Black can be added to the selected hue by decreasing the setting for V while S is held constant. On the other hand, white can be added to the selected hue by decreasing S while keeping V constant. To add some black and some white we have to decrease both V and S. The point S = 0 and V = 1 we have white colour. The intermediate values of V for S = 0 (on the center line) are gray shades. Thus, when S = 0, the value of H is irrelevant. When S is not zero, H is relevant. At the apex V coordinate is 0. At this point, the values of H and S are irrelevant.

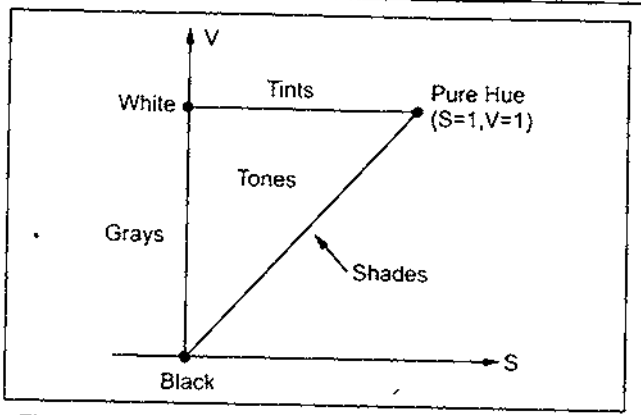


Fig. 10.44 Cross sectional plane of the HSV showing tints, tones and shades

Fig. 10.44 shows the cross sectional plane of the HSV hexcone. This plane represents the colour concepts associated with the terms shades, tints and tones. As shown in Fig. 10.44, we can add to

- black colour to pure hue to produce different shades of the colour.
- white colour to pure hue to produce different tints of the colour.
- both white and black colours to pure hue to produce tones of the colour.

10.10.6 HLS Colour Model

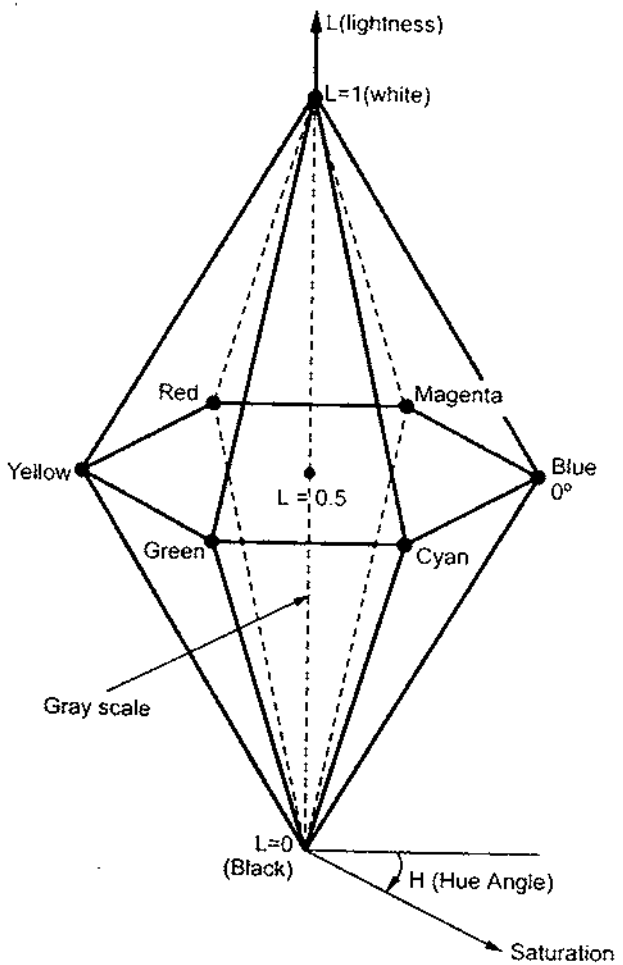


Fig. 10.45 Double-hexcone HLS colour model

Another model based on intuitive colour parameters is the HLS colour model used by Tektronix. The three colour parameters in this model are : hue (H), lightness (L) and saturation (S). It is represented by double hexcone, as shown in the Fig. 10.45.

The hue specifies the angle around the vertical axis of the double hexcone. In this model, H = 0° corresponds to blue. The remaining colours are specified around the perimeter of the hexcone in the same order as in the HSV model. Magneta is at 60°, red is at 120°, and yellow is located at H = 180°. Again, complementary colours are 180° apart on the double hexcone.

The vertical axis in this model represents the lightness, L. At L = 0, we have black and at L = 1, we have white. In between value of L we have gray levels. The saturation parameters S varies from 0 to 1 and it specifies relative purity of a colour.

At S = 0, we have the gray scale and at S = 1 and L = 0.5, we have maximum saturated (pure) hue. As S decrease, the hue saturation decreases i.e. hue becomes less pure.

In HLS model a hue can be selected by selecting hue angle H, and the desired shade, tint, or tone can be obtained by adjusting L and S. The colours can be made lighter by increasing L and can be made darker by decreasing L. The colours can be moved towards gray by decreasing S.

Review Questions

1. Define : diffuse illumination, diffuse reflection and coefficient of reflection.
2. Explain the Lambert's cosine law.
3. What is diffused reflection ? Give the illumination model that incorporate this reflection.
4. What is specular reflection ? Give the illumination model that incorporate this reflection.
5. Derive the illumination model with combine diffuse and specular reflections.
6. Describe the Phong's illumination model.
7. What is halfway vector ? Where it is used ?
8. Explain constant intensity shading algorithm.
9. Explain Gouraud shading algorithm ? Discuss its advantages and disadvantages.
10. Explain Phong shading model. Give its merits and demerits.
11. What is halftoning ? Explain the halftone shading.
12. What is a matchband effect ?
13. Compare Gouraud shading and Phong's shading.
14. Write a short note on transparency
15. What is refraction effect ?
16. Write a short note on shadows.
17. Write a note on visible surface ray tracing.
18. Discuss the properties of light.
19. Define chromaticity, complementary colours, colour gamut and primary colours.
21. Draw and explain the CIE chromaticity diagram.

22. Explain the usefulness of CIE chromaticity diagram with examples.
23. Explain RGB colour model.
24. Explain CMY colour model.
25. Explain HSV colour model.
26. Define hue, saturation and value.
27. Explain HLS colour model.

University Questions

1. Develop an illumination model to consider ambient light, specular and diffuse reflections. **(Dec-96, Dec-2000, May-2000, Dec-2001)**
2. Explain the Phong shading technique and compare this with the Gourand shading technique. **(Dec-96, Dec-2000)**
3. What do you mean by the illumination model ? Develop a suitable illumination model to consider specular and diffused reflection. **(May-97)**
4. Explain the Gourand shading method for shading. State its advantages and disadvantages. **(May-97)**
5. Write a detailed note on colour models **(May-97, Dec-97, May-2000)**
6. What is an illumination model ? Develop an illumination model to consider ambient light, specular reflection and diffused reflection. **(Dec-97)**
7. Explain the algorithm for Phong shading and Gaurand shading. List its advantages over the other. **(Dec-97)**
8. Write a detailed note on Halftoning and Dithering techniques. **(May-98, Dec-2000, May-2002)**
9. What is the purpose of illumination mode ? Give the classification of light sources. Whether reflection depends upon the surface characteristics, explain in short. **(May-98)**
10. Explain the method of Gouraud shading and Phong shading. State relative merits. **(Dec-98)**
11. How the pattern of black and white are used to give impression of intermediate intensities ? Explain the method. **(Dec-98)**
12. How is shading done? Describe one method. **(May-99)**
13. Explain the methods of speeding up Phong, shading technique. **(Dec-99)**
14. Compare and Contrast Gouraud shading technique with Phong shading technique. **(Dec-99, May-2003)**
15. State and explain the features of Phong illumination model. **(Dec-99)**
16. Describe Gourand shading technique in pseudocode. **(Dec-99)**
17. Explain Gourand Shading method. Discuss its advantages and disadvantages over the Phong technique. **(May-2000)**
18. Write a detailed note on colour representation **(Dec-2000)**
19. Explain phong illumination model in detail. **(May-2001)**
20. Explain advantages and disadvantages of Gourand shading, Phong shading and Ray tracing method. **(May-2001)**

21. Write a detailed note on RGB colour model. (May-2001, Dec-2001, May-2002)
22. Discuss the merits and demerits of constant shading method, gouraud shading method, phong shading method and ray casting method. (Dec-2001)
23. Write a short note on diffuse reflection ilumination model. (May-2002)
24. Illustrate HSV colour model and RGB colour model. (May-2003)
25. Write a short note on dithering techniques. (May-2003)

❑❑❑

Index

2D clipping 152

A

Aliasing 59
Angle of incidence 290
Angle of refraction 301
Antialiasing 59
Antiliasing of lines 59
Appel's algorithm 252
Area coherence 249
Area subdivision algorithm 258
Art and commerce 4
Aspect ratio 28
Axis vanishing point 229
Axonometric 228

B

Back-face culling 250
Back-face removal algorithm 261
Basic concepts in circle drawing 63
Basic concepts in line drawing 45
Beam-penetration technique 23
Bitmap 16
Blending function 268
BMP 9
Boundary colour 94
Boundary fill algorithm 93
Bounding volumes 250
Bresenham's circle drawing algorithm 67, 70
Bresenham's line algorithm 53, 54

C

Cabinet projections 228, 229
Cameras 36
Cartesian coordinate systems 37
Cartesian reference system 37, 43
Cartography 4
Cathode-ray-tube 12
Cavalier 228
Center of projection 227
Chain printers 30
Charge coupled devices 10
Chromaticity 313
CIE chromaticity diagram 313
Circle drawing 63
Circle drawing algorithms 65
Circular arc generation 265
Classification of applications 4
Clip window 152
Clipping volume 235
Clipping window 152
Clipping 147
CMY colour model 316
Coefficient of reflection 288
Cohen-Sutherland algorithm 173
Coherence properties 98
Coherence 249
Colour 312
Colour gamut 313
Colour video monitors 22
Combined diffuse and specular reflections 293
Complementary colours 313
Composition of 2D transformations 121
Computer-aided drafting and design 4
Concave polygon 89

Constant-intensity shading 294
Contour line 252
Convex polygon 88
Coordinate representations 40
Cyrus-Beck algorithm 167, 173
Cyrus-Beck line clipping algorithm 171

D

Data glove 8
DDA algorithm 265
DDA circle drawing algorithm 65
DDA line algorithm 48
Decision variable 53, 68
Deflection plates 13
Depth buffer 256
Depth coherence 249
Depth sort algorithm 253
Diffuse illumination 288
Diffuse reflection 288
Digital differential analyzer 47
Digitizier 8
Direction number 92
Direct-view storage tubes 24
Display file interpreter 21
Display file 18
Display processor 21
Dither noise 300
Dithering techniques 300
Dot matrix printers 30
Drum plotter 35
Drum printers 29
Drum scanners 10

E

Edge coherence 249
Edge-fill 93
Efficient visible-surface algorithms 249
Electro luminescent displays 25
Electron beam 12,13
Electrostatic deflection 12
Electrostatic plotter 35
Ellipse drawing algorithm 75
Emissive displays 25
Entering polygons 90
Error 53
Error diffusion 301
Even-odd method 91
Extents and bounding volumes 250
Exterior clipping 193

F

Face coherence 249
Faceted shading 294
Fill colour 94
Flat bed plotter 34
Flat panel displays 25
Flat shading 294
Flatbed scanners 10
Flood fill algorithm 95
Flood gun 24
Flood-fill algorithms 93
Frame buffer 16
Frame coherence 249

G

Generalized Bresenham's algorithm 57
Generalized clipping 167, 192
GIF 9
Gouraud shading 294
Graphical tablet 8

H

Halftone shading 299

Halftones 299
Halftoning 299
Halfway vector 293
Haloed lines 253
Handheld scanners 10
Hardcopy devices 28
Hidden line elimination algorithms 251
Hidden line 248
Hidden line/surface algorithms 248
Hidden surface algorithms 248
Hidden surface elimination algorithms 253
Hierarchy 251
HLS colour model 319
Homogeneous coordinate system 115
Homogeneous coordinates for rotation 116
Homogeneous coordinates for scaling 117
Homogeneous coordinates for translation 116
Homogeneous coordinates 115
HSV colour model 317
Hue 312

I

Illumination model 288
Image processing as picture analysis 2
Image-space methods 248
Impact printers 29
Implied edge coherence 249
Increasing resolution 59
Incremental algorithm 46
Index of refraction 301
Ink jet printer 31
Input devices 5
Inside test 91
Interior and exterior clipping 193
Interpolation 267
Inverse transformations 130

J

Jittering 310
Joysticks 7
JPEG 9

K

Keyboard buffer 6
Keyboard 5

L

Lagrange interpolation 269
Lambert's cosine law 288
Lambert's law 288
Laser printer 32
Liang-Barsky line clipping algorithm 173
Light detectors 10
Light pen 11
Lighting model 288
Line clipping 153
Line drawing 45
Line drawing algorithms 46
Line printers 29
Line-display algorithms 248
Liquid crystal display 25
Liquid crystal monitors 26

M

Mach bands 297
Methods of antialiasing 59
Midpoint circle drawing algorithm 71

Midpoint subdivision algorithm 162, 164
Motion dynamics 3
Mouse 6

N

Nonemissive displays 25
Non-impact printers 29
Normalization transformation 148, 149
Normalized device coordinates 148
Normal-vector interpolation shading 297

O

Object coherence 249
Object precision methods 248
Object-space methods 248
Oblique projection 228
Office automation and desktop publishing 4
On given view plane 232
On xy plane 231
Orientation dependent 50
Orthographic projection 228
Outcodes 153
Output devices 12
Outside 170

P

Painter's algorithm 253
PEL 16
Pen plotter 34
Persistence 27
Perspective projection 227
Perspective transformation 249
Phong illumination model 291
Phong shading 297
Phong shading technique 298
Phong specular reflection model 292
Photomultiplier tube 10
Photo-paint 9
Photo-shop 9
Physical coordinate system 248
Physical device coordinate system 147
Pixel position 16
Pixel 16
Pixmap 16
Plasma panel display 25
Plasma panels 25
Plotters 34
Plotting of graphics and chart 4
Point clipping 153
Point-source illumination 290
Polar coordinate system 39
Polygon clipping 178
Polygon filling 93
Polygon vertices 89
Polygon's shadow volume 303
Polyline 88
Polynomial method 64
Primary colours 313
Primary gun 24
Principle vanishing point 229
Printer buffer 30
Printers 29
Process control 4
Projection reference point 227
Projections 226
Properties of light 312
Pure 314

R

Random scan 14
Raster scan display 15, 16
Rasterization 2
Ray tracing tree 304

Ray-tracing 303
Reflection with respect to any plane 208
Reflection with respect to given plane208
Reflection with respect to xy plane 208
Reflection 123
Reflectivity 288
Refraction 301
Refresh buffer 14
Region codes 153
Representation of a circle 64
Representation of polygons 89
Representative uses of computer graphics 4
Resolution 27, 28
RGB colour model 315
Robert's algorithm 251
Rotation about an arbitrary point 121
Rotation about arbitrary axis 201
Rotation 111, 200

S

Scaling transformation 149
Scaling 113, 199
Scan code 6
Scan conversion 2, 22
Scan line algorithm 93,96, 255
Scan line coherence 249
Scanners 9
Screen coordinate system 248
Secondary ray 304
Seed fill 93
Serial printers 30
Shading algorithms 294
Shading model 288
Shadow mask grid 23
Shadow mask technique 23
Shadow ray 305
Shadow volumes 303
Shadows 302
Shear 125
Shearing relative to other reference line 126
Sheet-fed scanners 10
Shift vector 110
Simulation and animation 4
Spaceball 7
Space-subdivision method 308
Span coherence 249
Spatial partitioning 251
Specular reflection 291
Subdivision of a Bezier spline 276
Subject polygon 191
Sutherland - Hodgeman polygon clipping 179
Sutherland and Cohen subdivision line clipping algorithm 153, 157

T

The advantages of interactive graphics 3
The halfway vector 293
The phong illumination model 291
Thick line segments 61
Three - dimensional midpoint subdivision algorithm 239
Three dimensional clipping 235
Three dimensional viewing 221
Tiff 9
Touch panels 10
Trackball 7
Transformation from world coordinate to viewing coordinates 225
Transformation matrices for general parallel projection 231
Transformation matrix for oblique projection onto XY plane 233
Transformation matrix for perspective projection 234
Transformations 123

Translation vector 110
Translation 198
Transparency 301
Trigonometric method 64
Two dimensional transformations 110
Types of parallel projections 227
Types of perspective projections 229
Types of polygons 88

U

Unweighted area sampling 60
Update dynamics 3
User interfaces 4

V

Vanishing point 229
Vector scan/random scan display 14
Video display devices 12
View - distance 222
View reference point 222
View volume 235
Viewing angle 291
Viewing parameters 222
Viewing transformation 147, 148, 151
View-plane normal vector 222
Viewport 149
Visible surface algorithms 248
Voice systems 11

W

Warnock's algorithm 258
Weighted area sampling 60
Weiler-Atherton algorithm 191
Winding number 92
Winding-number method 91
Window 149
Windowing 147
Workstation transformation 149
World coordinate system 147

X

X shear 125

Y

Y shear 125

Z

Z-buffer algorithm 256

BDIMU
2009
Be Taken Back Only
The Syllabus &
Mumbai