COMPUTER GRAPHICS

BCA - 503

Dr. N.P. Singh
Dean

Directorate of Distance Education
ਪੰਜਾਬ ਟੈਕਨੀਕਲ ਯੂਨੀਵਰਸਿਟੀ ਜਲੰਧਰ
Punjab Technical University
Estd. Under Punjab Technical University Act, 1996
(Punjab Act No. 1 of 1997)

Preface

The Punjab Government established Punjab Technical University (PTU) in 1997 by an act of State Legislative. The University was entrusted with the responsibility of developing the new generation of technical manpower that can spearhead the industrial development of the State. Punjab Technical University has been envisaged to be the grooming ground for the future Engineers, Managers and Researchers.

As of today, PTU affiliates more than 300 Engineering, Management, Pharmacy, Hotel Management and Architecture institutions in the State that are approved by All India Council of Technical Education (AICTE).

PTU understands that restricting technical education to its campuses will not serve its objective of effective spreading of knowledge in the society. It is firmly understood that latest technical education has to be spread to the masses in every corner of the nation. This is how the Distance Education Programme (DEP) of the Punjab Technical University was conceived.

The objectives of the programme are to impart affordable, relevant, skill-based & remunerative technical education to the masses in the different corner of the country.

Today, the University has more than 2000 Learning Centres spread across the country offering quality technical education in the fields of Information Technology and Management, Paramedical Technology, Fashion Technology, Hotel Management and Tourism, Media and Mass Communication and Journalism etc.

The main purpose of this book is to impart the student an insight into the subject, explaining the complexities involved, in a simplified manner and helping them to achieve their academic goals.

For an easier navigation and understanding, this book contains the complete PTU curriculum of this subject and the topics. The various topics are dividing into Chapters, Units & Sub-Units and sufficient space is provided for students to make their brief notes.

This book encompasses a global approach for providing the simplified study material to both working as well as non-working students and is certain to get benefitted from the efforts of the authors of this book.

Dr.N.P. Singh
Dean (Distance Education Programme)

"Propelling Punjab to a Prosperous Knowledge Society"

Punjab Technical University Jalandhar
Jalandhar-Kapurthala Highway, Near Science City, Kapurthala- 144601
Phones : 01822-662502 Fax : 01822-255532
Website : www.ptu.ac.in email : deandep@ptu.ac.in, singhnp59@gmail.com

COMPUTER GRAPHICS

BCA - 503

This SIM has been prepared exclusively under the guidance of Punjab Technical University (PTU) and reviewed by experts and approved by the concerned statutory Board of Studies (BOS). It conforms to the syllabi and contents as approved by the BOS of PTU.

Reviewer

Dr. N. Ch. S.N. Iyengar	Senior Professor, School of Computing Sciences, VIT University, Vellore

Authors: Anirban Mukhopadhyay & Arup Chattopadhyay

Reprint 2008, 2010

VIKAS® PUBLISHING HOUSE PVT LTD
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: 576, Masjid Road, Jangpura, New Delhi 110 014
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

CAREER OPPORTUNITIES

For the qualified BCA, job opportunities abound in the software development industry or in industries/areas which use IT like banking, insurance, manufacturing, entertainment, education and training. Presently there are huge requirements of application programmers, testing engineers, system analysts and domain experts with working knowledge in areas like C, C++, Java, OOP, S/w Engg., RDBMS and knowledge in specialized areas like Networking, Graphics & Multimedia, Data Mining, E-Commerce. Graphics and multimedia offer great prospects of growth in the future.

BCAs can specialize in Computer Graphics aiming at the following job opportunities in the market:

- Games or multimedia programmer
- Graphics programmer
- CAD programmers or designers
- Graphics technicians for analysis, design and production of multimedia tools or application
- Art directors or technical directors in publishing or web designing
- Research in potential application areas of Computer Vision, Image Processing, AI, etc.

These are a few application areas of Computer Graphics, which are in no way exhaustive, but only indicative of the trend.

PTU DEP SYLLABI-BOOK MAPPING TABLE

BCA - 503 Computer Graphics

Syllabi	Mapping in Book
Section-I	
Input Devices: Keyboard, Touch Panel, Light Pens, Graphic Tablets, Joysticks, Trackball, Data Glove, Digitizers, Image Scanner, Mouse, Voice & Systems.	**Unit 1: Input Devices** (Pages 3-10)
Hardcopy Devices: Impact and Non Impact Printers, Such as Line Printer, Dot Matrix, Laser, Ink-jet, Electrostatic, Flatbed and Drum Plotters.	**Unit 2: Hardcopy Devices** (Pages 11-16)
Section II	
Video Display Devices: Refresh Cathode-Ray Tube, Raster Scan Displays, Random Scan Displays, Color CRT-Monitors, Direct View Storage Tube, Flat-Panel Displays; 3-D Viewing Devices, Raster Scan Systems, Random Scan Systems, Graphics Monitors and Workstations.	**Unit 3: Display Devices** (Pages 17-42)
Scan Conversion Algorithms for Line, Circle and Ellipse, Bresenham's Algorithms, Area Filling Techniques, Character Generation.	**Unit 4: Scan Conversion Algorithms** (Pages 43-73)
Section III	
2-Dimensional Graphics: Cartesian and Homogeneous Co-ordinate System, Geometric Transformations (Translation, Scaling, Rotation, Reflection, Shearing), Two-dimensional Viewing Transformation and Clipping (Line, Polygon and Text).	**Unit 5: 2-Dimensional Graphics** (Pages 74-120)
3-Dimensional Graphics: Geometric Transformations (Translation, Scaling, Rotation, Reflection, Shearing), Mathematics of Projections (Parallel & Perspective). 3-D Viewing Transformations and Clipping.	**Unit 6: 3-Dimensional Graphics** (Pages 121-153)

CONTENTS

INTRODUCTION

NOTES

Computer graphics refers to the creation, storage and manipulation of pictures and drawings using a digital computer. With developments in computing technology interactive computer graphics has become an effective tool for the presentation of information in such diverse fields as science, engineering, medicine, business, industry, government, art, entertainment, advertising, education, and training. There is virtually no field in which graphical displays cannot be used to some advantage and that is the basic reason why application of computer graphics is so widespread.

It is a fact that one picture is worth a thousand words. Therefore, interfaces empowered with graphics enhances the communication between the computer and its users. Representation of a huge set of numbers in the form of a graph or a picture helps in a better understanding and interpretation of the characteristics or pattern of the data contained in the set of numbers. Graphic displays also improve understanding of complex systems, and visualization of two-dimensional (2D), three-dimensional (3D) objects.

A major application of computer graphics is designing, particularly engineering and architectural systems. Almost all consumer products are now computer-designed. Computer-aided design (CAD) techniques are now routinely used in the design of building, automobile, aircraft, watercraft, spacecraft, defense mechanism, computer-component, industrial machinery and device, textile and increasing number of other products ranging from a pen to a refrigerator. In a computer-aided design environment, traditional tools of design are replaced by parameterized modeling routines with interactive graphic capabilities that are so active and dynamic that the designer can carry out unlimited number of experiments involving complex computation in search for better design. Powerful digital graphic tools for molecular modeling have added a new dimension in the research of emerging fields like Biotechnology and Bioinformatics.

Among the other applications of computer graphics, Image Processing, Animation, Morphing, Simulation, e-Learning Material Designing and Graphic Designing are rapidly gaining demand and usage in education, training, advertisement and entertainment. Computer graphics has highly influenced the film industry with its multimedia applications. Controlled animation, simulation and morphing have increasingly been applied in the study of time-varying physical phenomena, object movement and operating sequences of machinery in scientific and industrial research. Computer-aided image processing and picture analysis are now indispensable tools for remote sensing, aerial survey, space research, pattern recognition, CT scans and research in medical science.

UNIT 1 INPUT DEVICES

Structure

NOTES

1.0 INTRODUCTION

Graphic hardware can be divided into three major categories of devices: (1) I*nput devices* with which the user interacts to generate necessary instruction or data for creating graphics (2) *Display systems* where the graphics are rendered on the monitor screen (3) *Hardcopy devices* or printers through which the tangible graphics output is produced.

Based on the logical interaction types the input devices can be broadly classified as – (1) *locator device* such as graphics tablet, touch panel, mouse, trackball, joystick, keyboard that indicates a position (e.g., point coordinate) or orientation, (2) *pick device* such as the light pen, joystick, mouse that can select a graphical object, (3) *valuator device* such as joystick or trackball that are used to input scalar values such as rotation angle, scale factors etc. (4) *keyboard* or text input device and (5) *choice device* such as the keyboard function keys, mouse, touch panel, voice systems that are used to select menu options. This unit deals exclusively with various input devices with different functional capabilities.

1.1 UNIT OBJECTIVES

- Understanding the basic functional characteristics and working of input devices
- Describing the basic architecture and typical varieties of input devices
- Highlighting the usual application area of each device

1.2 INPUT DEVICES: AN OVERVIEW

Various devices are available for data input ranging from general purpose computer systems with graphic capabilities to sophisticated workstations designed for graphics applications. Among these devices are graphic tablets, light pens, joysticks, touch panels, data gloves, image scanner, trackballs, digitizer, voice systems and of course the common alphanumeric keyboard and mouse. Given below are the basic functional characteristics and applications of these devices.

1.2.1 Keyboard

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

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

NOTES

- Function keys (F1 to F12)
- Control keys (Ctrl, Alt, Del, Pg Up, Pg Dn, Home, End, Esc, , Fn, arrow keys etc.)

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

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

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

Figure 1.1: *Microsoft Wireless Keyboard*

1.2.2 Mouse

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

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

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

NOTES

Figure 1.2: *Microsoft's Two-Button Wireless Mouse*

1.2.3 Trackball

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

Figure 1.3: *A Logitech Trackball*

1.2.4 Joystick

A joystick is used as a personal computer peripheral or general control device consisting of a hand-held stick that pivots about the base and steers the screen cursor around. Most joysticks are two-dimensional, having two axes of movement (similar to a mouse), but three-dimensional joysticks do exist. A joystick is generally configured so that moving the stick left or right signals movement along the X-axis, and moving it forward (up) or back (down) signals movement along the Y-axis. In joysticks that are configured for three-

NOTES

dimensional movement, twisting the stick left (counter-clockwise) or right (clockwise) signals movement along the Z-axis. In conventional joystick potentiometers, or variable resistors, are used to dynamically detect the location of the stick and springs are there to return the stick to centre position as it is released.

In many joysticks, optical sensors are used instead of analog potentiometer to read stick movement digitally. One of the biggest additions to the world of joysticks is force feedback technology. On using a force feedback (also called haptic feedback) joystick if you're shooting a machine gun in an action game, the stick would vibrate in your hands. Or if you crashed your plane in a flight simulator, the stick would push back suddenly which means the stick moves in conjunction with onscreen actions.

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

Figure 1.4: *The Flighterstick, a Modern Programmable USB Joystick*

1.2.5 Digitizer and Graphics Tablet

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

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

and price ranges—A6-sized tablets being relatively inexpensive and A3-sized tablets being far more expensive.

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

NOTES

Figure 1.5: *A Tablet with Hand Cursor*

1.2.6 Touch Panel

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

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

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

NOTES

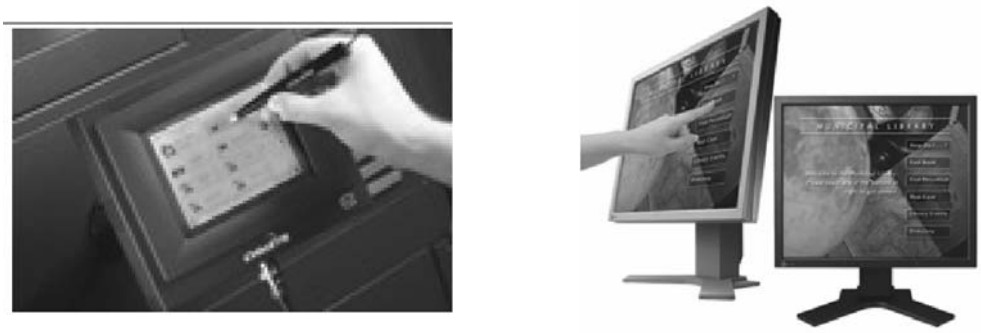


Figure 1.6: *Touch Panels*

1.2.7 Light Pen

A light pen is a pointing device shaped like a pen and is connected to the computer. The tip of the light pen contains a light-sensitive element (photoelectric cell) which, when placed against the screen, detects the light from the screen enabling the computer to identify the location of the pen on the screen. It allows the user to point to displayed objects, or draw on the screen, in a similar way to a touch screen but with greater positional accuracy. A light pen can work with any CRT-based monitor, but not with LCD screens, projectors or other display devices.

The light pen actually works by sensing the sudden small change in brightness of a point on the screen when the electron gun refreshes that spot. By noting exactly where the scanning has reached at that moment, the x, y position of the pen can be resolved. The pen position is updated on every refresh of the screen.

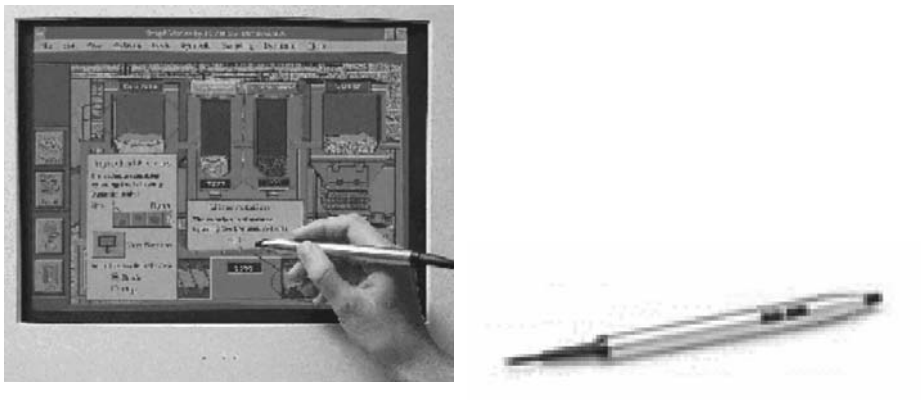


Figure 1.7: *Light Pen*

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

1.2.8 Data Glove

The data glove is an interface device that uses position tracking sensors and fiber optic strands that run down each finger and are connected to a compatible computer; the movement of the hand and fingers are displayed live on the computer monitor which in turn allows the user to virtually touch an object displayed on the same monitor. With the object animated it would appear that the user (wearing the data glove) can pick up an object and do things with it just as he would do with a real object. In modern data glove devices, tactile sensors are used to provide the user with an additional feeling of touch or the amount of pressure or force the fingers or hands are exerting even though the user is not actually touching anything. Thus data glove is an agent to transport the user to virtual reality.

Check Your Progress

1. The output of a scanner is
   (a) bitmap file
   (b) OCR document
   (c) graphics metafile
   (d) postscript file
2. Which of the following components are common for mouse, trackball and joystick?
   (a) capacitor
   (b) potentiometer
   (c) optical sensor
   (d) ball
3. A keyboard can be called a
   (a) locator device
   (b) valuator device
   (c) string device
   (d) all the above

NOTES

Figure 1.8: *Data Glove*

1.2.9 Voice-System

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

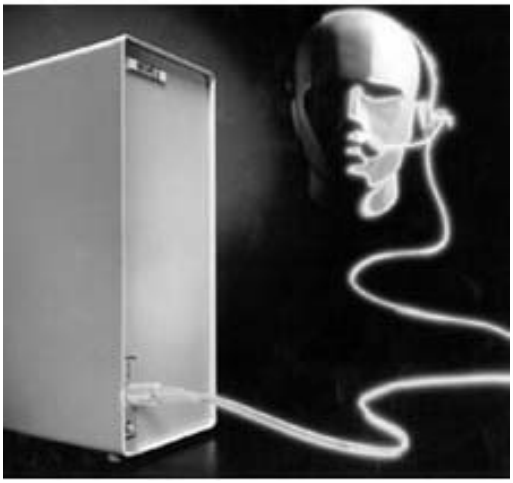


Figure 1.9: *Operator's Speech Recording*

1.3 SUMMARY

This unit introduces the forms and functions of input devices which are used as computer peripherals based on the requirement of specific computer graphics applications. An application program can make simultaneous use of several input devices operating in different modes. For example, a keyboard, a mouse, a scanner and a voice system can all be attached to the same computer interacting through the same or different graphical user interfaces. Many input devices are multifunctional and so more than one device can provide the same class of input data. Still the popularity of input devices varies based on the functional capabilities and users convenience. *Continuous* devices like tablets, joysticks and mouse

allow more natural, easier and faster cursor movements than do *discrete* devices like cursor control keys of the keyboard. However, devices like data glove, webcam and bar-code reader are popular in specialized applications for their unique functionality.

NOTES

1.4 ANSWERS TO 'CHECK YOUR PROGRESS'

1. (a)
2. (b)
3. (d)

1.5 EXERCISES AND QUESTIONS

1. Describe the touch-sensing mechanism as used in a touch panel.
2. What are the advantages of feedback in graphical input techniques?
3. What is position interaction task? Briefly discuss the main issues.
4. Enumerate the differences between pointing devices and positioning devices.
5. Explain how users can interact with a virtual screen. List the typical inputs that are used with VR systems.
6. Compare the working device used in touch panel and light pen.
7. Compare the working of a locator device and valuator device.

1.6 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

UNIT 2 HARDCOPY DEVICES

Structure

NOTES

2.0 INTRODUCTION

There are two broad categories of hardcopy devices — one is the *printer* and the other is the *plotter.* Though plotters have limited and specialized uses, printer is a common yet important accessory of any computer system, specially for a graphics system. Most of the computer graphics creation have their ultimate utilizations in printed/plotted forms that are used for design documentation, exhibition or publication in books or other print media. So it is the quality of printed/plotted output that makes computer graphics applications appealing to the businesses it caters to. In keeping with its importance in real life there have been close competitions amongst the manufacturers in developing newer and cheaper models of hardcopy devices ranging from low cost dot matrix and popular deskjets to heavy duty laserjets or sophisticated pen plotters. This unit describes various hardcopy technologies and functional aspects of a variety of printers and plotters.

2.1 UNIT OBJECTIVES

- Understanding basic printing technologies and major categories of hardcopy devices with reference to factors that affect the print/plot quality
- Describing the structural and functional aspects of each hardcopy device
- Highlighting the usual application area of each device
- Understanding how a computer communicates with state-of-the-art printers

2.2 HARDCOPY DEVICES: AN OVERVIEW

2.2.1 Printer

The printer is an important accessory of any computer system, specially for a graphics system. This is because most of the graphics creation using computer graphics has its ultimate utilization in printed form – for documentation, exhibition or publication in print media or books. It is the quality of printed output that finally matters in many businesses.

Based on the available printing technology the major factors which control the quality of printer are individual dot size on the paper and number of dots per inch (dpi). Clearly, the lesser the size of the dots the better the detail of the figure reproduced. Higher dpi values increase the sharpness and detail of a figure and enhance the intensity levels that a printer supports. Other important factors for selection of a printer are printing speed and print area or printer memory.

NOTES

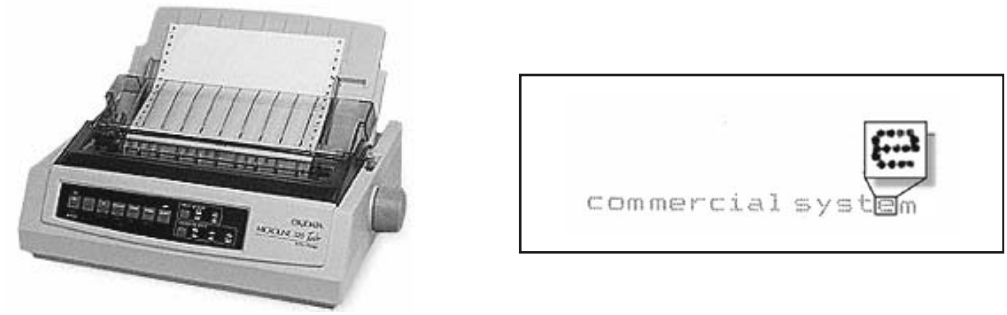
2.2.2 Impact vs. Non-impact

There are several major printer technologies available. These technologies can be broken down into two main categories with several types in each:

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

2.2.3 Dot Matrix Printer

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



Dot Matrix Printer *Typical Output from Dot Matrix Printer*

Figure 2.1

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

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

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

2.2.4 Line Printer

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

In a typical design, a fixed font character set is engraved onto the periphery of a number of print wheels, the number matching the number of columns (letters in a line). The wheels spin at high speed and paper and an inked ribbon are moved past the print

NOTES

position. As the desired character for each column passes the print position, a hammer strikes the paper and ribbon causing the desired character to be recorded on the continuous paper. Printed type is set at fixed positions and a line could consist of any number of character positions with 132 columns as the most common, but 80 column, 128 column and 160 column variants are also in use. Other variations of line printer have the type on moving bars or a horizontal spinning chain.

The line printer technology is usually both faster and less expensive (in total ownership) than laser printers. It has its use in medium volume accounting and other large business applications, where print volume and speed is a priority over quality. Because of the limited character set engraved on the wheels and the fixed spacing of type, this technology was never useful for material of high readability such as books or newspapers.

Figure 2.2: *Line Matrix Printer*

2.2.5 Inkjet Printer

An inkjet printer is a non-impact printer that places extremely small droplets of ink onto the paper to create an image. These printers are popular because they less costly but generate attractive graphic output.

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

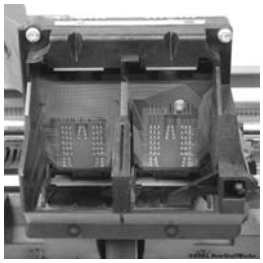
The core of an inkjet printer is the print head that contains a series of nozzles that are used to spray drops of ink. The ink is contained in ink cartridges that come in various combinations, such as separate black and colour cartridges, or a cartridge for each ink colour. A stepper motor moves the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

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

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

NOTES

a tiny amount of ink out of the nozzle. When it vibrates out, it pulls some more ink into the reservoir to replace the ink sprayed out.



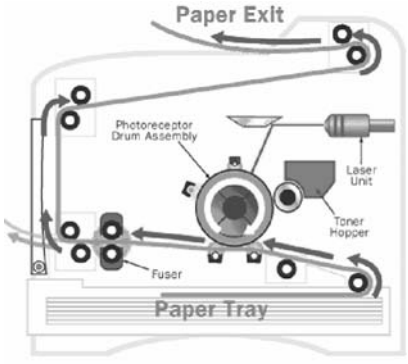
Canon Inkjet Printer *Print Head Assembly*

Figure 2.3

2.2.6 Laser Printer

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

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



HP Laser Printer *The Path of Paper Inside a Laser Printer*

Figure 2.4

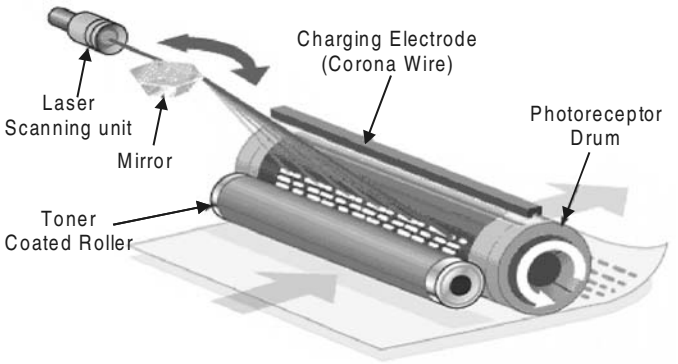


Figure 2.5: *Image Creation by the Laser on Drum*

2.2.7 Electrostatic Printer

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

The printer creates colour prints by breaking colour data down into three basic colours (cyan, magenta, and yellow) plus black, and printing one colour at a time. In *5-pass* print mode, combinations of cyan, magenta, yellow and black provide a wide range of different colours. Using the registration marks printed during the preliminary registration pass ensures that the colour plot is beautiful with no misalignment.

Figure 2.6: *HP Electrostatic Printer*

2.2.8 Plotter

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

NOTES

Check Your Progress

1. Which of the following is not a printer language?
   (a) PCL
   (b) Postscript
   (c) GDI
   (d) HPGL
2. dpi is the measure of print quality for a
   (a) dot matrix printer
   (b) dot matrix printer and inkjet printer
   (c) laser printer
   (d) dot matrix printer and pen plotter
3. For which of the following pair of hardcopy devices is the nature of ink or printing material not identical?
   (a) Dot matrix printer and line printer
   (b) Laser printer and electrostatic printer
   (c) Inkjet printer and wet-ink pen plotter
   (d) Flatbed plotter and drum plotter

Figure 2.7: *Flatbed Plotter and Drum Plotter*

2.3 SUMMARY

This unit introduces the forms and functions of common hardcopy devices. The technology used by these devices ultimately constrains the quality of realistic graphics. So the knowledge of printing technology and printer or plotter capabilities is needed to make an optimized use of resources and generate output of acceptable accuracy w.r.t. the end use. In addition to the printers discussed in this unit there are costlier varieties as dye-sublimation printer, thermal wax printer and thermal autochrome printer. The autochrome printers have three layers (cyan, magenta and yellow) that colour in the paper instead of in the printer. Most interestingly some thermal printer accepts video signals thus creating hardcopy of video images. Looking at the future of printer the one which is very much in the line of evolution is the Bluetooth-enabled wireless printer.

2.4 ANSWERS TO 'CHECK YOUR PROGRESS'

1. (c) 2. (b) 3. (b)

2.5 EXERCISES AND QUESTIONS

1. Compare the working of a digitizer with that of a plotter.
2. Explain the laser printer with reference to the following points: printing mechanism, speed, printing quality and application.
3. Compare the working principles of the electrostatic printer with that of a laser printer.
4. For an electrostatic printer with an 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?
5. Describe briefly how a computer communicates with a laser printer.

2.6 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

UNIT 3 DISPLAY DEVICES

Structure

NOTES

3.0 INTRODUCTION

The display medium for computer graphic-generated pictures has become widely diversified. Typical examples are CRT-based display, Liquid Crystal, LED and Plasma-based display and stereoscopic display. CRT display is by far the most common display technology and most of the fundamental display concepts are embodied in CRT technology. This unit focuses on CRT-based display technologies explaining the related concepts followed by illustrations of structural and functional components and working principles of each. The unit briefly explains few other common display technologies.

3.1 UNIT OBJECTIVES

- Understanding the basic concepts and parameters related to an image and physical display screen
- Understanding the basic display technology employed in raster scan CRT with reference to the architecture of a CRT
- Understanding the theory of colour display with reference to graphics memory and CRT circuitry
- Outlining the display systems and display technologies

3.2 VIDEO

Just like text, audio and still image digital videos are also a powerful elements of multimedia systems. To understand how digital video is used as a media we need to understand some fundamental aspects of analog video technology.

Basically video or motion pictures are created by displaying images depicting progressive stages of motion at a rate fast enough so that the projection of individual images overlap on the eye. *Persistence of vision* of human eye, which allows any projected image to persist for 40–50 ms, requires a frame rate of 25–30 frames per second to ensure perception of smooth motion picture.

NOTES

In a video display:

- *Horizontal resolution* is the number of distinct vertical lines that can be produced in a frame.
- *Vertical resolution* is the number of horizontal scan lines in a frame.
- *Aspect ratio* is the width-to-height ratio of a frame.
- *Interface*† *ratio* is the ratio of the frame rate to the field rate.

Constitution-wise there are three types of video signals: *Component video, Composite video* and *S-video*. Most computer systems and high-end video systems use component video whereby three signals R, G and B are transmitted through three separate wires corresponding to red, green and blue image planes respectively.

However, because of the complexities of transmitting the three signals of component video in exact synchronism and relationship these signals are encoded using a frequency-interleaving scheme into a composite format that can be transmitted through a single cable. Such format known as composite video, used by most video systems and broadcast TV, uses one luminance and two chrominance signals. *Luminance* (Y) is a monochrome video signal that controls only the brightness of an image. *Chrominance* is actually two signals (I and Q or U and V), called colour differences (B–Y, R–Y) and contains colour information of an image. Each chrominance component is allocated half as much band width as the luminance, a form of analog data compression*, which is justified by the fact that human eyes are less sensitive to variations in colour than to variations in brightness. Theoretically, there are infinite possible combinations (additive) of R, G, B signals to produce Y, I, Q or Y, U, V signals. The common CCIR 601 standard defines —

Luminance (Y) = 0.299R + 0.587 G + 0.114B

Chrominance (U) = 0.596R – 0.247 G – 0.322B

Chrominance (V) = 0.211R – 0.523G + 0.312B

The inverse of the above transformation formula gives

Red (R) = 1.0 Y + 0.956 U + 0.621 V

Green (G) = 1.0 Y − 0.272 U − 0.647 V

Blue (B) = 1.0 Y − 1.061 U − 1.703 V

Unlike composite video, S-video (separated video or super video as S –VHS) uses two wires, one for luminance and another for a composite chrominance signal. Component video gives the best output since there is no cross-talk or interference between the different channels unlike composite video or S-video.

In a television transmission system, every part of every moving image is converted into analog electronic signals and is transmitted. The VCR can store TV signal on magnetic tapes, which can be played to reproduce stored images. There are three main standards for analog video signals used in television transmission: NTSC, SECAM and PAL. A characteristic comparison of these standards is listed in the Table 3.1.

NOTES

Table 3.1: *Comparison of Analog Video Signals used in TV transmission*

Standard	Used in	Vertical Resolution	Frame Rate	Hori-zontal Resolution	Interface Ratio	Aspect Ratio	Channel Band-width
NTSC (National Television Standards Committee	USA, Canada, Japan, Korea	525 lines per frame	30 frames per second	340 lines	2:1	4:3	4.2 MH
SECAM (Systeme Electro-nique Couleur Avec Memoire)	France and Eastern Europe	625 lines per frame	25 frames per second	409 lines	2:1	4:3	6.0 MH
PAL (Phase Alternate Line)	Western Europe (except France), The United Kingdom, South America, Australia and parts of Asia	625 lines per frame	25 frames per second	409 lines	2:1	4:3	5.5 MHz in UK 5.0 MHz elsewhere

3.2.1 Digital Video

For video to be played and processed in computers it needs to be converted from analog to digital representation. Video digitization is achieved just like audio digitization by *sampling* the analog video signal at a preset frequency and subsequently *quantizing* the discrete samples in digital format. This is done with the help of analog to digital converter or ADC.

There are two kinds of possible digitizations or digital coding–*Composite coding* and *Component coding*. In composite coding, all signal components taken together as a whole are converted into digital format. In component coding, each signal component is digitized separately using different sampling frequency (13.5 MHz for luminance, 6.75 MHz for chrominance).

Based on whether the ADC is inside a digital camera, or in an external unit or inside the computer there can be three types of digital video systems – *Digital Camera-based System*, *External ADC System* and *Video Frame Grabber Card-based System.*

The main function of the video *frame grabber card* is to take the composite (luminance-chrominance) analog video signal, decode it to RGB signal, then convert it to the digital format and store each frame first in the frame buffer on the card itself. At an adequate frame-rate consecutive frames are streamed to the monitor, routed through the frame buffer of the main memory to present live video on the computer screen. The Sun video digitizer from Sun Microsystems captures NTSC video signal (RGB) with frame resolution of 320×240 pixels, quantization of 8 bits/pixels and a frame rate of 30 frames per second. How closely the digital video approximates the original analog video depends on the sampling resolution or number of bits used to represent any pixel value.

NOTES

Advantages of Digital Video

1. Storing video on digital devices memory ready to be processed, (noise removal, cut and paste, size and motion control and so on) and integrated into various multimedia applications is possible.
2. It allows direct access, which makes non-linear video editing (audio mixing, adding text, titles and digital effects etc.) simple.
3. It allows repeated recording without degradation of image quality.
4. Ease of encryption and better tolerance to channel noise is possible.

3.2.2 Digital Video Standards

To improve the picture quality and transmission efficiency new generation televisions systems are designed based on international standards that exploit the advantage of digital signal processing. These standards include *High Definition Television* or *HDTV*, *Improved Definition Television* or *IDTV*, *Double Multiplexed Analog Components* or *D2-MAC*, *Advanced Compatible Television, First System* or *ACTV-I*. HDTV standard that support progressive (non-interlaced) video scanning; has much wider aspect ratio (16:9 instead of 4:3); greater field of view; higher horizontal and vertical resolution (9600 and 675 respectively in USA) and more bandwidth (9 MHz in USA) as compared to conventional colour TV systems.

3.2.3 Video Compression

Of the different multimedia elements the need for compression is greatest for video as the data volume for Full Screen Full Motion (FSFM) video is very high. Frame size for NTSC video is 640 pixels × 480 pixels and if we use 24 bits colour depth then each frame occupies $640 \times 480 \times 3$ bytes, i.e., 900 KB. So each second of NTSC video comprising 30 frames occupies 900 30 KB which is around 26 MB and each minute occupies $26 \times 60 \approx 1.9$ GB. Thus a 600 MB CD would contain maximum 22 seconds of FSFM video. Now imagine the storage space required for a 2-hour movie. So the only way to achieve digital motion video on PC is to reduce or compress the redundant data in video files.

Redundancy in digital video occurs when the same information is transmitted more than once. Primarily in any area of an image frame where same colour or intensity spans more than one pixel location, there is *spatial redundancy*.

Secondly, when a scene is stationary or only slightly moving, there is redundancy between frames of motion sequence – the contents of consecutive frames in time are similar, or they may be related by a simple translation function. This kind of redundancy is called *temporal redundancy*.

Spatial redundancy is removed by compressing each individual image frame in isolation and the techniques used are generally called *spatial compression* or *intra-frame compression.* Temporal redundancy is removed by storing only the differences of subsequence of frames instead of compressing each frame independently and the technique is known as *temporal compression* or *inter-frame compression*.

Spatial compression applies different lossless and lossy method same as those applied for still images. Some of these methods are:

(i) Truncation of least significant image data
(ii) Rum Length Encoding or RLE
(iii) Interpolative techniques
(iv) Predictive technique – DPCM (Differential Pulse Code Modulation), ADPCM (Adaptive DPCM)
(v) Transform Coding Techniques – DCT (Discrete Cosine Transform)
(vi) Statistical or Entropy Coding – Huffman Coding, LZW coding, Arithmetic coding

NOTES

The most simplistic approach for temporal compression is to perform a pixel-by-pixel comparison (subtraction) between two consecutive frames. The compare should produce zero for pixels, which have not changed, and non-zero for pixels, which are involved in motion. Only then can the pixels with non-zero differences be coded and stored, thus reducing the burden of storing all the pixel value of a frame. But there are certain problems with this approach. Firstly, the even if there is no object motion in a frame, slightest movement of camera would produce non-zero difference of all or most pixels. Secondly, quantization noise would yield non-zero difference of stationary pixels.

In an alternative approach the motion generators camera and/or object can be 'compensated' by detecting the displacements (*motion vectors*) of corresponding pixel blocks or regions in the frames and measuring the differences of their content (*prediction error*). Such approach of temporal compression is said to be based on *motion compensation*. For efficiency each image is divided into *macroblocks* of size $N \times N$. The current image frame is referred to as *target frame*. Each macroblock of the target frame is examined with reference to the most similar macroblocks in previous and /or next frame called *reference frame*. This examination is known as *forward prediction* or *backward prediction* depending on whether the reference frame is a previous frame or next frame. If the target macroblock is found to contain no motion, a code is sent to the decompressor to leave the block the way it was in the reference frame. If the block does have motion the motion vector and difference block need to be coded so that the decompressor can reproduce the target block from the code.

3.2.4 MPEG

MPEG is the international standard for audio/video digital compression and MPEG-1 is most relevant for video at low data rate (up to 1.5 M bit/s) to be incorporated in multimedia. MPEG-1 is a standard in 5 parts, namely – Systems, Video and Audio, Conformance Testing and Software Simulation (a full C-language implementation of the MPEG-1 encoder and decoder). Though higher standards like MPEG-2, MPEG-4, MPEG-7 and MPEG-21 have evolved in search of higher compression ratio, better video quality, effective communication and technological upgradation, we will discuss MPEG-1 only for the understanding of basic MPEG scheme.

3.2.5 MPEG-1

MPEG-1 standard doesn't actually define a compression algorithm; it defines a datastream syntax and a decompressor. The datastream architecture is based on a sequence of frame, each of which contains the data needed to create a single displayed image. There are four different kind of frames, depending on how each image is to be decoded:

I-frames (Intra-coded images) are self-contained, i.e., coded without any reference to other images. These frames are purely spatially compressed using a transform coding method similar to JPEG. The compression ratio for I-frames is the lowest within MPEG. An I-frame must exist at the start of any video stream and also at any random access entry point in the stream.

P-frames (Predictive-coded images) are compressed images resulting from removal of temporal redundancy between successive frames. These frames are coded by a forward predictive coding method in which target macroblocks are predicted from most similar reference macroblocks in the preceding I or P-frame. Only the difference between the spatial location of the macroblocks, i.e., the motion vector and the difference in content of the macroblocks are coded. Instead of the difference macroblocks itself is coded as non-motion compensated macroblock when a good match as reference macroblocks is not found. Usually in P-frames large compression ratio (three times as much in I-frames) is achieved.

NOTES

B–frames (Bi-directionally predictive-coded frames) are coded by interpolating between two macroblocks – one from forward prediction (from previous I or P-frame) and the other from backward prediction (from future I or P-frame). Interpolative motion compensation is used here. If matching in both directions is successful, two motion vectors will be sent, and the two corresponding matching macroblocks will be averaged (interpolated) for comparing to the target macroblock for generating the difference macroblock. If an acceptable match can be found in only one of the reference frames, then only one motion vector and its corresponding macroblocks is used for generating the difference macroblock. Maximum compression ratio (one and half times as much as in P-frame) is achieved in B-frames.

D-frames (DC-coded frames) are intraframe coded and are used for fast forward or fast rewind modes.

Macroblocks which are basic frame elements for predictive encoding are partitioned into 16×16 pixels for luminance component and 8×8 pixels for each of the chrominance components. Hence for 4:2 chroma subsampling employed in MPEG coding a macroblock (including a difference macroblock) consists of four Y blocks, one Cb and one Cr block each of size 8×8 pixels. As in JPEG while coding, a DCT transform is applied for each 8×8 block which then undergoes quantization, zig-zag scan and entropy coding.

As far as the sequence of I, P and B-frames is concerned in a MPEG-1 video datastream, there are certain guiding factors like resolution, access speed and compression ratio. For fast random access, the best resolution would be achieved by coding the whole datastream as I-frames. On the other hand the highest degree of compression is attained by using as many B-frames as possible. However, to perform B-frame decoding, the future I or P frame involved must be transmitted before any of the dependent B-frames can be processed. This would cause delay in the decoding proportional to the number of B-frames in the series. Considering all the issues, an optimized and proven sequence is 'IBBPBBPBBI'.

MPEG uses M to indicate the interval between a P-frame and its preceding I or P frame and N to indicate the interval between two consecutive I-frames.

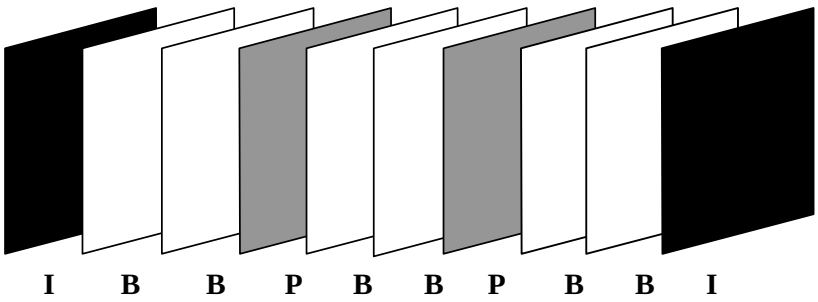


Figure 3.1: *MPEG Frame Sequence*

MPEG –1 video datastream comprises six hierarchical layers.

1. *Sequence layer*: A video sequence consists of one or more group of pictures and always starts with a sequence header. The header contains picture information such as horizontal size and vertical size, aspect ratio, frame rate, bit rate, and buffer size. These parameters can be changed with optional sequence headers between GOPs.
2. *Group of Pictures (GOPs) layer*: A GOP contains one or more pictures or frames at least one of which should be an I-frame. At this layer it is possible to distinguish the order of frames in the datastream with that in the display. The decoder decodes the

I-frame first in the datastream. But in the order of display, a B-frame can occur before an I-frame.

Datastream Decoding order:

Type of Frame I B B P B B P B B I

Frame number 1 2 3 4 5 6 7 8 9 10

Display order:

Type of frame B B I B B P B B P

Frame number 1 2 3 4 5 6 7 8 9

NOTES

3. *Picture layer*: This layer contains a displayable picture. Pictures are of 4 types – I, B, P and D though MPEG-1 doesn't allow mixing D pictures with other types.
4. *Slice layer*: Each slice consists of a number of macroblocks that may vary in a single image. The length and position of each slice are specified in the layer header. Slices are useful for recovery and synchronization of lost or corrupted bits.
5. *Macroblock layer*: Each image is divided into macroblocks. Each macroblock contains four Y blocks, one C_b block and one C_r block each of size 8×8. The coded blocks are preceded by a macroblock header, which contains all the control information (spatial address, motion vectors, prediction modes, quantizer step size etc.) belonging to the macroblock.
6. *Block layer*: Each 8×8 pixels block, the lowest level entity in the stream, can be intra-coded (using DCT), motion compensated or interpolated. Each block contains DC coefficients first, followed by variable length codes (VLC) and terminated by end-of-block marker.

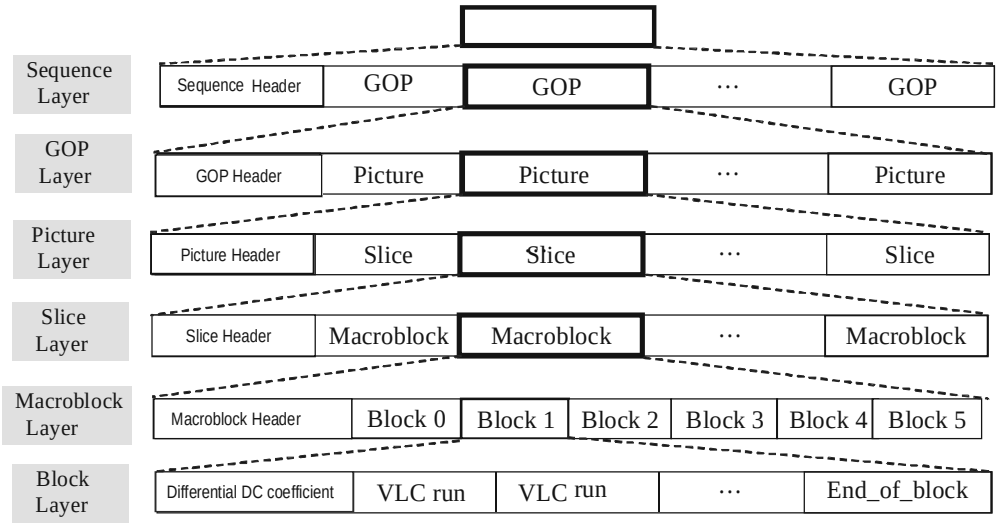


Figure 3.2: *Layers of MPEG-1 Video Stream*

An MPEG video data stream is not of much use on its own, except for playing silent video clips over the Internet or from a server. MPEG-7 standard defines the MPEG *Transport Stream* that transmits combined or multiplexed or packeted video, audio and ancillary data streams together. The *Programme Map Tables (PMTs*) identifies which audio and video signals go together to make a particular program out of several are channels transmitted.

NOTES

We have seen MPEGs that only define a video data stream and not a video file format. Several approaches have been proposed for standardization of video files architecture but QuickTime has established itself as a de facto standard. Another widely used video file format is AVI of Microsoft for Windows.

3.2.6 Video Editing Software

For building multimedia applications, digitized video is often required to be edited using specialized softwares. *Adobe Premiere* is a popular mid-range non-linear video editing application that provides some post-production facilities on desktop platform. Three windows are used by Premiere namely *Project*, *Timeline* and *Monitor*. Project window is used for importing and displaying raw video and audio clips and still images with all relevant information. The timeline window provides a visual display of the linear extent of the completed movie, showing the order of its component clips. It uses multiple audio and video tracks for transitions and overlays. Audio and video clips can be dragged from the project window and dropped on the timeline for assembly.

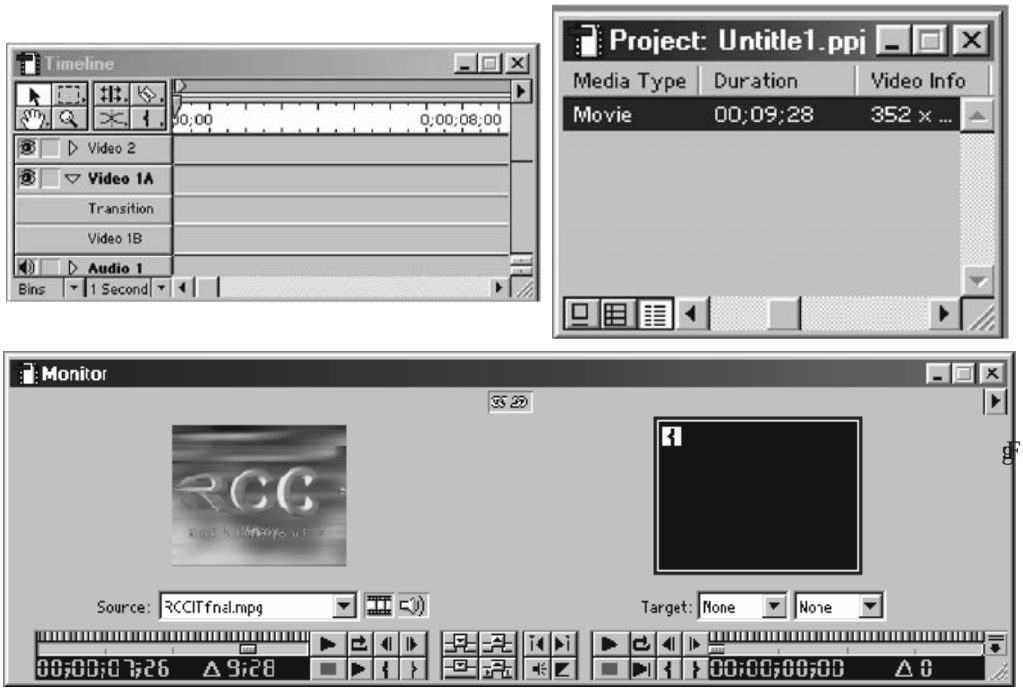


Figure 3.3: *Timeline, Project and Monitor Window*

The monitor window is used for editing and viewing the video frames. Editing includes trimming, overlaying, applying effects like dissolve, wipes, spins and page turns for transition of one clip to another. Serious post production operations (changes) like colour and contrast corrections, blurring or sharpening of images, element insertion and compositing, applying filter to a clip and vary it over time, sophisticated interpolation between key frames and so on can be done with more control and perfection by using dedicated post-production softwares like *Adobe After Effects*.

3.3 DISPLAY DEVICES

The most prominent part in a personal computer is the display system that makes graphic display possible. The display system may be attached to a PC to display character, picture and video outputs. Some of the common types of display systems available in the market are:

1. Raster Scan Displays
2. Random Scan Displays
3. Direct View Storage Tube
4. Flat Panel Displays
5. Three Dimensional Viewing Devices
6. Stereoscopic and Virtual Reality System

NOTES

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

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

3.3.1 Pixel

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

3.3.2 Resolution

There are two distinctly different terms, which are often confused. One is *Image Resolution* and the other is *Screen Resolution*. Strictly speaking image resolution refers to the pixel spacing, i.e., the distance from one pixel to the next pixel. A typical PC monitor displays screen images with a resolution somewhere between 25 pixels per inch and 80 pixels per inch (ppi). In other words, resolution of an image refers to the total number of pixels along the entire height and width of the image. For example, a full-screen image with resolution 800 × 600 means that there are 800 columns of pixels, each column comprising 600 pixels, i.e., a total of 800 × 600 = 4,80,000 pixels in the image area.

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

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

NOTES

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

3.3.3 Image Resolution versus Dot Pitch

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

3.3.4 Aspect Ratio

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

Table 3.2: *Common Resolutions, Respective Number of Pixels and Standard Aspect Ratios*

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

3.4 RASTER SCAN DISPLAY

This type of display basically employs a *Cathode Ray Tube (CRT)* or *LCD Panel* for display. The CRT works just like the picture tube of a television set. Its viewing surface is coated with a layer of arrayed phosphor dots. At the back of the CRT is a set of electron guns (cathodes) which produce a controlled stream of electrons (electron beam). The phosphor material emits light when struck by these high-energy electrons. The frequency and intensity of the light emitted depends on the type of phosphor material used and energy of the electrons. To produce a picture on the screen, these directed electron beams start at the top of the screen and scan rapidly from left to right along the row of phosphor dots. They return to the left-most position one line down and scan again, and repeat this to cover the entire screen. The return of the beam to the leftmost position one line down is called *horizontal retrace* during which the electron flow is shut off. In performing this scanning or sweeping type motion, the electron guns are controlled by the video data stream that comes into the monitor from the video card. This varies the intensity of the electron beam

at each position on the screen. The instantaneous control of the intensity of the electron beam at each dot is what controls the colour and brightness of each pixel on the screen. All this happens very quickly, and the entire screen is drawn in a fraction (say, 1/60th) of a second.

NOTES

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

3.4.1 Refresh Rate and Interlacing

When a dot of phosphor material is struck by the electron beam, it glows for a fraction of a second and then fades. As brightness of the dots begins to reduce, the screen-image becomes unstable and gradually fades out.

In order to maintain a stable image, the electron beam must sweep the entire surface of the screen and then return to redraw it a number of times per second. This process is called *refreshing* the screen. After scanning all the pixel-rows of the display surface, the electron beam reaches the rightmost position in the bottommost pixel line. The electron flow is then switched off and the vertical deflection mechanism steers the beam to the top left position to start another cycle of scanning. This diagonal movement of the beam direction across the display surface is known as *vertical retrace*. If the electron beam takes too long to return and redraw a pixel, the pixel will begin to fade; it will return to full brightness only when redrawn. Over the full surface of the screen, this becomes visible as a *flicker* in the image, which can be distracting and hard on the eyes.

In order to avoid flicker, the screen image must be redrawn fast enough so that the eye cannot tell that refresh is going on. The *refresh rate* is the number of times per second that the screen is refreshed. It is measured in Hertz (Hz), the unit of frequency. The refresh rates are somewhat standardized; common values are 56, 60, 65, 70, 72, 75, 80, 85, 90, 95, 100, 110 and 120 Hz. Though higher refresh rates are preferred for better comfort in viewing the monitor, the maximum refresh rate possible depends on the resolution of the image. The maximum refresh rate that a higher resolution image can support is less than that supported by a lower resolution image, because the monitor has more number of pixels to cover with each sweep. Actually support for a given refresh rate requires two things: a video card capable of producing the video images that many times per second, and a monitor capable of handling and displaying that many signals per second.

Every monitor should include, as part of its specification, a list of resolutions it supports and the maximum refresh rate for each resolution. Many video cards now include setup utilities that are pre-programmed with information about different monitors. When you select a monitor, the video card automatically adjusts the resolutions and respective allowable refresh rates. Windows 95 and later versions extend this facility by supporting Plug and Play for monitors; you plug the monitor in and Windows will detect it, set the correct display type and choose the optimal refresh rate automatically.

Some monitors use a technique called *interlacing* to cheat a bit and allow themselves to display at a higher resolution than is otherwise possible. Instead of refreshing every line of the screen, when in an interlaced mode, the electron guns sweep alternate lines on each pass. In the first pass, odd-numbered lines are refreshed, and in the second pass, even-numbered lines are refreshed. This allows the refresh rate to be doubled because only half the screen is redrawn at a time. The usual refresh rate for interlaced operation is 87 Hz, which corresponds to 43.5 Hz of 'real' refresh in half-screen interlacing.

NOTES

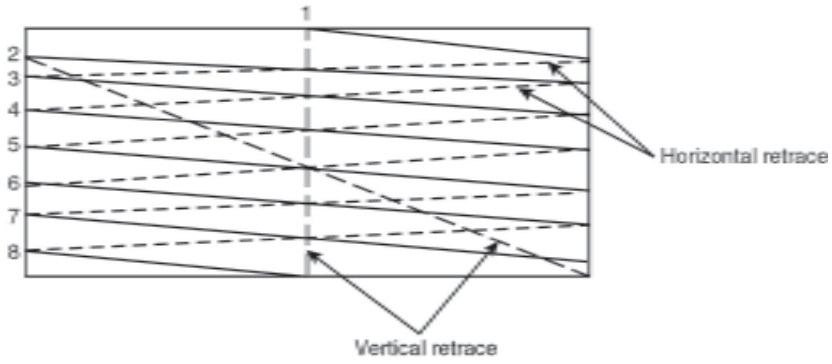


Figure 3.4: *Schematic Diagram of an Interlaced Raster Scan*

In the Figure 3.4, the odd-numbered lines represent scanning one half of the screen and the even-numbered lines represent scanning of the other half. There are two separate sets of horizontal and vertical retrace.

3.4.2 CRT

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

The primary component in an electron gun is a *cathode* (negatively charged) encapsulated by a metal cylinder known as the *control grid*. A heating element inside the cathode causes the cathode to be heated as current is passed. As a result electrons 'boil-off' the hot cathode surface. These electrons are accelerated towards the CRT screen by a high positive voltage applied near the screen or by an *accelerating anode*. If allowed to continue uninterrupted, the naturally diverging electrons would simply flood the entire screen. The cloud of electrons is forced to converge to a small spot as it touches the CRT screen by a *focusing system* using an electrostatic or magnetic field. Just as an optical lens focuses a beam of light at a particular focal distance, a positively charged metal cylinder focuses the electron beam passing through it on the centre of the CRT screen. A pair of magnetic *deflection coils* mounted outside the CRT envelope deflects the concentrated electron beam to converge at different points on the screen in the process of scanning. Horizontal deflection is obtained by one pair of coils and vertical deflection by the other pair, and the deflection amount is controlled by adjusting the current passing through the coils. When the electron beam is deflected away from the centre of the screen, the point of convergence tends to fall behind the screen resulting in a blurred (defocused) display near the screen edges. In high-end display devices this problem is eliminated by a mechanism which dynamically adjusts the beam focus at different points on the screen.

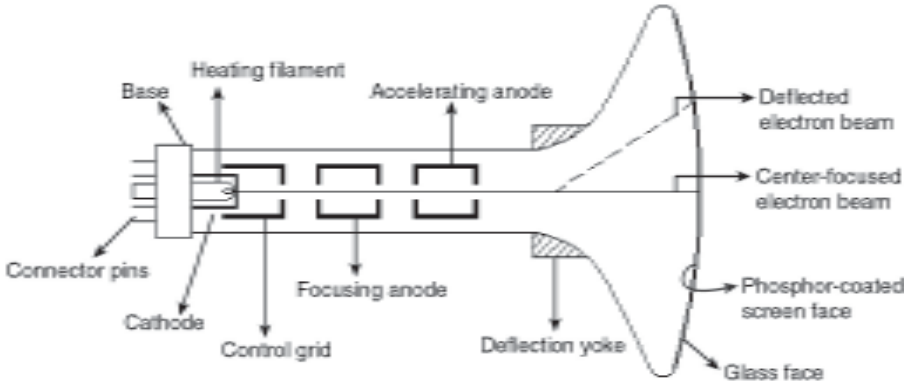


Figure 3.5: *Schematic Diagram of a Raster Scan CRT*

NOTES

When the electron beam converges on to a point on the phosphor-coated face of the CRT screen, the phosphor dots absorb some of the kinetic energy from the electrons. This causes the electrons in the phosphor atoms to jump to higher energy orbits. After a short time these excited electrons drop back to their earlier stable state, releasing their extra energy as small quantum of light energy. As long as these excited electrons return to their stable state phosphor continue to glow (*phosphorescence*) but gradually loses brightness. The time between the removal of excitation and the moment when phosphorescence has decayed to 10 per cent of the initial brightness is termed as *persistence* of phosphor. The brightness of the light emitted by phosphor depends on the intensity with which the electron beam (number of electrons) strikes the phosphor. The intensity of the beam can be regulated by applying measured negative voltage to the control grid. Corresponding to a zero value in the frame buffer a high negative voltage is applied to the control grid, which in turn will shut off the electron beam by repelling the electrons and stopping them from coming out of the gun and hitting the screen. The corresponding points on the screen will remain black. Similarly, a bright white spot can be created at a particular point by minimising the negative voltage at the control grid of the three electron guns when they are directed to that point by the deflection mechanism.

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

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

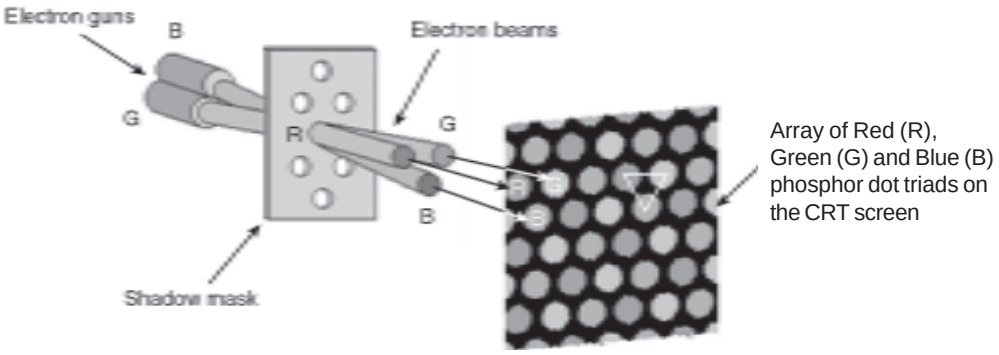


Figure 3.6: *Electron Guns Passing Through CRT*

NOTES

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

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

3.4.3 Bit Planes, Colour Depth and Colour Palette

The appearance and colour of a pixel of an image is a result of intersection of three primary colours (red, green and blue) at different intensities. When the intensities of all three electron beams are set to the highest level (causing each dot of a triad to glow with maximum intensity), the result is a white pixel; when all are set to zero, the pixel is black. And for many different combinations of intermediate intensity levels, several million colour pixels can be generated. For a mono monitor using a single electron gun, the phosphor material can glow with varied intensities depending on the intensity of the electron beam. As a result a pixel can be black (zero intensity) or white (maximum intensity) or have different shades of grey.

The number of discrete intensities that the video card is capable of generating for each primary colour determines the number of different colours that can be displayed. The number of memory bits required to store colour information (intensity values for all three primary colour components) about a pixel is called *colour depth* or *bit depth.* A minimum of one memory bit (colour depth = 1) is required to store intensity value either 0 or 1 for every screen point or pixel. Corresponding to the intensity value 0 or 1, a pixel can be black or white respectively. So if there are n pixels in an image a total of n bits of memory used for storing intensity values will result in a pure black and white image. The block of memory which stores (or is mapped with) bilevel intensity values for each pixel of a full-screen pure black and white image is called a *bit plane* or *bitmap*.

Figure 3.7: *For bit depth =1, a pixel is illuminated (white) if intensity value 1 is stored in the corresponding memory address in the frame buffer.*

Colour or grey levels can be achieved in the display using additional bit planes. First consider a single bit plane – a planar array of bits, with one bit for each screen pixel. This plane is replicated as many times as there are bits per pixel, placing each bit plane behind its

predecessor. Hence, the result for *n*-bits per pixel (colour depth = *n*) is a collection of *n* bit planes that allows specifying any one of 2^n colours or grey shades at every pixel.

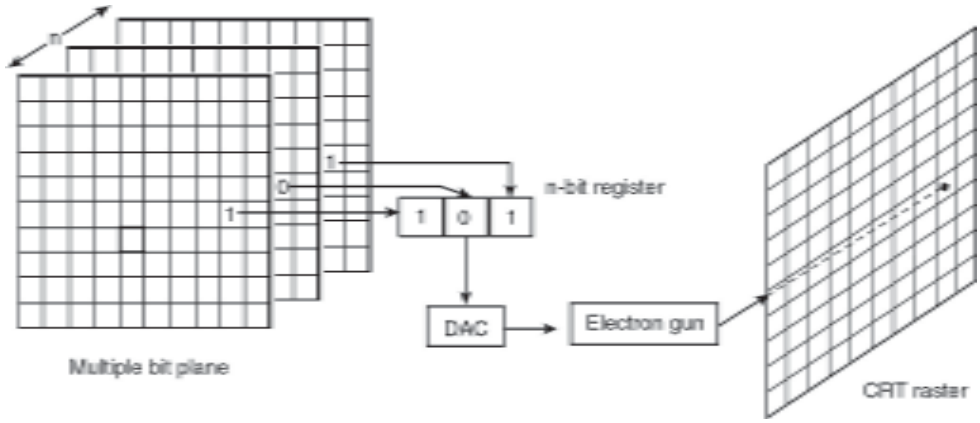


Figure 3.8: *For bit depth = n , n number of bit planes are used; each bit plane contributes to the gray shade of a pixel.*

The more the number of bits used per pixel, the finer the colour detail of the image. However, increased colour depths also require significantly more memory for storage, and also more data for the video card to process, which reduces the allowable refresh rate.

Table 3.3: *Common Colour Depths Used in PCs*

Colour Depth	Number of Displayed Colours	Bytes of Storage Per Pixel	Common Name for Colour Depth
4-Bit	16	0.5	Standard VGA
8-Bit	256	1.0	256-Colour Mode
16-Bit	65,536	2.0	High Colour
24-Bit	16,777,216	3.0	True Colour

For *true colour* three bytes of information are used, one each of the red, blue and green signals that make a pixel. A byte can hold 256 different values and so 256 voltage settings are possible for each electron gun which means that each primary colour can have 256 intensities, allowing over 16 million (256 × 256 × 256) colour possibilities. This allows for a very realistic representation of the images, without necessitating any colour compromise. In fact, 16 million colours is more than the human eye can discern. True colour is a necessity for those involved in high quality photo editing, graphical design, etc.

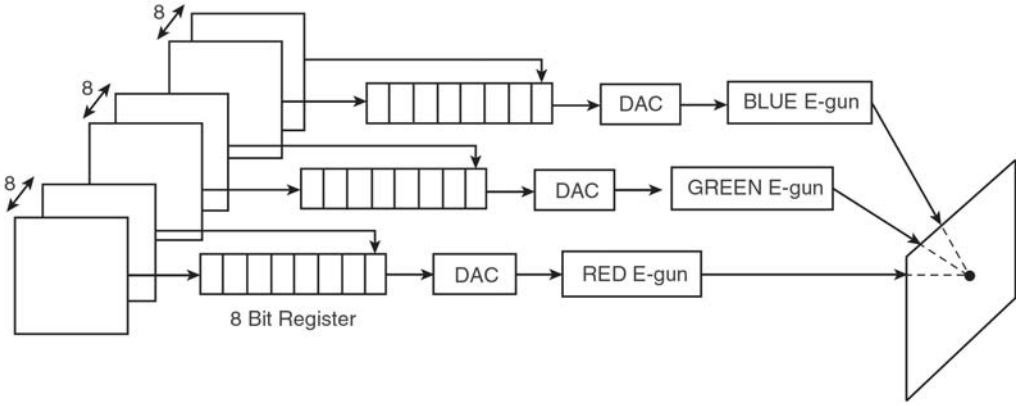


Figure 3.9: *For bit depth =24 (true colour display), 8 bit planes are used for storing each primary colour component of the colour value of a pixel.*

NOTES

NOTES

For *high colour* two bytes of information are used to store the intensity values for all three colours. This is done by dividing 16 bits into 5 bits for blue, 5 bits for red and 6 bits for green. This means 32 (= 2^5) intensities for blue, 32 (= 2^5) for red, and 64 (= 2^6) for green. This reduced colour precision results in a loss of visible image quality, but one cannot easily see the difference between true colour and high colour image. However high colour is often used instead of true colour because high colour requires 33 per cent (or 50 per cent in some cases) less memory and also image generation is faster.

In *256-colour mode* the PC uses only 8 bits; this means something like 2 bits for blue and 3 each for green and red . There are chances that most of the colours of a given picture are not available, and choosing between only 4 (= 2^2) or 8 (= 2^3) different values for each primary colour would result in rather blocky or grainy look of the displayed image. A *palette* or *look-up table* is used here. A palette is a separate memory block (in addition to the 8 bit plane) created containing 256 different colours. The intensity values stored therein are not constrained within the range 0 to 3 for blue and 0 to7 each for green and red. Rather each colour is defined using the standard 3-byte colour definition that is used in true colour. Thus the intensity values for each of the three primary colour components can be anything between 0 and 255 in each of the table entries. Upon reading the bit planes, the resulting number instead of directly specifying the pixel colour, is used as a pointer to the 3-byte colour value entry in the look-up table. For example, if the colour number read from the bit-planes is 10 for a given pixel, then the intensities of red, green and blue to be displayed for that pixel will be found in the 10th entry of the table. So the full range of true colour can be accessed, but only 256 of the available 16 million colours can be used at a time.

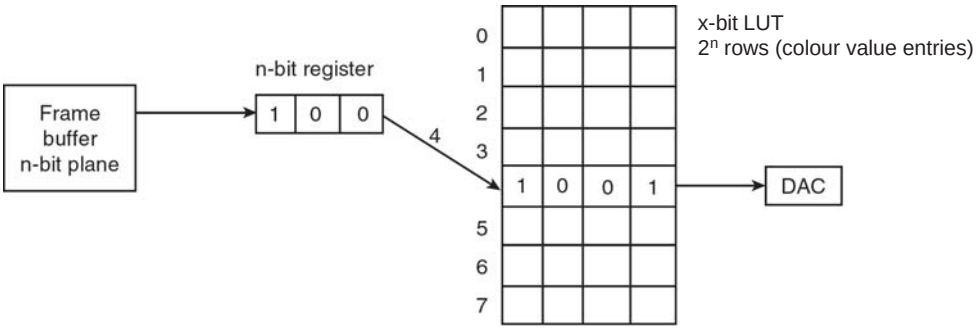


Figure 3.10: *The n-bit register holds the row number of the look-up table; the particular row pointed contains the actual pixel intensity value which is a x-bit number (x>n).*

The palette is an excellent compromise at the cost of moderate increase in memory: it allows only 8 bits of the frame buffer to be used to specify each colour in an image and allows the creator of the image to pick any of the 256 colours for the image. This is because the palette can be reloaded any time with different combinations of 256 colours out of 16 million without changing the frame buffer values. Since virtually no image contains an even distribution of colours, this allows for more precision in an image by using more colours than would be possible by assigning each pixel a 2-bit value for blue and 3-bit value each for green and red. For example, an image of the sky with clouds (like the Windows 95 standard background) would have different shades of blue, white and gray, and virtually no red, green, yellow and the like.

256-colour is the standard for much of computing, mainly because the higher-precision colour modes require more resource (especially video memory) and are not supported by many PCs. Despite the ability to 'hand pick' the 256 colours, this mode produces noticeably worse image quality than high colour.

NOTES

3.4.4 Frame Buffer and Output Circuitry

In the early days of PCs, the amount of information displayed was less. A screen of monochrome text, for example, needs only about 2 KB of memory space. Special parts of the upper memory area (UMA) were dedicated to hold this video data. As the need for video memory increased into the megabyte range, it made more sense to put the memory on the video card itself. In fact, to preserve existing PC design limitations, it was necessary (the UMA does not have the space to hold bigger screen images). The *frame buffer* is the video memory (RAM) that is used to hold or map the image displayed on the screen. The amount of memory required to hold the image depends primarily on the resolution of the screen image and also the colour depth used per pixel. The formula to calculate how much video memory is required at a given resolution and bit depth is:

Memory in MB = (X-resolution × Y-resolution × Bits-per-pixel) / (8 × 1024 × 1024)

However one needs more memory than this formula computes. One major reason is that video cards are available only in certain memory configurations (in terms of whole megabytes). For example you can't order a card with 1.7 MB of memory; you have to use a standard 2MB card available in the market. Another reason is that many video cards, especially high end accelerators and 3D cards, use memory for computation as well as for the frame buffer. Thus, they need much more memory than is required strictly to hold the screen image.

Table 3.4 displays, in binary megabytes, the amount of memory required for the frame buffer for each common combination of screen resolution and colour depth. The smallest industry standard video memory configuration required to support the combination is shown in parentheses.

Table 3.4: *Radio Memory Configurations*

Resolution	4 Bits	8 Bits	16 Bits	24 Bits	32 Bits
320 × 200	0.03 (256 KB)	0.06 (256 KB)	0.12 (256 KB)	0.18 (256 KB)	--
640 × 480	0.15 (256 KB)	0.29 (512 KB)	0.59 (1 MB)	0.88 (1 MB)	1.17 (2 MB)
800 × 600	--	0.46 (512 KB)	0.92 (1 MB)	1.37 (2 MB)	1.83 (2 MB)
1024 × 768	--	0.75 (1 MB)	1.50 (2 MB)	2.25 (4 MB)	3.00 (4 MB)
1280 × 1024	--	1.25 (2 MB)	2.50 (4 MB)	3.75 (4 MB)	5.00 (6 MB)
1600 × 1200	--	1.83 (2 MB)	3.66 (4 MB)	5.49 (6 MB)	7.32 (8 MB)

Some motherboards designed are to integrate the video chipset into itself and use a part of the system RAM for the frame buffer. This is called *unified memory architecture*. This is done to save costs. The result is poorer video performance, because in order to use higher resolutions and refresh rates, the video memory needs a higher performance than the RAM normally used for the system. This is also the reason why video card memory is so expensive compared to regular system RAM.

In order to meet the increasing demand for faster and dedicated video memory at a comparable price, a technology was introduced by Intel which is fast becoming a new standard. It is called the *Accelerated Graphics Port* or *AGP*. The AGP allows the video processor to access the system memory for graphics calculations, but keeps a dedicated video memory for the frame buffer. This is more efficient because the system memory can be shared dynamically between the system processor and the video processor, depending on the needs of the system. However it should be remembered that AGP is considered a port – a dedicated interface between the video chipset and the system processor.

The display adapter circuitry (on video card or motherboard) in a raster graphics system typically employs a special purpose processor called *Display Processor* or *Graphics Controller* or *Display Coprocessor* which is connected as an I/O peripheral to the CPU.

NOTES

Such processors assist the CPU in scan-converting the output primitives (line, circle, arc etc.) into bitmaps in frame buffer and also perform raster operations of moving, copying and modifying pixels or block of pixels. The output circuitry also includes another specialized hardware called *Video Controller* which actually drives the CRT and produces the display on the screen.

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

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

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

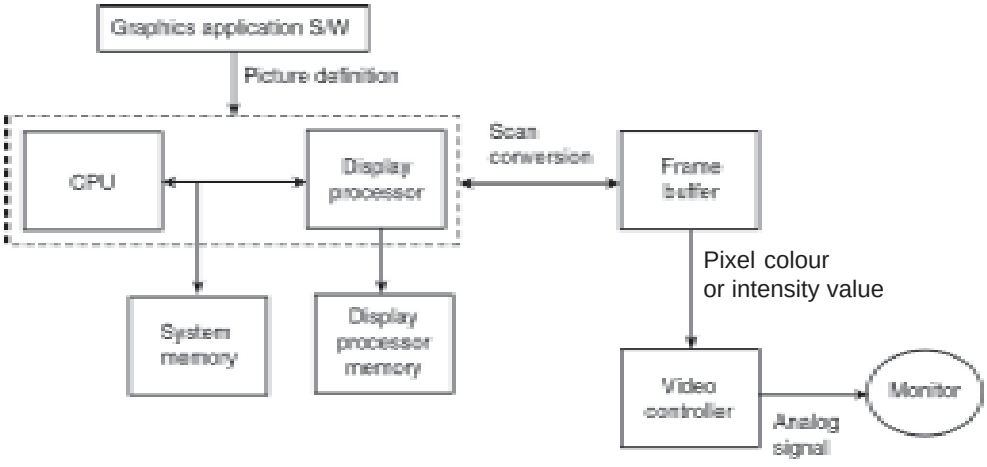


Figure 3.11: *General Architecture of a Raster Display System*

NOTES

As the time available to display or refresh a single pixel on the screen is too less (in the order of a few nanoseconds), accessing the frame buffer every time for reading each pixel intensity value would consume more time than what is allowed. Therefore, multiple adjacent pixel values are fetched to the frame buffer in a single access and stored in a register. After every allowable time gap (as dictated by the refresh rate and resolution) one pixel value is shifted out from the register to control the beam intensity for that pixel. The procedure is repeated with the next block of pixels and so on, thus the whole group of pixels will be processed.

Figure 3.12: *Logical Operations of the Video Controller*

3.5 RANDOM SCAN DISPLAY

Basically there are two types of CRTs – Raster Scan type and Random Scan type. The main difference between the two is the technique with which the image is generated on the phosphor coated CRT screen. In raster scan method, the electron beam sweeps the entire screen in the same way you would write a full page text in a notebook, word by word, character by character, from left to right, and from top to bottom. In random scan technique, the electron beam is directed straightaway to the particular point(s) of the screen where the image is to be produced. It generates the image by drawing a set of random straight lines much in the same way one might move a pencil over a piece of paper to draw an image – drawing strokes from one point to another, one line at a time. This is why this technique is also referred to as *vector drawing* or *stroke writing* or *calligraphic display*.

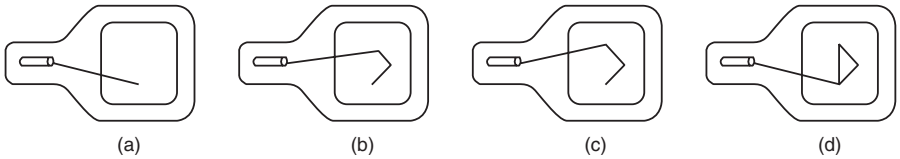


Figure 3.13: *Drawing a Triangle on a Random Scan Display*

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

NOTES

the display commands. When the beam focus must be moved from the end of one stroke to the beginning of the other, the beam intensity is set to 0.

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

3.6 DIRECT VIEW STORAGE TUBE

Direct View Storage Tube (DVST) is rarely used today as part of a display system. However, DVST marks a significant technological change in the usual refresh type display. Both in the raster scan and random scan system the screen image is maintained (flicker free) by redrawing or refreshing the screen many times per second by cycling through the picture data stored in the refresh buffer. In DVST there is no refresh buffer; the images are created by drawing vectors or line segments with a relatively slow-moving electron beam. The beam is designed not to draw directly on phosphor but on a fine wire mesh (called *storage mesh*) coated with dielectric and mounted just behind the screen. A pattern of positive charge is deposited on the grid, and this pattern is transferred to the phosphor-coated screen by a continuous flood of electrons emanating from a separate flood gun.

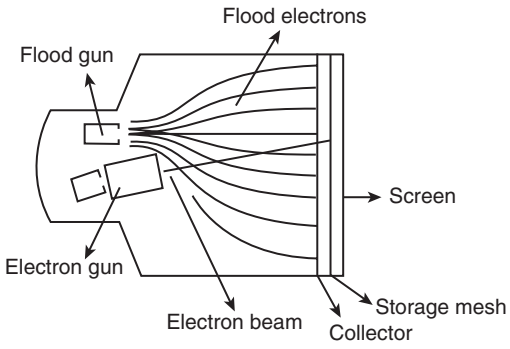


Figure 3.14: *Schematic Diagram of a DVST*

Just behind the storage mesh is a second grid, the *collector*, whose main purpose is to smooth out the flow of flood electrons. These electrons pass through the collector at low velocity and are attracted to the positively charged portions of the storage mesh but repelled by the rest. Electrons not repelled by the storage mesh pass right through it and strike the phosphor.

To increase the energy of these slow-moving electrons and thus create a bright picture, the screen is maintained at a high positive potential.

The storage tube retains the image generated until it is erased. Thus no refreshing is necessary, and the image is absolutely flicker free.

A major disadvantage of DVST in interactive computer graphics is its inability to selectively erase parts of an image from the screen. To erase a line segment from the displayed image, one has to first erase the complete image and then redraw it by omitting that line segment. However, the DVST supports a very high resolution which is good for displaying complex images.

3.7 FLAT PANEL DISPLAY

To satisfy the need of a compact portable monitor , modern technology has gifted us with *LCD panel*, *Plasma display panel*, *LED panel* and *thin CRT*. These display devices are

NOTES

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

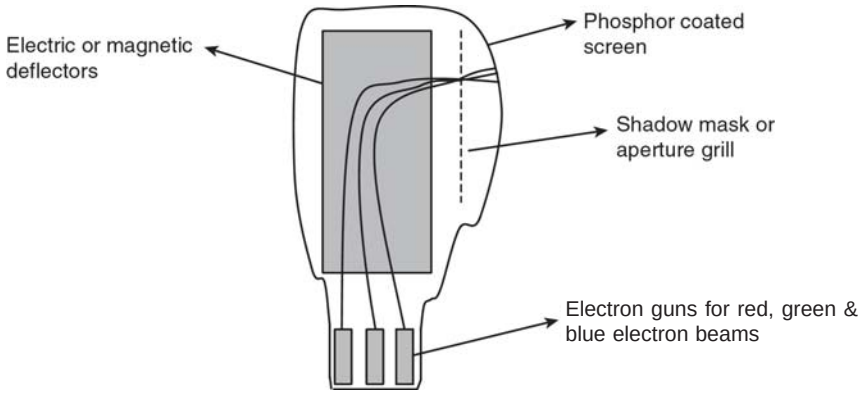


Figure 3.15: *Thin CRT*

3.7.1 LCD

To understand the fundamental operation of a simple LCD, a model is shown Figure 3.15. LCD basically consists of a layer of liquid crystal, sandwiched between two polarizing plates. The polarizers are aligned perpendicular to each other (one vertical and the other horizontal), so that the light incident on the first polarizer will be blocked by the second. This is because a polarizer plate only passes photons (quanta of light) with their electric fields aligned parallel to the polarizing direction of that plate.

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

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

NOTES

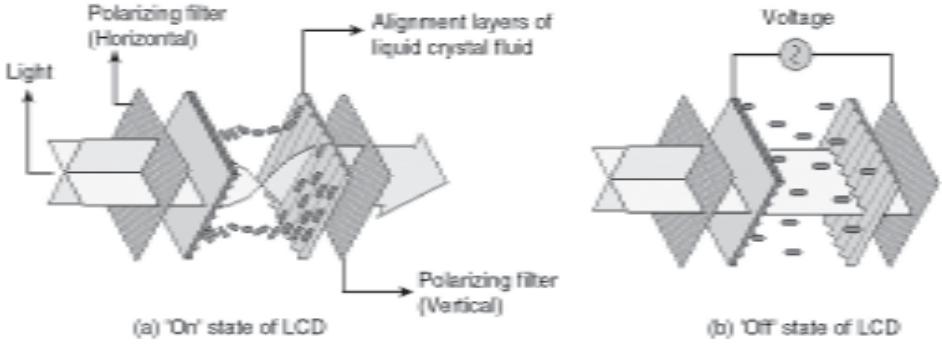


Figure 3.16: *LCD*

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

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

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

3.7.2 Plasma Panel

Here a layer of gas (usually neon) is sandwiched between two glass plates. Thin vertical (column) strips of conductor run across one plate, while horizontal (row) conductors run up and down the other plate. By applying high voltage to a pair of horizontal and vertical conductors, a small section of the gas (tiny neon bulb) at the intersection of the conductors breaks down into glowing plasma of electrons and ions. Thus, in the array of gas bulbs, each one can be set to an 'on' (glowing) state or 'off' state by adjusting voltages in the appropriate pair of conductors. Once set 'on' the bulbs remain in that state until explicitly turned 'off' by momentarily reducing the voltage applied to the pair of conductors. Hence no refreshing is necessary.

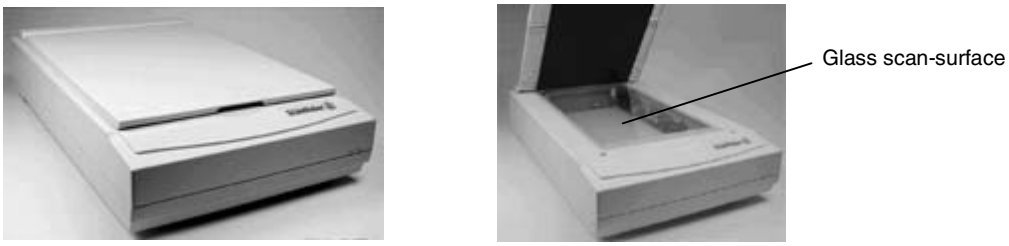
Because of its excellent brightness, contrast and scalability to larger sizes, plasma panel is attractive. Research is on to eliminate the colour-display limitation of such device at low production cost.

3.8 READYMADE IMAGE

So far we have discussed some fundamental concepts on how graphic images are generated and stored in some of the most common and widely used display systems. Let us briefly

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

There are basically three types of scanners – *Drum*, *Flatbed* and *Sheetfed* scanners. Drum scanners are the high-end ones, whereas sheetfed scanners are the ordinary type. Flatbed scanners strike a balance between the two in quality as well as price. There are also *handheld scanners* or *bar-code readers* which are typically used to scan documents in strips of about 4 inches wide by holding the scanner in one hand and sliding it over the document.



(a) Scanner (b) Scan-surface in a Scanner

Figure 3.17: *Flatbed Scanner*

3.8.1 Flatbed Scanner

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

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

Figure 3.18: *Mirror and Lens Assembly in the Scanner Carriage*

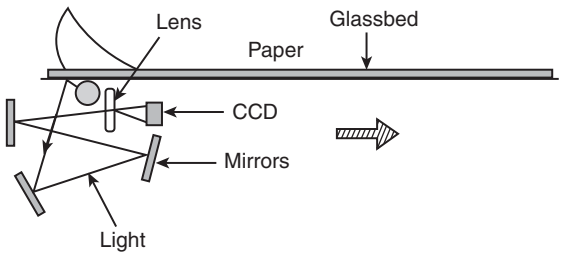


Figure 3.19: *Scanning Operation; The assembly of light, mirrors, lens and CCD moves over the length of the glassbed in the direction shown while scanning a paper.*

NOTES

Check Your Progress

1. The refresh buffer is not used in
   (a) raster scan display system
   (b) random scan display system
   (c) thin CRT
   (d) DVST
2. The laptop generally uses
   (a) LCD panel
   (b) LED panel
   (c) plasma panel
   (d) active matrix TFT
3. The display on the CRT-based monitor screen is produced by
   (a) video controller
   (b) raster scan generator
   (c) display processor
   (d) DAC

NOTES

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

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

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

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

In the later chapters we will explore how basic graphic entities are drawn, manipulated and viewed on a computer.

3.9 SUMMARY

The three most common types of CRT display technologies are raster scan display, random scan display and DVST display. While discussing the various display technologies this unit addresses issues from conceptual point of view, i.e. the functional aspects and not the details of electronics.

Interactive computer graphics demand display devices whose image can be changed quickly and those which have high resolutions with low memory overhead. Flat panel display technology is developing at a rapid rate and may largely replace raster display in the near future. Even with recent advances, an individual display may incorporate more than one technology.

3.10 ANSWERS TO 'CHECK YOUR PROGRESS'

1. (d)
2. (a)
3. (a)

3.11 EXERCISES AND QUESTIONS

NOTES

1. What are frame grabbers? Are frame buffers different from frame grabbers?
2. Draw a neat block diagram, to explain the architecture of a raster display.
3. Give the logical organization of the video controller in a raster display system.
4. What is refresh buffer? Identify the contents and the organization of the refresh buffer for the case of raster display and vector display.
5. Describe the function of an image scanner.
6. What role does CCD play in an image scanner?
7. How are different shades of colour generated on the RGB monitors?
8. What is the role of shadow masks in graphics monitors?
9. What do you understand by VGA and SVGA monitors?
10. What is computer graphics? Indicate five practical applications of computer graphics.
11. Discuss in brief different interactive picture construction techniques.
12. How is colour depth and resolution of an image related to the video memory requirement?
13. What is the fundamental difference in the method of operation of a monochrome CRT and a colour CRT?
14. Compare storage type CRT against refresh type CRT display. List the important properties of phosphor being used in CRTs.
15. Compare and contrast the operating characteristics of raster refresh systems, plasma panels and LCDs.
16. Bring out the need for a colour look-up table. Give the organization of a colour look-up table providing 12 bits per entry, per colour for each pixel position and with 8 bits per pixel in the frame buffer.
17. A colour display device has 8 bitplanes and a look-up table of 256 entries, each of which can hold a 24 bit number. The manufacturer claims it can 'display 256 colours out of a palette of 16 million'. Explain this statement.
18. Briefly explain two main classes of hardware device for user interaction.
19. If a monitor screen has 525 scan lines and an aspect ratio of 3:4 and if each pixel contains 8 bits worth of intensity information, how many bits per second are required to show 30 frames each second?
20. For a medium resolution display of 640 pixels by 480 lines refreshing 60 times per second, the video controller fetches 16 bits in one memory cycle. RAM memory chips have cycle times around 200ns. How many memory cycles will be needed for displaying 16 one bit pixels? *DOEACC-'A' level — Jan 2001.*
21. 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 × 1024, a refresh rate of 60Hz, a horizontal retrace time of 5 μsec and a vertical retrace time of 500 μsec.
22. Assume a raster scan display system supports a frame buffer size of 256 × 256 × 2 bits. Two bits/pixel are used to look up a 4 × 2 colour table. The entries in the colour table are writable once per raster scan only during the vertical retrace period. The actual colour codes are given as follows.

    00 Black 01 Red 10 Yellow 11 White

NOTES

(i) Give a scheme for using the frame buffer if it consists of two separate image planes of size 256 × 256 × 1 each. Plane 1 is to be displayed as yellow on red image. Plane 2 is to be displayed as white on black.

(ii) How will you turn on a pixel in either plane?

(iii) How will you delete a pixel in either plane?

3.12 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

UNIT 4 SCAN CONVERSION ALGORITHMS

NOTES

Structure

4.0 INTRODUCTION

The major objective of computer graphics is to model graphic objects in a scene. Such objects comprise entity primitives like point, line, circle, ellipse and curves. For displaying these entities on a raster display the representative sets of pixels of the appropriate intensity or colour must be identified. There are standard scan conversion algorithms to calculate first the position coordinates of the representative pixels of each entity primitive from basic input data and then store pixel-wise intensity information into the graphics memory. This unit discusses the different approaches and issues for implementing line, circle and ellipse in digital space.

4.1 UNIT OBJECTIVES

- Understanding the concept of scan conversion of an object's true mathematical definition
- Presenting algorithms for scan converting a line with reference to specific case study and background mathematics
- Presenting algorithms for scan converting a circle with reference to specific case study and mathematical derivation
- Presenting algorithms for scan converting an ellipse
- Discussing the side-effects of scan conversion and techniques to minimize those effects

4.2 POINTS AND LINES

In the previous chapters we have seen that in order to draw the primitive objects, one has to first *scan convert* the objects. This refers to the operation of finding out the location of pixels to be intensified and then setting the values of corresponding bits, in the graphics memory, to the desired intensity code. Each pixel on the display surface has a finite size depending on the screen resolution and hence a pixel cannot represent a single mathematical point. However, we consider each pixel as a unit square area identified by the coordinate of its lower left corner, the origin of the reference coordinate system being located at the lower left corner of the display surface. Thus each pixel is accessed by a non-negative integer coordinate pair (x, y). The x values start at the origin and increase from left to right

NOTES

along a scan line and the *y* values (i.e., the scan line numbers) start at bottom and increase upwards.

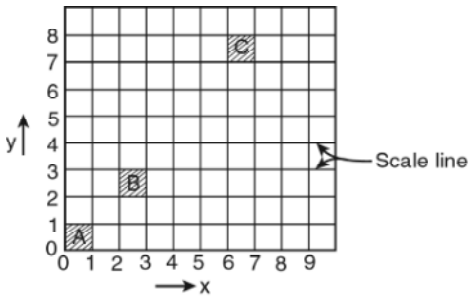


Figure 4.1: *Array of Square Pixels on the Display Surface. Coordinate of pixel A: 0, 0; B: 2, 2; C: 6,7. A coordinate position (6.6,7.25) is represented by C, whereas (2.3,2.5) is represented by B. For plotting on the screen the calculated pixel coordinates are rounded off to nearest integers.*

Line drawing is accomplished by calculating intermediate point coordinates along the line path between two given end points. Since screen pixels are referred with integer values, plotted positions may only approximate the calculated coordinates – i.e., pixels which are intensified are those which lie very close to the line path if not exactly on it which is in the case of perfectly horizontal, vertical or 45° lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desired line. Still, screen resolution is a big factor towards improving the approximation. In a high resolution system the adjacent pixels are so closely spaced that the approximated line-pixels lie very close to the actual line path and hence the plotted lines appear to be much smoother — almost like straight lines drawn on paper. In a low resolution system, the same approximation technique causes lines to be displayed with a 'stairstep appearance' – not smooth (see Fig. 4.2(a)).

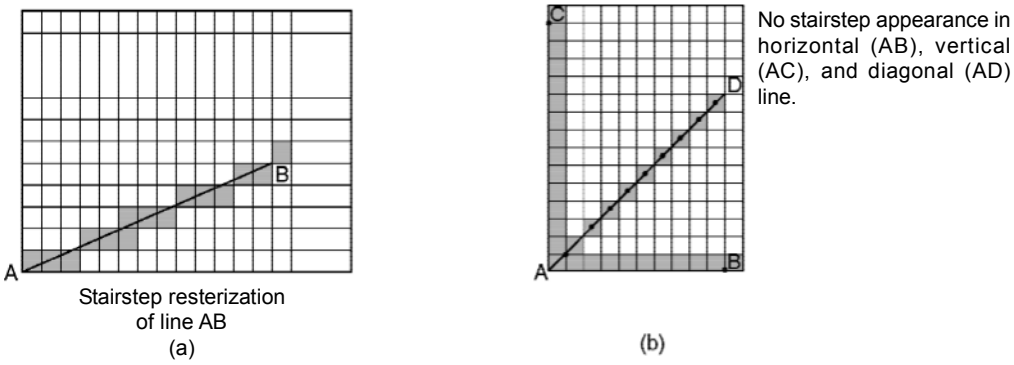


Figure 4.2

4.3 LINE DRAWING ALGORITHMS

Several line drawing algorithms are developed with the basic objective to create visually satisfactory images in least possible time. This is achieved by reducing the calculations to a minimum preferably using integer rather than floating point arithmetic. Such ways of minimizing even a single arithmetic operation is so important, because every drawing or image generated will have a large number of line segments in it and every line segment will have many pixels. So saving of even one computation per pixel will save number of computations in generating an object, which, in turn, minimizes the time required to generate the whole image on the screen.

Here, we will discuss *DDA* and *Bresenham's* algorithms, which scan converts lines with acceptable approximation in sufficiently less time.

4.3.1 DDA Algorithm

Assume that a line is to be rasterized between given endpoints (*xstart*, *ystart*) and (*xend*, *yend*). Now let us consider the equation of the line as

$$y = mx + c$$

where m represents the slope of the line and c is the y intercept. This slope can be expressed as

$$m = \frac{yend - ystart}{xend - xstart}$$

NOTES

In fact any two consecutive points (x_i, y_i) and (x_{i+1}, y_{i+1}) lying on this line segment should satisfy the equation $\frac{y_{i+1} - y_i}{x_{i+1} - x_i} = m$

If we say $y_{i+1} - y_i = \Delta y$ and $x_{i+1} - x_i = \Delta x$, then $\frac{\Delta y}{\Delta x} = m \Rightarrow \Delta y = m\Delta x$

Thus for any given x interval Δx along the line, we can compute the corresponding y interval Δy. Now if

$$\begin{aligned} &\Delta x = 1, \quad \text{i.e.,} \quad x_{i+1} - x_i = 1, \quad \text{i.e.,} \quad x_{i+1} = x_i + 1 \quad \text{then} \quad \Delta y = m \\ &\text{which} \quad \text{implies} \quad y_{i+1} - y_i = m, \quad \text{i.e.,} \quad y_{i+1} = y_i + m \end{aligned} \tag{1}$$

Thus a unit change in x changes y by m which is a constant for a given line. We know that if $x_{i+1} = x_i + 1$, then $y_{i+1} = y_i + m$; the values of x and y (on the line) are defined in terms of their previous values. Initializing (x_i, y_i) with (*xstart*, *ystart*) the line can be generated by incrementing the previous x values by one unit and solving the corresponding y value at each step, till *xend* is reached. At each step, we make incremental calculations based on the previous step. This is what is defined as incremental algorithm and often referred to as the *Digital Differential Analyzer (DDA)* algorithm.

While incrementing the values of y by constant m, in this DDA method, we have to keep in mind that m being the slope of the line, it can be any real number negative or positive – so that the calculated y values must be rounded off to the nearest integer for the sake of plotting on the screen.

But an obvious doubt at this point would be why not increment y values by 1 instead of x and accordingly calculate x values from the lines equation. The determining factor here is absolute value of the line's slope, i.e., $|m|$.

If $|m| \leq 1$ which implies $|\Delta y| \leq |\Delta x|$ we sample the line at unit x intervals, as done above.

But if $|m| > 1$ implying $|\Delta y| > |\Delta x|$, a unit step in x creates a step in y that is greater than 1, which is not desirable. In that case we reverse the roles of x and y by sampling at unit y intervals as,

$$\begin{aligned} &\Delta y = y_{i+1} - y_i = 1 \\ &\Rightarrow y_{i+1} = y_i + 1 \quad \text{and} \\ &\Delta x = \frac{\Delta y}{m} \quad \Rightarrow x_{i+1} - x_i = \frac{1}{m} \\ &\qquad \Rightarrow x_{i+1} = x_i + \frac{1}{m} \end{aligned} \tag{2}$$

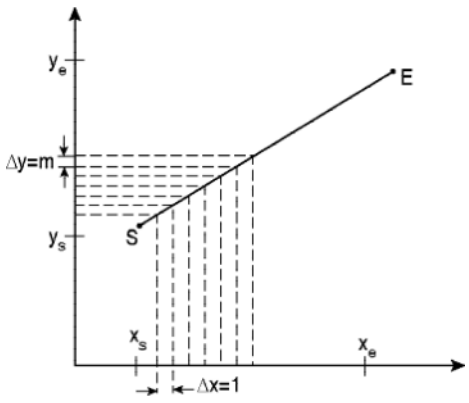


Figure 4.3: *A line Segment with Positive Slope and Slope < 1 is Sampled Along X Direction.*

Eqns. (1) and (2) are based on the assumption that *xstart* < *xend* and *ystart* < *yend,* i.e., slope is positive. But if it is not so then we have to apply negative increment as shown below for the other possible cases.

NOTES

	$xstart < xend$, $ystart > yend$ $\Rightarrow m$ is negative	$xstart > xend$, $ystart < yend$ $\Rightarrow m$ is negative	$xstart > xend$, $ystart > yend$ $\Rightarrow m$ is positive
$\lvert m \rvert \le 1$	 $x_{i+1} = x_i + 1$, $y_{i+1} = y_i + m$	 $x_{i+1} = x_i - 1$, $y_{i+1} = y_i + m$	 $x_{i+1} = x_i - 1$, $y_{i+1} = y_i - m$
$\lvert m \rvert > 1$	 $y_{i+1} = y_i - 1$, $x_{i+1} = x_i - \frac{1}{m}$	 $y_{i+1} = y_i + 1$, $x_{i+1} = x_i + \frac{1}{m}$	 $y_{i+1} = y_i - 1$, $x_{i+1} = x_i - \frac{1}{m}$

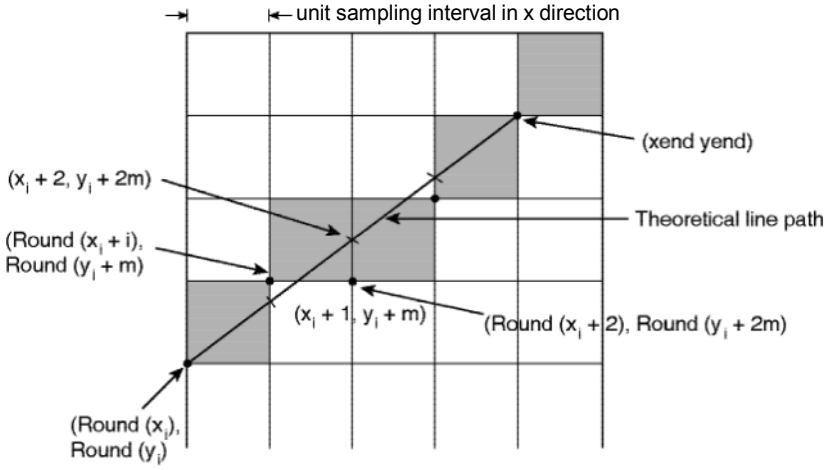


Figure 4.4: *Incremental Calculation of* x_i, y_i *pixel Coordinates for Scan Converting a Theoretical Line Path*

The *Pseudocode* for rasterizing a line according to DDA logic is presented below. This code works for any line in all the 4 quadrants.

We assume that points are to be plotted on a bilevel intensity system and a function Setpixel () is such that Setpixel (*x*, *y*, 1), will load the intensity value 1 into the frame buffer at a position corresponding to column *x* along scanline *y* on the screen.

We also assume a function Round () which rounds off the argument to the nearest integer.

Input = *xstart*, *ystart*, *xend*, *yend*
if abs (*xend* – *xstart*) $\ge$ abs (*yend* – *ystart*) : check whether | slope | $\le$ 1 or > 1 , then
span = abs (*xend* – *xstart*) and accordingly set the sampling length
else span = abs (*yend* – *ystart*)
Δx = (*xend* – *xstart*)/span : set the larger of Δx and Δy as one raster unit or unit sampling interval
Δy = (*yend* – *ystart*)/span : initialize *x*, *y* pixel coordinate with *xstart*,
x = *xstart*
ystart
y = *ystart*
i = 1

while ($i \leq$ span) : continue loop till *xend* or *yend* is reached.
Setpixel (Round (x), Round (y), 1)

$x = x + \Delta x$
$y = y + \Delta y$: incremental calculation to select next pixel.
$i = i + 1$
end while

NOTES

4.3.2 Bresenham's Line Algorithm

One serious drawback of the DDA algorithm is that it is very time-consuming as it deals with a rounding off operation and floating point arithmetic. Moreover, successive addition of floating point increment (m or $1/m$) causes accumulation of round off error and eventually a drift away of the plotted pixels from the true line path in case of long line segments.

The algorithm developed by Bresenham is more accurate and efficient compared to DDA algorithm because it cleverly avoids the 'Round' function and scan converts lines using only incremental integer calculation.

This algorithm samples a line by incrementing by one unit either x or y depending on the slope of the line and then selects the pixel lying at least a distance from the true line path at each sampling position.

To illustrate Bresenham's approach let us consider a line (L) with positive slope less than 1. So the line will be sampled at unit intervals in X direction. Assuming we have already determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot at next sampling position, i.e., at $x_k + 1$ grid line.

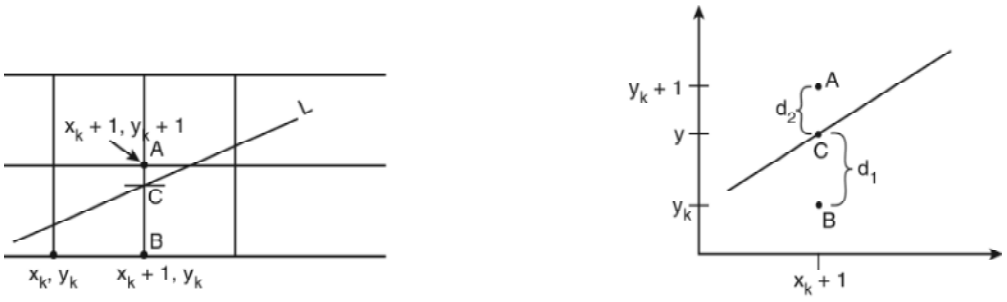


Figure 4.5: *Bresenhams Line Algorithm*

Our choices are clearly the pixels at $(x_k + 1, y_k + 1)$ and $(x_k + 1, y_k)$, i.e., A and B respectively in Figure 4.5. Let C be the intersection point of the line L with the gridline $x = x_k + 1$. In Bresenham's formulation, the difference between the vertical distances of A and B to C is computed, and the sign of the difference is used to select the pixel whose distance from C is smaller as the best approximation to the line.

Let the vertical distance of pixel B from the true line path, i.e., BC be denoted as d_1 and the vertical distance of pixel A from the true line path i.e. AC be denoted as d_2 at sampling position $x_k + 1$.

Now as C is a point on the true line path at sampling position $x_k + 1$ hence the coordinates (x, y) of C are given by

NOTES

$x = x_k + 1$

$y = m(x_k + 1) + c$ considering the equation of the line in the standard slope intercept form, $y = mx + c$

Then,

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

and,

$$d_2 = (y_k + 1) - y$$
$$= y_k + 1 - m(x_k + 1) - c$$

$\therefore$ The difference between these two distances is,

$$d_1 - d_2 = 2m(x_k + 1) - 2y_k + 2c - 1 \quad (3)$$

Substituting $m = \frac{\Delta y}{\Delta x}$, where Δy and Δx are the vertical and horizontal separations of the endpoint positions of the line, we get after rearranging

$$\Delta x(d_1 - d_2) = 2\Delta y(x_k) - 2\Delta x(y_k) + 2\Delta y + 2c\Delta x - \Delta x \quad (4)$$

Since $\Delta x > 0$ in this case, $\Delta x(d_1 - d_2)$ has the same sign as $(d_1 - d_2)$. Therefore $\Delta x(d_1 - d_2)$ which involves only integer calculations can be used as a decision parameter to decide the correct pixel, (x_{k+1}, y_{k+1}) next to (x_k, y_k).

We call this $\Delta x(d_1 - d_2)$ as parameter P_k.

If $P_k > 0$ then $d_1 > d_2$ hence choose A implying $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k + 1)$

If $P_k < 0$ then $d_1 < d_2$ hence choose B implying $(x_{k+1}, y_{k+1}) = (x_k + 1, y_k)$

If $P_k = 0$ then $d_1 = d_2$ hence we can choose either A or B.

Now what happens to the value of P_k and therefore the location of pixel (x_{k+2}, y_{k+2}) for the next grid line; both depend of course on whether we choose A or B as (x_{k+1}, y_{k+1}).

The parameter that decides the pixel (x_{k+2}, y_{k+2}) is P_{k+1} which is evaluated similarly from eqn. (4) as,

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

Subtracting P_k from P_{k+1} to get the incremental difference

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

If A is chosen previously then $y_{k+1} - y_k = 1$

And $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Else if B is chosen previously then $y_{k+1} - y_k = 0$

And $P_{k+1} = P_k + 2\Delta y$

Now checking the sign of P_{k+1} the choice is made between pixel at $(x_{k+1} + 1, y_{k+1} + 1)$ and $(x_{k+1} + 1, y_{k+1})$ as the pixel (x_{k+2}, y_{k+2}).

This process is continued till the endpoint of the line is reached.

Thus we see at each step the algorithm chooses between two pixels based on the sign

of the decision parameter calculated in the previous iteration; then it updates the decision parameter simply by incrementing the old value by a factor depending upon the choice of pixel.

This method of obtaining values of successive decision parameters (P_k) using incremental integer calculation avoids direct computation of P_k from eqn. (4) involving floating point operations.

NOTES

Since the first pixel is simply the first endpoint (x_0, y_0) of the line we can directly calculate the initial value of P_k i.e. P_0 for fixing (x_1, y_1).

As the point (x_0, y_0) lies on the true line path, it satisfies the line equation $y = mx + c$.

$$\Rightarrow c = y_0 - mx_0$$
$$= y_0 - \left(\frac{\Delta y}{\Delta x}\right)x_0$$
$$\Rightarrow 2c\Delta x = 2y_0\Delta x - 2x_0\,\Delta y$$

Putting $k = 0$ and replacing $2c\Delta x$ with $2y_0\Delta x - 2x_0\,\Delta y$ in the expression of P_k, we get

$$P_0 = 2\Delta y(x_0) - 2\Delta x(y_0) + 2\Delta y + 2y_0\Delta x - 2x_0\Delta y - \Delta x$$
$$= 2\Delta y - \Delta x$$

We can summarize Bresenham's line drawing algorithm for a line with positive slope less than 1 in the following steps.

Step 1 *Calculate horizontal separation Δx and vertical separation Δy of the given line endpoints. Plot the pixel (x_0, y_0), i.e., the starting endpoint.*

Step 2 *Calculate* $P_0 = 2\Delta y - \Delta x$

Step 3 *At each x_k along the line, starting at $k = 0$, check the sign of the decision parameter P_K. If $P_K < 0$ the next point to plot (x_{k+1}, y_{k+1}) is (x_{k+1}, y_k) and* $P_{K+1} = P_K + 2\Delta y$

Otherwise the next point to plot is (x_{k+1}, y_{k+1}) and $P_{k+1} = P_k + 2\Delta y - 2\Delta x$

Set $k = k + 1$

Step 4 *Repeat step 3 as long as $k < \Delta x$.*

Example 4.1 To digitalise a line from point (0, 2) to point (4, 5)

$$(x_0, y_0) = (0, 2)$$
$$\Delta y = 5 - 2 = 3$$
$$\Delta x = 4 - 0 = 4$$
$$\text{Slope } m = \left(\frac{5-2}{4-0}\right) = \frac{3}{4} < 1$$

Now calculate the successive decision parameters P_k and corresponding pixel positions (x_{k+1}, y_{k+1}) closest to the line path as follows.

NOTES

$$P_0 = 2\Delta y - \Delta x = 2 > 0$$
$$\Rightarrow x_1 = 1,\ y_1 = y_0 + 1 = 3$$
$$\therefore\ P_1 = P_0 + 2\Delta y - 2\Delta x$$
$$= 2 + 2(3) - 2(4)$$
$$= 0$$
$$\Rightarrow\ x_2 = 2, y_2 = y_1 + 1 = 4$$
$$\therefore\ P_2 = P_1 + 2\Delta y - 2\Delta x$$
$$= 0 + 2(3) - 2(4)$$
$$= -2 < 0$$
$$\Rightarrow x_3 = 3, y_3 = y_2 = 4$$
$$\therefore\ P_3 = P_2 + 2\Delta y$$
$$= -2 + 2(3)$$
$$= 4 > 0$$
$$\Rightarrow x_4 = 4, y_4 = y_3 + 1 = 5$$

We stop further calculation because the end point (4, 5) is reached.

The result is tabulated as follows:

k	P_k	coordinate of pixel to be plotted	
		x_{k+1}	y_{k+1}
		$x_0 = 0$	y_0 =2 start pixel
0	$P_0 = 2$	$x_1 = 1$	$y_1 = 3$
1	$P_1 = 0$	$x_2 = 2$	$y_2 = 4$
2	$P_2 = -2$	$x_3 = 3$	y_3 =4
3	$P_3 = 4$	$x_4 = 4$	y_4 = 5 end pixel

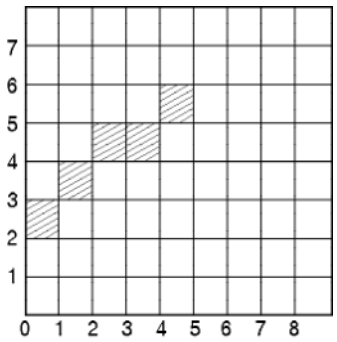


Figure 4.6: *The Shaded Pixels (x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4) – Representing the Digitalised Line, Between (0, 2) & (4, 5)*

Once again note that this version of Bresenham's algorithm works only for lines with slope between 0 and 1.

When the absolute magnitude of the slope of the line is greater than 1, we step along the direction of Y (instead of X) in unit steps and calculate successive x values using the Brenham's algorithm (refer Figure 4.7)

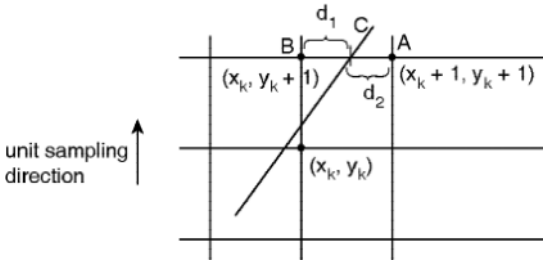


Figure 4.7

However, for negative slopes the procedure is similar except that now one coordinate decreases as the other increases from start point to the endpoint of such lines. Therefore the decreasing coordinate is incremented by –1 instead of +1 in Bresenham's logic.

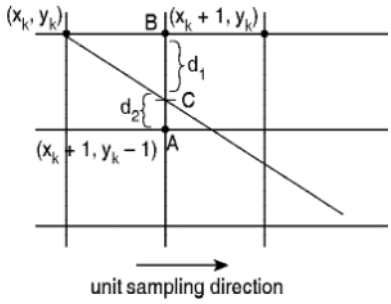


Figure 4.8: *They Coordinate Decreases from start point to end point.*

Taking into account all the above cases, a generalized version of Bresenham's algorithm is given below which will work for any line. The algorithm yields the same result even if we reverse the processing direction (start point to end point) for a given line.

4.3.3 Generalised Bresenham's Algorithm (Pseudocode)

Input The line endpoints (x_s, y_s) and (x_e, y_e) which are assumed to be not equal. All variables are assumed to be integers.

The sign ( ) function returns –1, 0, 1 as its argument is < 0, = 0, or > 0; sign $(k) = \dfrac{k}{\text{abs}(k)}$

$x = x_s$
$y = y_s$: initialise variables
$\Delta x = \text{abs}\,(x_e - x_s)$
$\Delta y = \text{abs}\,(y_e - y_s)$
$s_1 = \text{sign}\,(x_e - x_s)$
$s_2 = \text{sign}\,(y_e - y_s)$
if $\Delta y > \Delta x$ then
temp $= \Delta x$
$\Delta x = \Delta y$: swap Δx and Δy if absolute slope $|m| > 1$
$\Delta y =$ temp
swap $= 1$
else swap $= 0$: set the flag
end if
$n = 1$
$P = 2\Delta y - \Delta x$: initial value of decision parameter (P_0)
Setpixel $(x, y, 1)$: intensify the starting pixel (x_s, y_s)
while $(n \le \Delta x)$: till the endpoint is reached
if $P \ge 0$ then
$x = x + s_1$
$y = y + s_2$: choose the next pixel by incrementing x and y by + 1 or –1
$P = P + 2(\Delta y - \Delta x)$: update decision parameter
else : i.e., if $P < 0$
if swap = 1 then : i.e., | slope | > 1

NOTES

$y = y + s_2$: choose the next pixel by incrementing y by + 1 or –1 while keeping x same

else : i.e., if | slope | < 1

$x = x + s_1$: choose the next pixel by incrementing x by +1 or –1 while keeping y same

end if : y by +1 or –1 while keeping x same

$P = P + 2\Delta y$: update decision parameter

end if

Setpixel $(x, y, 1)$: set the next pixel

$n = n + 1$

end while : while loop end

If you are thinking how we have generalized the decision parameter's (P_{k+1}) expression, here we present the derivations for enthusiastic readers.

Considering $\left.\begin{matrix} s_1 = \text{sign } (x_e - x_s) \\ s_2 = \text{sign } (y_e - y_s) \end{matrix}\right\}$ and $\left.\begin{matrix} \Delta x = \text{abs}\,(x_e - x_s) \\ \Delta y = \text{abs } (y_e - y_s) \end{matrix}\right\}$

Slope $m = \dfrac{(y_e - y_s)}{(x_e - x_s)} = \dfrac{(s_2\,\Delta y)}{(s_1\,\Delta x)}$

Let $\Delta y < \Delta x,$ then sampling is along $\Delta x.$ Considering (x_k, y_k) as the pixel chosen already, we need to choose the next pixel $(x_{k+1},\ y_{k+1})$ between the two probables $(x_k \pm 1,\ y_k \pm 1)$, i.e., $(x_k + s_1,\ y_k + s_2)$ and $(x_k \pm 1,\ y_k)$ i.e. $(x_k + s_1,\ y_k)$. Let us also assume that the y coordinate of the point where the true line path intersects the gridline $x_k \pm 1$ (i.e., $x_k + s_1$) is y.

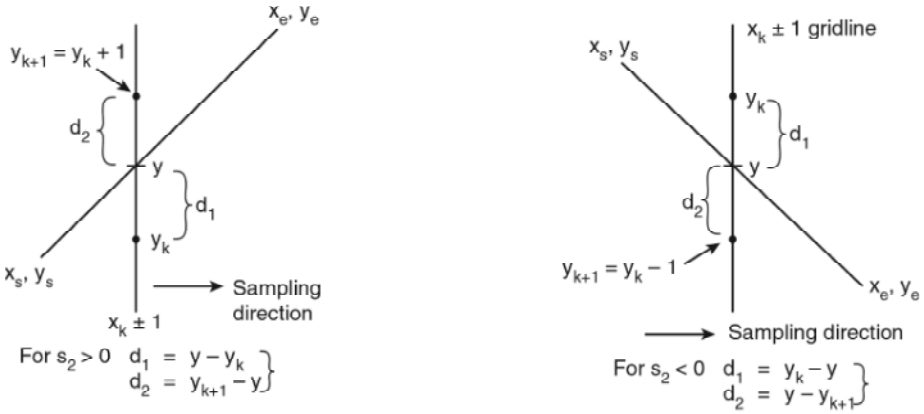


Figure 4.9

Then using the symbols as before we can write

$y = m(x_k + s_1) + c$

$d_1 = s_2(y - y_k)$

$d_2 = s_2(y_{k+1} - y)$

Replacing the value of y in the above expressions of d_1, d_2, we get

NOTES

$$\frac{d_1}{s_2} = m(x_k + s_1) + c - y_k$$

$$\frac{d_2}{s_2} = (y_k + s_2) - m(x_k + s_1) - c \qquad [\text{since } y_{k+1} = y_k + s_2]$$

$$\therefore \frac{(d_1 - d_2)}{s_2} = 2mx_k + 2ms_1 + 2c - 2y_k - s_2$$

$$= 2\frac{s_2 \Delta y}{s_1 \Delta x} x_k + 2\frac{s_2\, \Delta y}{s_1\, \Delta x} s_1 + 2c - 2y_k - s_2$$

$$\Rightarrow \frac{s_1}{s_2} \Delta x(d_1 - d_2) = (2\Delta y\ x_k)s_2 - 2(\Delta\, xy_k)s_1 + (2\Delta y)s_1 s_2 + (2c\Delta x)s_1 - (\Delta x) \cdot s_1 s_2$$

$$\Rightarrow P_k = \Delta x(d_1 - d_2) = (2\Delta y\, x_k)\ \frac{s_2^2}{s_1} - (2\Delta x\ y_k)s_2 + (2\Delta y)s_2^2 + (2c\Delta x)s_2 - (\Delta x)s_2^2$$

$$= \frac{2\Delta y\, x_k}{s_1} - (2\Delta x\, y_k)s_2 + (2\Delta y) + (2c\Delta x)s_2 - \Delta x$$

$$\therefore\ P_{k+1} = \frac{2\Delta y\, x_{k+1}}{s_1} - (2\Delta x\, y_{k+1})\, s_2 + (2\Delta y) + (2c\Delta x) - \Delta x \quad [\text{since } s_2^2 = 1]$$

$$\therefore\ P_{k+1} - P_k = \frac{2\Delta y}{s_1}(x_{k+1} - x_k) - 2\Delta x\, s_2 (y_{k+1} - y_k)$$

For $P_k \geq 0$ $\left.\begin{matrix} x_{k=1} = x_k + s_1 \\ y_{k+1} = y_k + s_2 \end{matrix}\right\}$

$$\Rightarrow P_{k+1} = P_k + (2\Delta y)\frac{s_1}{s_2} - (2\Delta x)(s_2)^2 = P_k + 2(\Delta y - \Delta x)$$

for $P_k < 0$ $\left.\begin{matrix} x_{k+1} = x_k + s_1 \\ y_{k+1} = y_k \end{matrix}\right\}$

$$\Rightarrow P_{k+1} = P_k + (2\Delta y)\frac{s_1}{s_1} - (2\Delta x)\, s_2 (0) = P_k + 2\Delta y$$

Putting $m = \frac{s_2\, \Delta y}{s_1\, \Delta x}$, $y = y_s$ and $x = x_s$ in $y = mx + c$

we get, $(2c\Delta x) = (2\Delta x\, y_s) - (2\Delta y\, x_s)\frac{s_2}{s_1}$

Now replacing this value of $(2c\ \Delta x)$ in the expression of P_k and putting $k = 0$, we get the initial value of decision parameter.

$$P_0 = \frac{2\Delta y\, x_0}{s_1} - (2\Delta x\, y_0)s_2 + 2\Delta y + (2\Delta x\, y_s)s_2 - (2\Delta y\, x_s)\frac{s_2^2}{s_1} - \Delta x$$

$$= 2\Delta y - \Delta x \left[\begin{matrix} \text{since, } x_0 = x_s \\ y_0 = y_s \end{matrix}\right\} \text{and } s_2^2 = 1 \Bigg]$$

In a similar manner, the algorithm can be proved for $\Delta y > \Delta x$ case where sampling is done along Δy and $P_0 = 2\Delta x - \Delta y$

NOTES

For $P_k \geq 0$, $x_{k+1} = x_k + s_1$
$y_{k+1} = y_k + s_2$ and
$P_{k+1} = P_k + 2\Delta x - 2\Delta y$

For $P_k < 0$, $y_{k+1} = y_k + s_2$
$x_{k+1} = x_k$ and
$P_{k+1} = P_k + 2\Delta x$

Example 4.2

$xstart = 0 \quad ystart = 0$

$Xend = -4 \quad yend = -8$

To find out using generalised Bresenham's algorithm the pixel locations approximating a line between the given points.

$\Delta x = \text{abs}(-4 - 0) = 4$

$\Delta y = \text{abs}(-8 - 0) = 8$

$s_1 = \text{sign}\,(-4 - 0) = -1,\ s_2 = \text{sign}\ (-8 - 0) = -1$

since $\Delta y > \Delta x$ i.e. slope $= \dfrac{\Delta y}{\Delta x} = \dfrac{8}{4} = 2 > 1$

Δy and Δx are interchanged i.e. $\Delta x = 8$ and $\Delta y = 4$

and flag swap $= 1$

$n = 1$

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

$= 2 \times 4 - 8 = 0$ [since swap $= 1$]

Setpixel $(0, 0, 1)$

since $n \leq (\Delta x = 8)$

Since $P = 0$

$x = x + s_1 = 0 + (-1) = -1$

$y = y + s_2 = 0 + (-1) = -1$

$P = P + 2(\Delta y - \Delta x)$

$= 0 + 2(4 - 8) = -8$

Setpixel $(-1, -1, 1)$

$n = 1 + 1 = 2$

Since $n < 8$

since $(P = -8) < 0$ and swap $= 1$ so

$y = y + s_2$

$= -1 + (-1) = -2$

$P = P + 2\Delta y$

$= -8 + 2 \times 4 = 0$

Setpixel $(-1, -2, 1) = 0$ [x remains unchanged as -1 in previous iteration]

$n = 2 + 1 = 3$

Since $n < 8$

Since $P = 0$

$x = x + s_1 = -1 - 1 = -2$

$y = y + s_2 = -2 - 1 = -3$

$P = P + 2(\Delta y - \Delta x)$

$= 0 + 2(4 - 8) = -8$

NOTES

Setpixel $(-2, -3, 1)$

$n = 3 + 1 = 4$

Since $n < 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -3 - 1 = -4$

$P = P + 2\Delta y$

$= -8 + 2 \times 4 = 0$

Setpixel $(-2, -4, 1)$ [x remains unchanged as -2 in previous iteration]

$n = 4 + 1 = 5$

Since $n < 8$

Since $P = 0$

$x = x + s_1 = -2 - 1 = -3$

$y = y + s_2 = -4 - 1 = -5$

$P = P + 2(\Delta y - \Delta x) = 0 + 2(4 - 8) = -8$

Setpixel $(-3, -5, 1)$

$n = n + 1 = 5 = 1 = 6$

Since $n < 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -5 - 1 = -6$

$P = P + 2\Delta y = -8 + 2 \times 4 = 0$

Setpixel $(-3, -6, 1)$ [$x = -3$ unchanged]

$n = n + 1 = 6 + 1 = 7$

Since $n < 8$

Since $P = 0$ so

$x = x + s_1 = -3 - 1 = -4$

$y = y + s_2 = -6 - 1 = -7$

$P = P + 2(\Delta y - \Delta x) = 0 + 2(4 - 8) = -8$

Setpixel $(-4, -7, 1)$

$n = n + 1 = 7 + 1 = 8$

Since $n = 8$

Since $(P = -8) < 0$ and swap $= 1$

$y = y + s_2 = -7 - 1 = -8$

$P = P + 2\Delta y = -8 + 2 \times 4 = 0$

Setpixel $(-4, -8, 1)$ [$x = -4$ unchanged]

$n = n + 1 = 8 + 1 = 9$

Since $n > 8$ hence no more iteration is done.

4.4 SCAN CONVERTING CIRCLE

4.4.1 Polynomial Method

A circle can be represented mathematically by the second order polynomial equation $(x - x_c)^2 + (y - y_c)^2 = r^2$ where (x_c, y_c) is the centre of the circle and r the radius. The equation can be solved for y as

NOTES

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \quad (5)$$

To draw a circle, the value of x is incremented in units of one from $x_c - r$ to $x_c + r$, eqn. (5) is used to solve for two values of y for each step, the resulting locations are converted from float to integer type and points are plotted. One problem with this method is that it involves time-consuming multiplications and square-root operations at each step. Furthermore, the spacing between plotted pixel positions will not be uniform; the spacing is larger for values of x closer to $x_c - r$ and $x_c + r$, because the slope of the circle become infinite there.

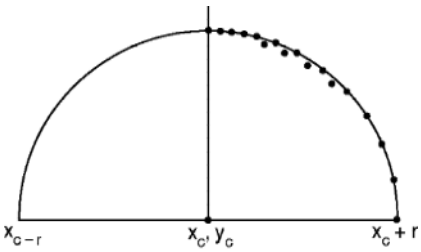


Figure 4.10: *Scan Converting Circle*

However, we can improve the drawing process by taking advantage of the symmetry in a circle. Consider a circle centred at the origin. If a point $P(x, y)$ is generated on the circle (from the polynomial expression $y = \sqrt{r^2 - x^2}$), then simultaneously we can generate corresponding seven other circle points simply from natural 8-way symmetry without requiring any real computation as such. (refer Figure 4.11) Thus we can generate all pixel positions around a circle by calculating the points only on one 45° sector, say, from $x = 0$ to $x = y$.

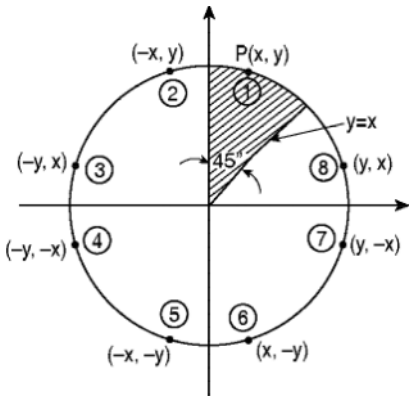


Figure 4.11: *Computation of a Circle-point ① in shaded octant yields symmetric points ② to ⑧ in other seven octants; Point ⑧ is the image of ① w.r.t. y = x, ② and ③ are images of ① abd ⑧ respectively w.r.t. Y axis, ④, ⑤, ⑥, ⑦ are images of ③, ②, ①, ⑧ respectively w.r.t. X axis*

4.4.2 Parametric Method

The parametric polar representation of a circle having centre at (x_c, y_c) and radius r is

$$x = x_c + r\cos\theta\,;\; y = y_c + r\sin\theta$$

where θ is measured in radians from 0 to 2π.

The method of incremental drawing can be adopted here to obtain the next point to be displayed in terms of the previous point.

Let (x_k, y_k) and (x_{k+1}, y_{k+1}) be two consecutive points on a origin centered circle and they are related by

$$x_k = r\cos\theta;\; y_k = r\sin\theta$$
$$x_{k+1} = r\cos(\theta + \Delta\theta);\; y_{k+1} = r\sin(\theta + \Delta\theta)$$

Here $\Delta\theta$ is angular increment. From trigonometry we get

$$x_{k+1} = r\cos\theta\cos(\Delta\theta) - r\sin\theta\sin(\Delta\theta)$$
$$y_{k+1} = r\sin\theta\cos(\Delta\theta) + r\cos\theta\sin(\Delta\theta)$$
$$\Rightarrow x_{k+1} = x_k\cos(\Delta\theta) - y_k\sin(\Delta\theta)$$
$$y_{k+1} = y_k\cos(\Delta\theta) + x_k\sin(\Delta\theta)$$

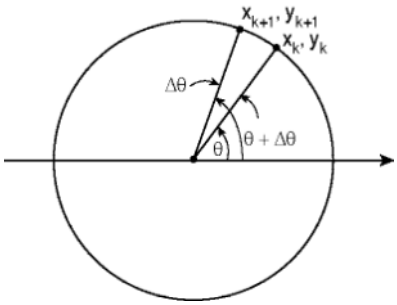


Figure 4.12: *Parametric Method*

Once the step size (angle increment) $\Delta\theta$ is fixed, to generate any octant of the circle, $\cos(\Delta\theta)$ and $\sin(\Delta\theta)$ has to be calculated only once . Using the property of symmetry seven other octants of the circle can be simultaneously generated without carrying out any computation.

As arc lengths (s) are proportional to θ ($s = r\theta$) fixed increment of θ i.e. $\Delta\theta$ results in equal spacing (Δs) between successively plotted points. To make this spacing equal to approximately one unit, the arc length for every step has to be taken as one unit, i.e. $\Delta s = r\Delta\theta = 1$. Then $\Delta\theta$ becomes $1/r$.

The following algorithm scan converts a circle (x_c, y_c, r) parametrically.

Input : x_c, y_c, r : centre = (x_c, y_c) radius = r

$\Delta\theta = 1/r$

$c = \cos(\Delta\theta)$; $s = \sin(\Delta\theta)$: calculate cos ($\Delta\theta$), sin ($\Delta\theta$) only once

$x = 0$
$y = r$: the top quadrant point of the circle

while ($y > x$) : continue the loop until the first octant ends at $y = x$

Setpixel (Round ($x_c + x$), Round ($y_c + y$), 1)* : generate a circle point in the 1st octant
Setpixel (Round ($x_c - x$), Round ($y_c + y$), 1)
Setpixel (Round ($x_c + x$), Round ($y_c - y$), 1)
Setpixel (Round ($x_c - x$), Round ($y_c - y$), 1) : generate seven other circle points from
Setpixel (Round (x_c + y), Round ($y_c + x$), 1) symmetry
Setpixel (Round ($x_c - y$), Round ($y_c + x$), 1)
Setpixel (Round ($x_c + y$), Round ($y_c - x$), 1)
Setpixel (Round ($x_c - y$), Round ($y_c - x$), 1)
x temp = *x*
x = *xc* – *ys* : calculate the next point in the first octant
y = *yc* + *xtemp*s
endwhile

4.4.3 Bresenham's Method

As with Bresenham's line generation algorithm, the sign of a decision parameter is checked for finding the closest pixel to the circumference of a circle at each sampling step in Bresenham's circle rasterizing algorithm.This algorithm is better than the previous two algorithms in (Polynomial and Parametric method) because it avoids trigonometric and square root calculation by adopting only integer operation involving squares of the pixel separation distances. For a given radius R and screen centre position (x_c, y_c), we can first

NOTES

NOTES

calculate pixel positions around a circle path centred at the origin (0, 0).Then each calculated position (x, y) is moved to its proper screen position by adding x_c to x and y_c to y, as done in Parametric method. Keeping in mind the advantage of generating a complete circle by only computing points on a single octant, we consider the first octant of an origin centred circle.Notice that, if the algorithm begins at $x = 0, y = R$, then for clockwise generation of the circle y is a monotonically decreasing function of x in the first quadrant. For any known point (x_k, y_k) on the circle, for clockwise generation of the circle there are only three possible candidate pixel for selection as the next pixel which best represents the circle: (1) adjacent pixel horizontally to the right –H (x_k+1, y_k), (2) adjacent pixel diagonally downward to the right –D (x_k+1, y_k-1) and (3) adjacent pixel vertically downward –V (x_k, y_k-1). Out of these three pixels the algorithm chooses the one for which the distance from the true circle is minimum.

Now consider the pixel P w.r.t the circular arc shown in Figure 4.13.

Let d_P denote the quantity $|OP^2 - OX^2|$. In this case $d_P = |OP^2 - OX^2|$

$$= x^2 + y^2 - R^2$$

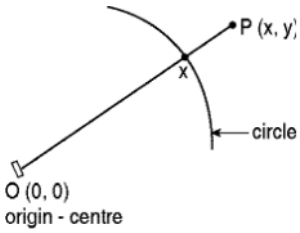


Figure 4.13: *Bresenham's Method*

Similarly if we introduce the terms d_H, d_D and d_V corresponding to pixels H, D and V respectively then we can write.

$$d_H = |OH^2 - R^2| = |(x_k + 1)^2 + (y_k)^2 - R^2|$$

$$d_D = |OD^2 - R^2| = |(x_k + 1)^2 + (y_k - 1)^2 - R^2|$$

$$d_V = |OV^2 - R^2| = |(x_k)^2 + (y_k - 1)^2 - R^2|$$

Though the (x_{k+1}, y_{k+1}) pixel has to be chosen among (x_k+1, y_k) , $(x_k + 1, y_k-1)$ and (x_k, y_k-1), there are five possible cases to be considered regarding position of the true circle path in the vicinity of the point (x_k, y_k) as illustrated in Figure 4.14.

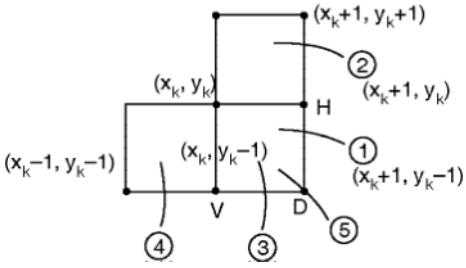


Figure 4.14

It is clear from Figure 4.14 that

For **Case** (1) & (2) the diagonal pixel D is inside the circle, implying

$$OD^2 < R^2, \text{ i.e., } OD^2 - R^2 < 0$$

For **Case** (3) & (4) the diagonal pixel D is outside the circle, implying

$$OD^2 > R^2, \text{ i.e., } OD^2 - R^2 > 0$$

For **Case** (5) the diagonal pixel D is on the circle, implying

$$OD^2 = R^2, \text{ i.e., } OD^2 - R^2 = 0$$

Let us now study the five cases separately while referring to the Figure 4.14

Case 1: *To choose between $H(x_k + 1, y_k)$ and $D(x_k + 1, y_k - 1)$*

Let $\delta_{HD} = d_H - d_D$

$$\therefore \quad \delta_{HD} = |OH^2 - R^2| - |OD^2 - R^2|$$

$$= (OH^2 - R^2) - (R^2 - OD^2) \quad \text{[since, } OD^2 < R^2 \; OH^2 > R^2 \text{ as } D \text{ is inside the circle while } H \text{ is outside the circle]}$$

$$= OH^2 + OD^2 - 2R^2$$

$$= \{(x_{k+1} + 1)^2 + (y_k)^2\} + \{(x_k + 1)^2 + (y_k - 1)^2\} - 2R^2$$

$$= 2(x_k + 1)^2 + 2(y_k - 1)^2 - 2R^2 + 2y_k - 1$$

$$= 2\{(x_k + 1)^2 + (y_k - 1)^2 - R^2\} + 2y_k - 1$$

$$= 2\Delta D_k + 2y_k - 1$$

where $\Delta D_k = OD^2 - R^2 = (x_k + 1)^2 + (y_k - 1)^2 - R^2$

If $\delta_{HD} < 0$ then $d_H < d_D$ which implies the horizontal pixel is closer to the actual circle than the diagonal pixel. So choose $H(x_k + 1, y_k)$

Conversely if $\delta_{HD} > 0$ then choose $D(x_k + 1, y_k - 1)$

However, if $\delta_{HD} = 0$ then the distances are equal; conventionally choose H.

Case 2: Since y is a monotonically decreasing function of x along the clockwise generated circle-path in first quadrant we cannot choose $(x_k + 1, y_k + 1)$ as the pixel next to (x_k, y_k) even if it is closer to the circle than $H(x_k + 1, y_k)$. So for Case(2) always choose the horizontal pixel $H(x_k + 1, y_k)$ as $(k + 1)$ thpixel.

Also note that $\delta_{HD} < 0$ for this case.

Case 3: *To choose between $V(x_k, y_k - 1)$ and $D(x_k + 1, y_k - 1)$*

Let, $\delta_{VD} = d_V - d_D$

$$\therefore \quad \delta_{VD} = |OV^2 - R^2| - |OD^2 - R^2|$$

$$= (R^2 - OV^2) - (OD^2 - R^2) \quad \text{[since, } OD^2 > R^2, OV^2 < R^2 \text{ as } D \text{ is outside the circle while } V \text{ is inside the circle]}$$

$$= 2R^2 - OV^2 - OD^2$$

$$= 2R^2 - \{(x_k)^2 + (y_k - 1)^2\} - \{(x_k + 1)^2 + (y_k - 1)^2\}$$

$$= 2R^2 - 2(x_k + 1)^2 - 2(y_k - 1)^2 + 2x_k + 1$$

$$= 2x_k + 1 - 2\{(x_k + 1)^2 + (y_k - 1)^2 - R^2\}$$

$$= 2x_k - 2\Delta D_k + 1 \quad \text{[since, } \Delta D_k = (x_k + 1)^2 + (y_k - 1)^2 - R^2]$$

If $\delta_{Vd} < 0 \Rightarrow d_V < d_D$ hence choose $V(x_k, y_k - 1)$

else if $\delta_{VD} < 0 \Rightarrow d_V > d_D$ hence choose $D(x_k + 1, y_k - 1)$

else if $\delta_{VD} < 0 \Rightarrow d_V = d_D$ then conventionally choose D.

Case 4: Since along the circle y monotonically decreases and x monotonically increases, we cannot choose the pixel $(x_k - 1, y_k - 1)$ with lower x-value than the preceding pixel (x_k, y_k), even if it is closer to the circle than the vertical pixel $V(x_k, y_k - 1)$. So our choice is V as the $(k + 1)$th pixel.

Also note that $\delta_{VD} < 0$ for this case.

NOTES

NOTES

Case 5: This is the case when the diagonal pixel D lies on the actual circle. So the obvious choice is $D(x_k+1, y_k-1)$.

Also note that for this case,

$$\delta_{HD} = OH^2 - OR^2 > 0 \quad [\text{since, } OD^2 = R^2 \ \& \ OH^2 > R^2]$$
$$\delta_{VD} = R^2 - OV^2 > 0 \quad [\text{since, } OD^2 = R^2 \ \& \ R^2 > OV^2]$$

We will call ΔD as the *decision parameter* because by checking the sign (<0, >0, or, = 0) of $\Delta D = OD^2 - R^2$ at each step, we can determine the cases(s) it corresponds to and proceed accordingly.

We can update the value of decision parameter ΔD at every step, following the incremental approach, i.e. by adding a factor (depending on the choice of pixel) to the old value.

1. If $x_{k+1} = x_k + 1,\ y_{k+1} = y_k,$ i.e. the horizontal pixel H

$$\begin{aligned}\Delta D_{k+1} &= (x_{k+1}+1)^2 + (y_{k+1}-1)^2 - R^2 \quad [\text{since, } \Delta D_k = (x_k+1)^2 + (y_k-1)^2 - R^2] \\ &= (x_k+2)^2 + (y_k-1)^2 - R^2 \\ &= (x_k+1)^2 + (y_k-1)^2 - R^2 + 2x_{k+1} + 1 \\ &= \Delta D_k + 2x_{k+1} + 1\end{aligned}$$

2. If $x_{k+1} = x_k + 1,\ y_{k+1} = y_k - 1,$ i.e. the diagonal pixel D

$$\begin{aligned}\Delta D_{k+1} &= (x_k+2)^2 + (y_k-2)^2 - R^2 \\ &= (x_k+1)^2 + (y_k-1)^2 - R^2 + 2x_{k+1} - 2y_{k+1} + 2 \\ &= \Delta D_k + 2x_{k+1} - 2y_{k+1} + 2\end{aligned}$$

3. If $x_{k+1} = x_k,\ y_{k+1} = y_k - 1,$ i.e. the vertical pixel D

$$\begin{aligned}\Delta D_{k+1} &= (x_k+1)^2 + (y_k-2)^2 - R^2 \\ &= (x_k+1)^2 + (y_k-1)^2 - R^2 - 2y_{k+1} + 1 \\ &= \Delta D_k - 2y_{k+1} + 1\end{aligned}$$

The pseudocode summarizing the above steps is given below.

4.4.4 Bresenham's Algorithm (Pseudocode)

The centre of the circle and the starting point are assumed to be located precisely at pixel elements. The radius is also assumed to be an integer.

Input : $x_c, y_c\ R$

$x = 0,\ y = R$: the starting pixel (x_0, y_0)

$\Delta D = 2(1-R)$: initial value of decision parameter

$\Delta D_0 = (0+1)^2 + (R-1)^2 - R^2 = 2 - 2R$

while $(y > x)$: continue till the diagonal axis in the first quadrant is reached

Setpixel $((x_c + x), (y_c + y), 1)$: plot the computed pixel in the 1st octant

Setpixel $((x_c - x), (y_c + y), 1)$

Setpixel $((x_c + x), (y_c - y), 1)$
Setpixel $((x_c - x), (y_c - y), 1)$
Setpixel $((x_c + y), (y_c + x), 1)$: plot the symmetric pixels in the other octants
Setpixel $((x_c - y), (y_c + x), 1)$
Setpixel $((x_c + y), (y_c - x), 1)$
Setpixel $((x_c - y), (y_c - x), 1)$
if $\Delta D < 0$ then : $OD^2 < R^2$
$\delta = 2\Delta D + 2y - 1$: $\delta = \delta_{HD}$
if $\delta \leq 0$ then
$x = x + 1$: choose horizontal pixel and update ΔD accordingly
$\Delta D = \Delta D + 2x + 1$
else
$x = x + 1$
$y = y - 1$: choose diagonal pixel and update ΔD accordingly for
$\Delta D = \Delta D + 2x - 2y + 2$ $\delta_{HD} > 0$
endif
elseif $\Delta D > 0$ then : $OD^2 > R^2$
$\delta = 2x - 2\Delta D + 1$: $\delta = \delta_{VD}$
if $\delta < 0$ then
$y = y - 1$: choose vertical pixel and update ΔD accordingly
$\Delta D = \Delta D - 2y + 1$
else
$x = x + 1$: choose diagonal pixel and update ΔD accordingly
$y = y - 1$ for $\delta_{VD} \geq 0$
$\Delta D = \Delta D + 2x - 2y + 2$
endif
else
$x = x + 1$
$y = y - 1$: choose diagonal pixel and update ΔD accordingly
$\Delta D = \Delta D + 2x - 2y + 2$ for $\Delta D = 0$
endif
endwhile

Example 4.3 To find out (using Bresenham's algorithm) the pixel location approximating the first octant of a circle having centre at (4, 5) and radius 4.

$$x_c = 4,\ y_c = 5,\ R = 4$$
$$x = 0$$
$$y = R = 4$$
$$\Delta D = 2(1 - R) = 2(1 - 4) = -6$$

NOTES

NOTES

Iteration 1 Since $(y = 4) > (x = 0)$

Setpixel $((x_c + x), (y_c + y),1) \Rightarrow$ Setpixel $(4, -9,1)$

Since $(\Delta D = -6) < 0$

$\delta = \delta_{HD} = 2\Delta D + 2y - 1 = 2(-6) + 2(4) - 1 = -5$

since $\delta < 0$

$x = x + 1 = 0 + 1 = 1$

$y = 4$ (unchanged)

$\Delta D = \Delta D + 2x + 1 = -6 + 2(1) + 1 = -3$

Iteration 2 Since $(y = 4) > (x = 1)$

Setpixel $((x_c + x), (y_c + y),1) \Rightarrow$ Setpixel $(5, 9,1)$

Since $(\Delta D = -3) < 0$

$\delta = \delta_{HD} = 2\Delta D + 2y - 1 = 2(-3) + 2(4) - 1 = 1$

since $\delta > 0$

$x = x + 1 = 1 + 1 = 2$

$y = y - 1 = 4 - 1 = 3$

$\Delta D = \Delta D + 2x - 2y + 2 = -3 + 2(2) - 2(3) + 2 = -3$

Iteration 3 Since $(y = 3) > (x = 2)$

Setpixel $((x_c + x), (y_c + y),1) \Rightarrow$ Setpixel $(6, 8,1)$

Since $(\Delta D = -3) < 0$

$\delta = \delta_{HD} = 2\Delta D + 2y - 1 = 2(-3) + 2(3) - 1 = -1$

since $\delta < 0$

$x = x + 1 = 2 + 1 = 3$

$y = 3$ (unchanged)

$\Delta D = \Delta D + 2x + 1 = -3 + 2(3) + 1 = 4$

Iteration 4 Since $(y = 3) = (x = 3)$

Setpixel $((x_c + x), (y_c + y),1) \Rightarrow$ Setpixel $(7, 8,1)$

Since $(\Delta D = 4) > 0$

$\delta = \delta_{VD} = 2x - 2\Delta D + 1 = 2(3) - 2(4) + 1 = -1$

since $\delta < 0$

$y = y - 1 = 3 - 1 = 2$

$x = 3$ (unchanged)

$\Delta D = \Delta D - 2y + 1 = 4 - 2(2) + 1 = 1$

Iteration 5 Since $(y = 2) \not\geq (x = 3)$ iteration 5 doesn't run. So the pixel locations in the first octant of the given circle are (4, 9), (5, 9), (6, 8) and (7, 8).

4.5 SCAN CONVERTING ELLIPSE

4.5.1 Midpoint Method for Generation of Ellipse

For simplicity, ellipse having center at origin and axes (major and minor) parallel to the coordinate axes is considered. The algebraic expression of such an ellipse is,

$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ where $2a$ = length of major axis

$2b$ = length of minor axis

$\Rightarrow b^2x^2 + a^2y^2 - a^2b^2 = 0$

From coordinate geometry, we get

$$f(x,y) = b^2x^2 + a^2y^2 - a^2b^2 \begin{cases} < 0 \text{ implies } (x,y) \text{ inside the ellipse} \\ = 0 \text{ implies } (x,y) \text{ on the ellipse} \\ > 0 \text{ implies } (x,y) \text{ outside the ellipse} \end{cases}$$

Now an ellipse can be divided equally into four parts. So if one part (or quadrant) can be generated then the other three parts can easily be replicated by mirroring the original part (4-way symmetry).

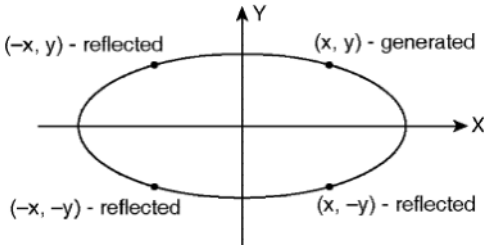


Figure 4.15: *Scan Converting Ellipse*

Let us generate the 1st quadrant of the ellipse. For applying the Midpoint method the 1st quadrant is logically divided into two regions –

Region 1 – arc closer to the Y axis with absolute slope less than 1.

Region 2 – arc closer to the X axis with absolute slope greater than 1.

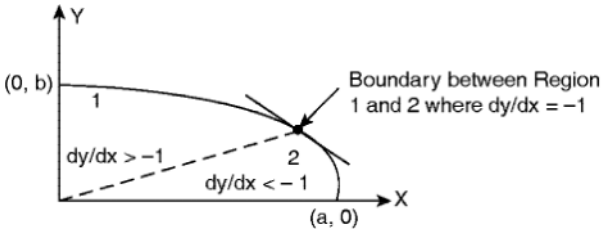


Figure 4.16

Provided we know coordinates (x_i, y_i) of the pixel that lies exactly on the ellipse on the first quadrant we need to find out the next nearest pixel (x_{i+1}, y_{i+1}) using incremental integer value of the decision parameter $f(x, y)$. As in the case of rasterizing a line we changed the unit sampling direction (X or Y) according to the slope here also we take the slope factor in mind. Starting at $(0, b)$ and moving clockwise along the ellipse-path in Region 1 we take unit steps in X direction until we reach the boundary between Region 1 and 2. Then we switch to unit steps in Y direction while sampling Region 2 of the curve.

The partial derivative of $f(x, y)$ w.r.t x and that w.r.t y being $f_x = 2b^2x$ and $f_y = 2a^2y$ respectively. The slope of the ellipse at any point (x, y) is given by

$$\frac{dy}{dx} = -\frac{f_x}{f_y} = -\frac{b^2x}{a^2y}$$

NOTES

NOTES

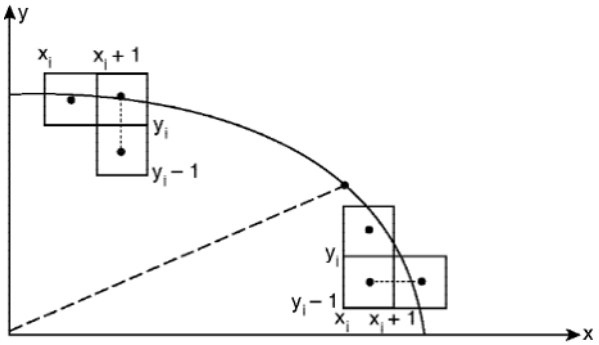


Figure 4.17

While deciding whether to plot $(x_i + 1, y_i)$ or $(x_i + 1, y_i - 1)$ as the $(i + 1)$th pixel, the choice is easily made by checking whether the halfway point (midpoint) between the centers of the two candidate pixels lies inside or outside the theoretical ellipse-path.

The checking is done by evaluating $f(x_i + 1, y_i - 1/2)$ and checking the sign.

Let, $$P_i = f\left(x_i + 1, \quad y_i - \frac{1}{2}\right) = b^2(x_i + 1)^2 + a^2\left(y_i - \frac{1}{2}\right)^2 - a^2b^2$$

Similarly, $$P_{i+1} = f\left(x_{i+1} + 1, \quad y_{i+1} - \frac{1}{2}\right) = b^2(x_{i+1} + 1)^2 + a^2\left(y_{i+1} - \frac{1}{2}\right)^2 - a^2b^2$$

$$\Rightarrow P_{i+1} - P_i = b^2[(x_{i+1} + 1)^2 - (x_i + 1)^2] + a^2\left[\left(y_{i+1} - \frac{1}{2}\right)^2 - \left(y_i - \frac{1}{2}\right)^2\right]$$

$$\Rightarrow P_{i+1} = P_i + b^2[(x_{i+1} + 1)^2 - (x_i + 1)^2] + a^2\left[\left(y_{i+1} - \frac{1}{2}\right)^2 - \left(y_i - \frac{1}{2}\right)^2\right]$$

If $P_i < 0$, midpoint is inside ellipse, hence $y_{i+1} = y_i$

$\therefore P_{i+1} = P_i + 2b^2 x_{i+1} + b^2$, where $x_{i+1} = x_i + 1$

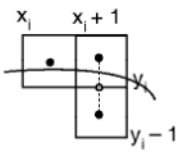
If $P_i \geq 0$, midpoint is outside or on ellipse, hence $y_{i+1} = y_i - 1$

$\therefore P_{i+1} = P_i + 2b^2 x_{i+1} + b^2 - 2a^2 y_{i+1}$, where $x_{i+1} = x_i + 1$

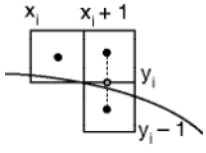
The starting value of P_i is obtained by putting $x_i = 0$, $y_i = b$ say for $i = 1$

$\therefore P_{i=1} = b^2 + a^2 (b - 1/2)^2 - a^2b^2 = b^2 - a^2b + a^2/4$

The value of f_x and f_y is calculated at every sampled point (x_i, y_i) as $f_x = 2b^2 x_i, f_y = 2a^2 y_i$. These two parameters keep track of the slope of the ellipse at each point and as long as $f_x < f_y$ we are in Region 1; the incremental formulae followed for calculating decision parameter and corresponding pixel coordinates are as derived above.



As soon as we reach the boundary between Region 1 and 2 (i.e., when $f_x = f_y$) we start sampling at unit steps in the (–)ve Y direction and the midpoint is now taken between two horizontal pixels. The choice between the candidate pixels $(x_j, y_j - 1)$ and $(x_j + 1, y_j - 1)$ is made by evaluating and checking the sign of f(candidate-midpoint), i.e., $f(x_j + 1/2, y_j - 1)$.



NOTES

Let $P_j = f\left(x_j + \frac{1}{2}, y_j\right) = b^2\left(x_j + \frac{1}{2}\right)^2 + a^2(y_j - 1)^2 - a^2 b^2$

Similarly, $P_{j+1} = f\left(x_{j+1} + \frac{1}{2}, y_{j+1} - 1\right) = b^2\left(x_{j+1} + \frac{1}{2}\right)^2 + a^2(y_{j+1} - 1)^2 - a^2 b^2$

$$\Rightarrow P_{j+1} - P_j = b^2\left[\left(x_{j+1} + \frac{1}{2}\right)^2 - \left(x_j + \frac{1}{2}\right)^2\right] + a^2\left[(y_{j+1} - 1)^2 - (y_j - 1)^2\right]$$

$$\Rightarrow P_{j+1} = P_j + b^2\left[\left(x_{j+1} + \frac{1}{2}\right)^2 - \left(x_j + \frac{1}{2}\right)^2\right] + a^2\left[(y_{j+1} - 1)^2 - (y_i - 1)^2\right]$$

If $P_j < 0$, midpoint is inside ellipse, hence $x_{j+1} = x_j + 1$

$\therefore P_{j+1} = P_j + 2b^2 x_{j+1} - 2a^2 y_{j+1} + a^2$, where $y_{j+1} = y_j - 1$

If $P_j \geq 0$, midpoint is outside or on ellipse, hence $x_{j+1} = x_j$

$\therefore P_{j+1} = P_j - 2a^2 y_{j+1} + a^2$ where $y_{j+1} = y_j - 1$

The starting value of P_j is obtained using the last point (x_i, y_i), calculated in Region 1.

$\therefore P_{j=1} = b^2 (x_i + 1/2)^2 + a^2 (y_i - 1)^2 - a^2 b^2$

4.5.2 Midpoint Algorithm (Pseudocode)

Input: a, b, x_c, y_c : semi-major axis – a, semi-minor axis – b, center = (x_c, y_c)

Initialize : $x = 0, y = b$

$f_x = 0, f_y = 2a^2 b$

$P = b^2 - a^2 b + a^2/4$: first value of decision parameter in Region 1

Setpixel $((x_c + x), (y_c + y), 1)$: plot the 4 symmetric pixels corresponding to $x = 0, y = b$

Setpixel $((x_c - x), (y_c + y), 1)$

Setpixel $((x_c + x), (y_c - y), 1)$

Setpixel $((x_c - x), (y_c - y), 1)$

while $(f_x < f_y)\{$: continue until Region 1 ends at $f_x = f_y$

$x = x + 1$: unit sampling direction in Region 1 is (+)ve X

$f_x = f_x + 2b^2$: $f_x = 2b^2 x$; for $x = x + 1, f_x = 2b^2(x+1) = f_x + 2b^2$

if $(P \geq 0)$ then

$y = y - 1$: for $P \geq 0$ in Region 1, $x = x + 1, y = y - 1$

$f_y = f_y - 2a^2$: $f_y = 2a^2 y$; when $y = y - 1, f_y = 2a^2(y - 1) = f_y - 2a^2$

end if

if $(P < 0)$ then : for $P < 0$, $x = x + 1$, but no change in y

$P = P + b^2 + f_x$: for $P < 0$, $P = P + b^2 + 2b^2 (x + 1)$

else

$P = P + b^2 + f_x - f_y$: for $P \geq 0$, $P = P + b^2 + 2b^2 (x + 1) - 2a^2 (y - 1)$

end if

Setpixel $((x_c + x), (y_c + y), 1)$: plot the 4 symmetric pixels corresponding to

Setpixel $((x_c - x), (y_c + y), 1)$: each sampled pixel in Region 1

Setpixel $((x_c + x), (y_c - y), 1)$

Setpixel $((x_c - x), (y_c - y), 1)$

end while : End of generating Region 1 and 3 symmetric regions

NOTES

```
P = b^2 (x + 1/2)^2 + a^2 (y – 1)^2 – a^2 b^2: Update the decision parameter before
                                              : entering Region 2
while (y > 0)                                 : In Region 2
        y = y – 1                             : Unit sampling direction in Region 2 is (–)ve Y
        f_y = f_y – 2a^2
          if (P < 0) then
              x = x + 1                       : for P < 0 in Region 2, x = x + 1, y = y – 1
              f_x = f_x + 2b^2
          end if
        if (P ≥ 0) then                       : for P ≥ 0, y = y – 1, but no change in x
            P = P + a^2 – f_y                 : for P ≥ 0, P = P + a^2 – 2a^2 (y – 1)
      else
            P = P + a^2 – f_y + f_x           : for P < 0, P = P + a^2 – 2a^2 (y – 1) + 2b^2 (x + 1)
        end if
  Setpixel ((x_c + x), (y_c + y), 1)          : plot the 4 symmetric pixels corresponding to
  Setpixel ((x_c – x), (y_c + y), 1)          : each sampled pixel in Region 2
  Setpixel ((x_c + x), (y_c – y), 1)
  Setpixel ((x_c – x), (y_c – y), 1)
  end while                                   : end of generating Region 1 and three symmetric
regions
```

We have discussed one of the mostly used algorithm for generating ellipse. Although there are other two basic methods for generating ellipse.

1. Algebraic method
2. Trigonometric method.

These are simple and briefly discussed below.

4.5.3 Algebraic Method

The equation of an ellipse with center at (x_c, y_c) and major axis and minor axis lengths $2a$ and $2b$ respectively is,

$$\frac{(x-x_c)^2}{a^2}+\frac{(y-y_c)}{b^2}=1$$

From the above equation the solution for y for corresponding value of x can be obtained as,

$$y = y_c \pm b\sqrt{1-\frac{(x-x_c)^2}{a^2}}$$

For generating an ellipse algebraically, we initialize the starting pixel at $x = x_c$ – a and $y = y_c$ (leftmost pixel). Then we move along the major axis in (+)ve X direction with unit steps in x ($x = x + 1$). At each sampling position (x) we find the two y values (say y_1 and y_2) applying the above expression and after rounding off to integer values we plot two pixels (x, Round (y_1)) and (x, Round (y_2)). Thus two symmetric ellipse-halves about the major axis are generated and we get the full ellipse when we reach $x = x_c + a$.

4.5.4 Trigonometric Method

We know that the parametric expression of a (0, 0) centered ellipse is

$x = a \cos \theta, y = b \sin \theta$, where θ is the variable parameter of a point (x, y) on the ellipse

When the ellipse center is (x_c, y_c) instead of (0,0), the expression becomes

$x = x_c + a \cos \theta, y = y_c + b \sin \theta$

NOTES

Algorithm

1. Input x_c, y_c, a, b
2. Plot the first pixel at the rightmost coordinate where $x = x_c + a, y = y_c$; $(\theta = 0)$
2. Initialize step size $\Delta\theta = .004$ to create dense pixels
4. Calculate $\theta = \theta + \Delta\theta$
5. Calculate $x = x_c + a \cos\theta, y = y_c + b \sin\theta$
6. Plot four symmetric pixels
   Setpixel (Round $(x_c + x)$, Round $(y_c + y)$, 1)
   Setpixel (Round $(x_c - x)$, Round $(y_c + y)$, 1)
   Setpixel (Round $(x_c + x)$, Round $(y_c - y)$, 1)
   Setpixel (Round $(x_c - x)$, Round $(y_c - y)$, 1)
7. Check whether θ exceeds 90°. If so then stop. Else repeat steps 4 –7.

Note: *The above two processes do not ensure uniform spacing between adjacent pixels.*

4.6 ANTIALIASING

If you try to implement the various scan conversion algorithms discussed so far in a computer you will soon discover that sometimes same pixel is unnecessarily set to same intensity multiple times; sometimes some objects appear dimmer or brighter though all are supposed to have same intensity; and most of the objects generated are not smooth and appear to have rough edges. All these are standard side effects of scan conversion: the first type is called *over striking effect*, the second is *unequal intensity effect* while the third type which is the most common and most pronounced, known as *aliasing effect*.

In fact aliasing is a typical image quality problem on all pixel devices, including a computer screen. Aliasing is the stair-step effect or jaggies on the edges of objects displayed on computer screens – all diagonal and curved lines are displayed as a series of little zigzag horizontal and vertical lines and can be extremely distracting for PC users.

Notice how the letters have jagged edges (except i, 1 and the horizontal portion of letter A which are either perfectly horizontal or vertical).

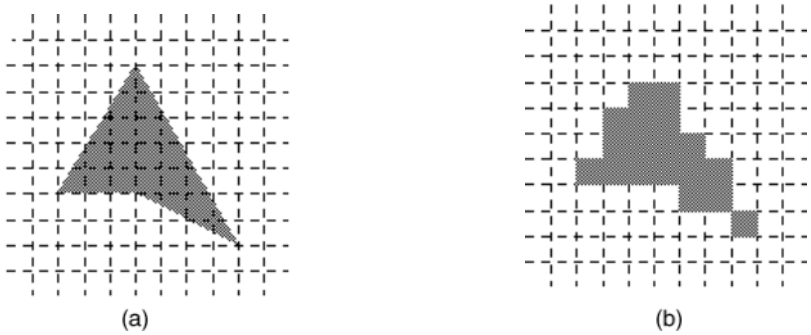
If we zoom in on the letter A, it is easy to see what is happening.

Figure 4.18: *The Aliasing Effect*

NOTES

Figure 4.19: *Aliasing is prominent in the entity primitives drawn –inclined line, circle & ellipse.*

Among the other effects of aliasing, jagged profiles and disintegrating textures are very common.



Figrue 4.20: *The picture on the left shows the sampling grid superimposed on the original scene. The picture on the right is the rendered image. A jagged profile is quite evident in the rendered image. As most of the pixels spanning the object's boundaries end up being shaded incorrectly as either wholly within the object or not at all within the object.*

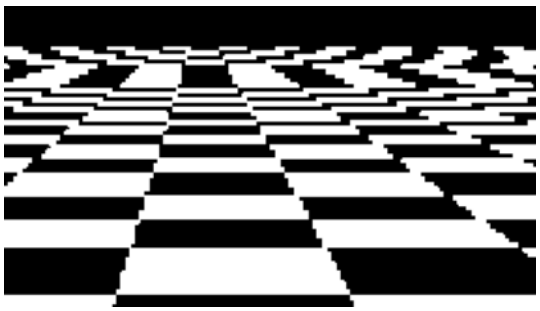
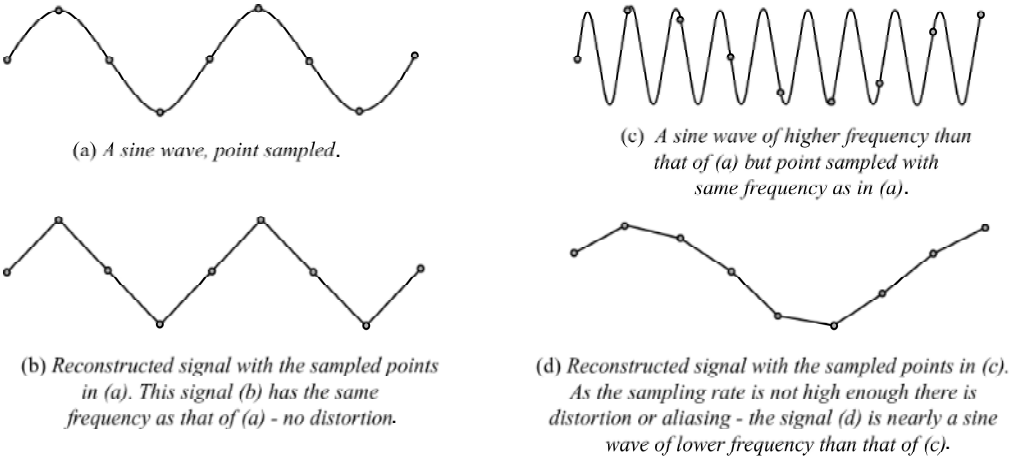


Figure 4.21: *This is a checkered texture on a plane. You can see how the checkers become disintegrated or irregularly shaped due to aliasing when their distance from the viewer increases.*

These jaggies are essentially caused by the problem of trying to map a continuous image onto a discrete grid of pixels. This continuous-to-discrete transformation (known as scan conversion) is performed by sampling the continuous line, curve etc. at discrete points (integer pixel positions) only followed by generating image pixels at integer locations that only approximate the true location of the sampled points. Pixels so generated at alias locations constitute aliases of the true objects or object edges. Therefore we can say aliasing occurs as a result of an insufficient sampling rate and approximation error (or more specifically quantization error).

In fact aliasing is a potential problem whenever an analog signal is point sampled to convert it into a digital signal. It can occur in audio sampling, for example, in converting music to digital forms to be stored on a CD-ROM or other digital devices. Whenever an analog audio signal is not sampled at a high enough frequency*, aliasing manifests itself in the form of spurious low frequencies.

NOTES



(a) *A sine wave, point sampled.*

(c) *A sine wave of higher frequency than that of (a) but point sampled with same frequency as in (a).*

(b) *Reconstructed signal with the sampled points in (a). This signal (b) has the same frequency as that of (a) - no distortion.*

(d) *Reconstructed signal with the sampled points in (c). As the sampling rate is not high enough there is distortion or aliasing - the signal (d) is nearly a sine wave of lower frequency than that of (c).*

Figure 4.22

Now coming to *Antialiasing,* quite obviously it implies the techniques used to diminish the jagged edges of an image so that the image appears to have smoother lines.

As aliasing problem is due to low sampling rate for low resolution, one easy solution is to increase the resolution, causing sample points to occur more frequently. This in turn will reduce the size of the pixels. The size of the 'jaggy' or stair-step error is never larger than the size of the actual pixel. Hence, reducing the size of the pixel reduces the size of these steps. But the cost of image production becomes higher with increased resolution as it calls for more graphics memory. The computational overhead increases for maintaining a high frame rate (60 fps) for bigger frame buffer. Thus, within the present limitations of hardware technology increased screen resolution is not a feasible solution.

There are quite a few standard methods for antialiasing. What antialiasing basically does is change the pixels around the edges to intermediate colours or grayscales. This has the effect of making the edges look smoother although it also makes them fuzzier.

Antialiasing

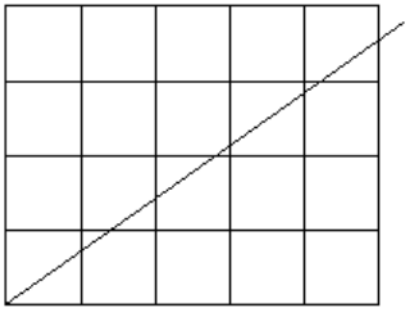
Figure 4.23(a): *Compare this text with the aliased text shown earlier. Can you see the jaggies any more? No, because this is an antialiased text.*

Figure 4.23(b): *If we zoom on the letters we see that grey shades are used to reduce the contrast of the black pixels at the edges with the white background. This is the reason why the text appears smooth though the jaggies are still there.*

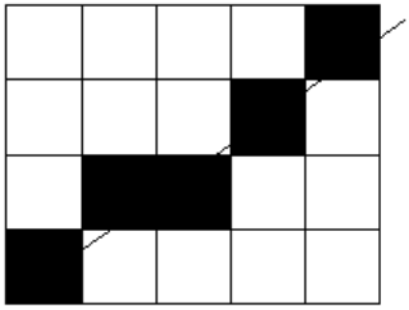
One of the methods of antialiasing is *Supersampling* or *Postfiltering*. In this method more than one sample is sampled per pixel. How? Every pixel area on the display surface is assumed to be subdivided into a grid of smaller subpixels (supersamples). Thus the screen is treated as having higher resolution than what actually is. A virtual image is calculated at this higher spatial resolution and then mapped (displayed) to the actual frame resolution

NOTES

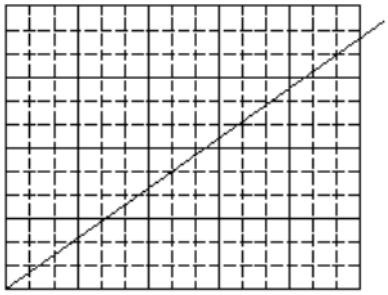
after combining (averaging) the results of the subpixels. The intensity value of a pixel is the average of the intensity values of all the sampled subpixels within that pixel. The following illustration describes the concept of supersampling.



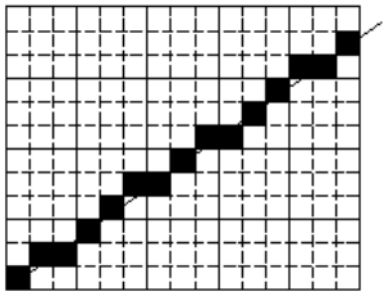
(a) *The theoretical line to be plotted on the actual pixel grid*



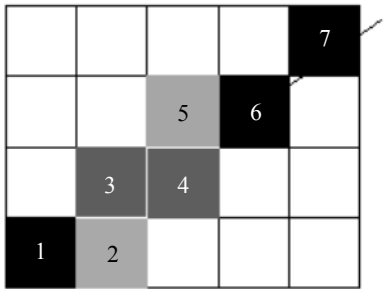
(b) *Using Bresenham's algo pixels are mapped – the line is displayed with aliases.*



(c) *For supersampling each square pixel is logically divided into 9 equal sized square subpixels.*



(d) *The virtual image of the line is calculated using same Bresenham's algo but with a smaller sampling interval (higher sampling frequency).*



(e) *The final antialiased image displayed on the actual pixel-grid after averaging the subpixel intensity values of the virtual image. Look three shades have been used to fill the pixels. As from Fig. 3.24(d) there are 3 sampling positions within a pixel box. In each of the boxes 1, 6 & 7 the three sampled subpixel colour value is black, hence the average colour of these three pixels is, (black +black +black)/3 = black. There are only two sampled subpixel in pixel 3 & 4 that falls on the line path; so the average colour of 3 & 4 is, (black + black + white)/3 = blackish grey these three pixels is, Similarly the colour of pixel 2 & 5 is (1*black + 2*white)/3 = whitish grey.*

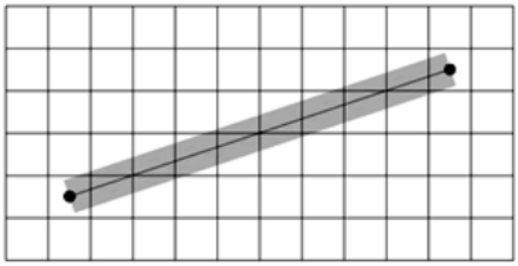
Figure 4.24

This method is also known as postfiltering because filtering is carried out after sampling. Filtering means eliminating the high frequencies, i.e., combining the supersamples to compute a pixel colour. This type of filtering is known as *unweighted filtering* because each supersample in a pixel, irrespective of its position, has equal influence in determining the pixel's colour. In other words, an unweighted filter computes an unweighted average. The other type of filter is a *weighted filter*. Through this filter each supersample is multiplied by its corresponding weight and the products are summed to produce a weighted average, which is used as the pixel colour. The weighting given to each sample should depend in some way on its distance from the centre of the pixel. The centre sample within a pixel has maximum weight. An array of values specifying the relative importance (weights) of subpixels can be set for different-sized grids and is often referred to as *Pixel-Weighting Masks*.

The other standard method of antialiasing is *Area Sampling or Prefiltering*. Prefiltering method treats a pixel as an area, and computes pixel colour based on the overlap of the

scene's objects with a pixel's area. These techniques compute the shades of grey based on how much of a pixel's area is covered by an object.

For example, a modification to Bresenham's algorithm was developed by *Pitteway* and *Watkinson*. In this algorithm, each pixel is given an intensity depending on the area of overlap of the pixel and the line.



In display, lines are of finite width (approximately equal to that of a pixel), so we can treat line as a rectangle. The intensity of each pixel is proportional to the area of intersection between the pixel and the rectangle. Pixel overlap areas are obtained by determining where rectangle boundaries intersect individual pixel boundaries.

Figure 4.25

Compared to postfiltering, prefiltering technique doesn't take into account calculating subpixel intensities (thus eliminates higher frequencies) before determining pixel intensities. A better technique, *weighted area sampling* uses the same basic approach, but gives greater weight to area near the centre of the pixel. Prefiltering thus amounts to sampling the shape of the object very densely within a pixel region. For shapes other than polygons, this can be very computationally intensive. There are other methods as well besides these two like *Adaptive Sampling, Stochastic Sampling, Raytracing* etc. But as an introduction to this topic we may stop here now. However, it should be noted that aliasing is an inherent part of any discrete process, its effect can be minimised but not eliminated.

4.7 CHARACTER GENERATION

There are three basic methods to generate characters on a computer screen: (1) hardware-based (2) vector-based and (3) bit map-based methods. In the hardware-based method, the logic for generating character is built into the graphics terminal. Though the generation time is less the typefaces* are limited due to hardware restrictions.

In the vector-based method the characters are developed using a set of polylines and splines that approximates the character outline. This form of character representation is completely device-independent; memory requirement is less as boldface, italics or different size can be produced by manipulating the curves outlining the character shapes – it doesn't require separate memory blocks for each variation.

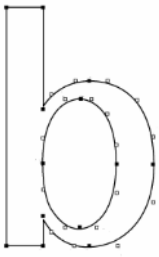


Figure 4.26: *Vector-Based Font Generated with Line and Spline Passing Through Control Points*

In the bitmap based method small rectangular bitmap called *character mask* (containing binary values 1 and 0) is used to store pixel representation of each character in a frame buffer area known as *font cache*. Relative pixel locations corresponding to a character bitmap are marked depending on the size, face and style (font) of character. Size of each character masks range from 5 × 7 to 10 × 12. A single font in 10 different font sizes and 4

NOTES

Check Your Progress

1. In general 'aliasing' is related to
   (a) scan conversion of line only
   (b) generation of characters only
   (c) conversion of any analog signal to digital signal
   (d) raster based graphics only
2. Which of the following is not true for Bresenham's approach?
   (a) It is a incremental method.
   (b) It deals with integers only.
   (c) It compares distances of candidate pixels from the entity path.
   (d) It can deal with parametric form of entities
3. The slope of a line is important for
   (a) DDA algorithm only
   (b) Bresenham's algorithm only
   (c) both DDA and Bresenham's algorithm
   (d) none of DDA and Bresenham's algorithm
4. State true (T) or false (F)
   (a) DDA algorithm works for lines in the 1st quadrant only
   (b) Midpoint method for scan converting ellipse uses 4-way symmetry
   (c) Sampling direction once set doesn't vary while scan converting a given entity primitive.
   (d) Bresenham's algorithm can be modified to produce a dashed line.
   (e) If the aspect ratio is not equal to 1 then a circle generated may look like an ellipse.

NOTES

faces (normal, **bold**, *italic*, ***bold italic***) would require 40 font caches. Characters are actually generated on display by copying the appropriate bitmaps from the frame buffer to the desired screen positions. A mask is referenced by the coordinate of the origin (lower left corner) of the mask w.r.t frame buffer addressing system.

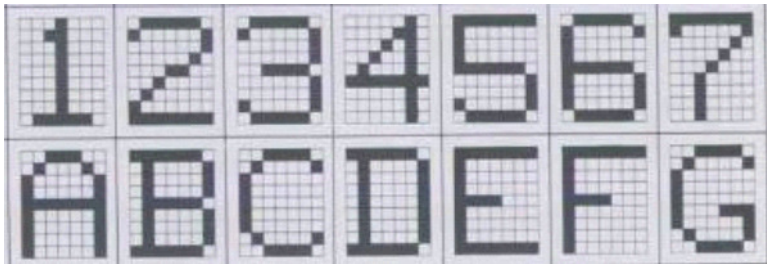


Figure 4.27: *Bitmapped Font*

In bit map based method a bold face character is obtained by writing the corresponding 'normal' character mask in consecutive frame buffer x-locations. Italics character is produced by necessary skewing of 'normal' character mask while being written in the frame buffer. In fact a typeface designer can create from scratch new fonts using a program like Windows Paint. The overall design style (font and face) for a set of characters is called a typeface.

4.8 SUMMARY

Scan converting entity primitives are the basic building blocks for developing a CAD package or for programmatically generating application graphics on a raster display. In this unit we have seen how stress is on minimizing floating point operations and why only incremental scan conversion algorithms are used. Moreover minimizing the error between the chosen pixels on a raster and the points on the ideal primitive defined on a Cartesian plane is a key factor in any graphics algorithm. Newer scan conversion algorithm evolves in pursuit of ideal optimization between 'correctness' and 'speed'. Only the basics have been covered here–many elaborations and special cases can be considered for robust implementation. The basic algorithms can be extended to handle thickness, as well as patterns for entity primitives and also real-time antialiasing.

4.9 ANSWERS TO 'CHECK YOUR PROGRESS'

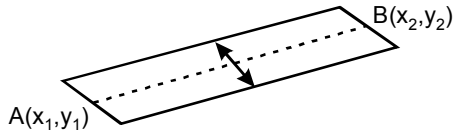
1. (c)
2. (d)
3. (c)
4. (a) F, (b)T, (c) F, (d) T, (e)T

4.10 EXERCISES AND QUESTIONS

1. What are graphics primitives? Mention some typical graphics primitives that a package may provide.
2. Bresenham's line drawing algorithm uses integer arithmetic. What is the justification for this approach?
3. Explain how rasterization accuracy is preserved despite use of integer arithmetic.
4. Develop the integer version of Bresenham's line drawing algorithm for lines in the third quadrant.
5. State the reason why we prefer unit x interval or unit y interval for corresponding slopes $m \leq 1$ and $m \geq 1$ in line drawing algorithms.

NOTES

6. It is desired to draw a line starting at A(3,6) and ending at B(6,2) on a graphics monitor. Use generalized Bresenham's algorithm to determine the pixels that would be put ON.
7. Develop an algorithm to draw a thick line from point A(x_1, y_1) to point $B(x_2, y_2)$ of thickness 'w' pixels.



8. Compare the advantages and disadvantages of the Bresenham's line drawing algorithm with those of the DDA algorithm.
9. Modify Bresenham's line generation algorithm so that it will produce a dashed line: the dash length should be independent of slope.
10. A line drawing algorithm draws a line by computing mid point pixel for a given pair of known end point pixels. Would you recommend it for raster graphics in preference to Bresenham's algorithm? You may record your observations based on hardware, software and visual considerations.
11. A line will be drawn from (x_1, y_1) to (x_2, y_2). Scan conversions are started from both (x_1, y_1) to (x_2, y_2) and also from (x_2, y_2) to (x_1, y_1) simultaneously following Bresenham's algorithm.
    (i) Write algorithm steps for such implementation.
    (ii) What is the advantage of this technique? Why?
12. Contrast the differences in both appearance and computational costs resulting from the use of unweighted and weighted antialiasing.
13. How can the Bresenham's line drawing algorithm be modified so that the antialiasing effects are produced during straight line generation.
14. It is desired that the circle with centre at the origin and radius 8 in the first quadrant is to be drawn. Using Bresenham circle generation algorithm determine the pixels which would approximate the desired portion of the circle.
15. When 8-way symmetry is used to obtain a full circle from pixel coordinates generated for the 0º to 45º octant some pixels are set or plotted twice. This phenomenon is sometimes referred to as overstrike. Identify where overstrike occurs.

4.11 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

NOTES

UNIT 5 2-DIMENSIONAL GRAPHICS

Structure

5.0 INTRODUCTION

The graphics packages range from the simple Windows Paint to the specialized AutoCAD and ANSYS. They all have the facility to change the shape or size or position or orientation of the displayed objects. Animations are produced by resizing an object or moving the object or camera along the animation path. All these concerns operate upon the object geometry by applying various geometric transformations like translation, rotation, and scaling. Each of these transformations is represented by specific transformation matrices, the nature and origin of which are demonstrated in this unit. The 2D point object is considered as representative of any graphic object.

5.1 UNIT OBJECTIVES

- Understanding the matrix representation of graphic object
- Analyzing the case study of different 2D transformations and derivation of corresponding transformation matrix
- Understanding the techniques for combination of individual transformation and case study of complex transformation with reference to concept of homogeneous coordinate
- Briefly addressing the use of transformation of coordinate systems

5.2 REPRESENTATION OF POINT AND OBJECT

A points coordinates can be expressed as elements of a matrix. Two matrix formats are used: one is row matrix and other is column matrix format.

For a 2D point (x, y) the row matrix format is a 1-row, 2-column matrix $[x\ y]$.

For a 3D point (x, y, z) it is a 1-row, 3-column matrix [x y z]. The same points in the column matrix format can be expressed as 1-column, 2-row matrix, $\begin{pmatrix} x \\ y \end{pmatrix}$ and 1-column 3-row matrix $\begin{pmatrix} x \\ y \\ z \end{pmatrix}$ respectively.

NOTES

In this book we will follow column matrix convention. As all graphic objects are built with finite no. of points each such object can be uniquely represented by a optimum no. of definition points lying on the object. Though there may be thousands of different adjacent points on a line between its two end points, this particular line segment can be uniquely expressed by the end points only. For example in column matrix format, $\begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix}$ may represent one and only one line between points (1, 2) and (3, 4).

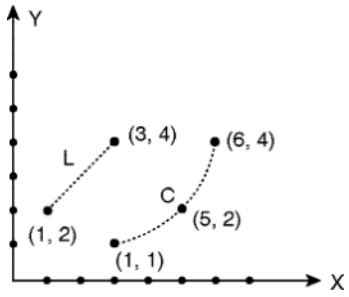


Figure 5.1: *A closer look at the line will reveal that it is basically a series of adjacent points, maintaining the straight line connectivity between the definition points (1, 2) & (3, 4). Similarly the arc segment 'C' is uniquely represented by 3 points (1, 1) – (5, 2) – (6, 4) in column-matrix format* $C = \begin{pmatrix} 1 & 5 & 6 \\ 1 & 2 & 4 \end{pmatrix}$

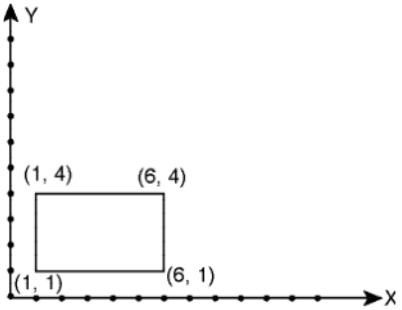
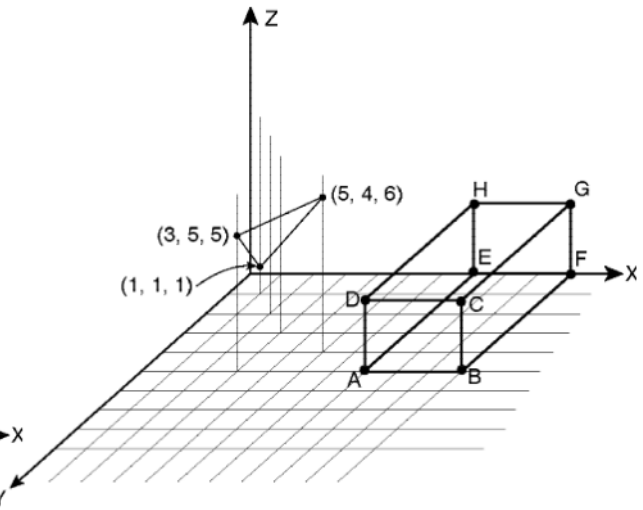


Figure 5.2



Figrue 5.3

A triangle having vertices at (1, 1, 1), (3, 5, 5) and (5, 4, 6) in 3D space (Figure 5.3) can be represented uniquely by

NOTES

$$\begin{pmatrix} 1 & 3 & 5 \\ 1 & 5 & 4 \\ 1 & 5 & 6 \end{pmatrix}$$

A 5 × 3 2D rectangle having its lower left vertex at (1, 1) in Figure 5.2 can be represented by

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

The rectangular parallelopiped *ABCDEFGH* in 3D space (Figure 5.3) may be defined by the matrix

$$\begin{pmatrix} 7 & 10 & 10 & 7 & 7 & 10 & 10 & 7 \\ 5 & 5 & 5 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 0 & 0 & 3 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ A & B & C & D & E & F & G & H \end{pmatrix}$$

In the following sections, we will see how the coordinates of these object definition points changes to represent the transformed object.

5.3 TRANSLATION

Let us think of a point *P* on a 2D plane. Assume that the current position or location of *P* is depicted by its coordinate (x, y) in a reference frame.

Now if we force *P* to move Δx distance horizontally and at the same time Δy distance vertically then the changed location of *P* becomes $(x + \Delta x, y + \Delta y)$.

In the terms of object transformation we can say that the original point object $P(x, y)$ has been translated to become P' $(x' y')$ and amount of translation applied is the vector $\overrightarrow{PP'}$, where $|\overrightarrow{PP'}| = \sqrt{(\Delta x)^2 + (\Delta y)^2}$

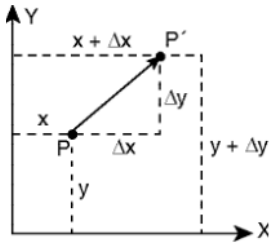


Figure 5.4: *Translation of a Point*

Vectorially we can express this transformation as, $P' = P + \overrightarrow{PP'}$

Algebraically, $x' = x + \Delta x$

$$y' = y + \Delta y$$

In matrix formulation the above relation can be more compactly expressed as,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \qquad (1)$$

$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$ is called *Translation vector* or *Shift vector*.

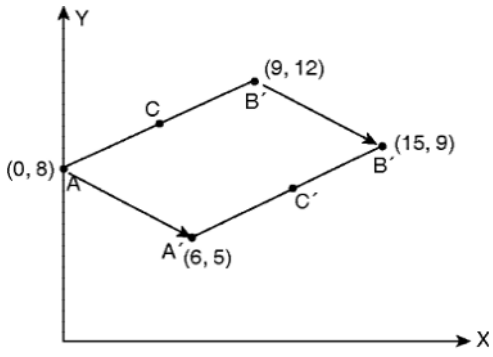


Figure 5.5: *Translation of a Line*

In general, the above equation (1) may be expressed as $[X'] = [X] + [T_T]$, where $[X']$ is the transformed object matrix, $[X]$ is the original object matrix and $[T_T]$ is the transformation (translation) matrix.

NOTES

Consider the line (shown in Figure 5.5) with end points A (0,8) and B (9,12). If we have to move this line AB to $A'B'$ we have to apply equal translation to each of the endpoints A and B and then redraw the line between the new end points deleting the old line AB. The actual operation of drawing a line between two endpoints depends on the display device used and the draw–algorithm followed. Here, we consider only the mathematical operations on the position vectors of the endpoints.

Here, A (0, 8) becomes A' (6, 5), implying $\Delta x = 6,\ \Delta y = -3$

B (9,12) becomes B' (15, 9), implying $\Delta x = 6,\ \Delta y = -3$

So we can say, $\begin{pmatrix}6\\5\end{pmatrix} = \begin{pmatrix}0\\8\end{pmatrix} + \begin{pmatrix}6\\-3\end{pmatrix}$ and $\begin{pmatrix}15\\9\end{pmatrix} = \begin{pmatrix}9\\12\end{pmatrix} + \begin{pmatrix}6\\-3\end{pmatrix}$

Combining these two $\Rightarrow \begin{pmatrix}6 & 15\\5 & 9\end{pmatrix} = \begin{pmatrix}0 & 9\\8 & 12\end{pmatrix} + \begin{pmatrix}6 & 6\\-3 & -3\end{pmatrix}$

It is to be noted that *whatever amount of translation we apply to a straight line, the length and orientation (slope) of the translated line remains same as that of the original line.*

It is implied from the Figrue 5.5 that this $\begin{pmatrix}6\\-3\end{pmatrix}$ translation matrix is theoretically applied to (i.e., added to) all the points forming the line AB. This can be tested with any intermediate point AB between A & B. Think of the midpoint; before transformation it is

$$C = \left(\frac{9+0}{2}, \frac{12+8}{2}\right) = (4.5,\ 10)$$

After transformation it is,

$$C' = \left(\frac{6+15}{2}, \frac{5+9}{2}\right) = (10.5,\ 7)$$

So, $\Delta x = 10.5 - 4.5 = 6$

$\Delta y = 7 - 10 = -3$

And this is the reason why we can express transformations in terms of any constituent point of the object concerned.

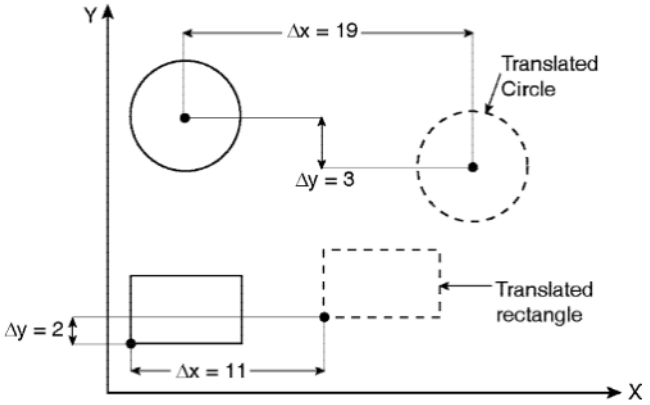


Figure 5.6: *The translated rectangle's point – coordinates are changed by* $\begin{pmatrix}11\\2\end{pmatrix}$ *w.r.t. the corresponding point – coordinates of the original rectangle.*

Similarly the circle is displaced by $\begin{pmatrix}19\\-3\end{pmatrix}$

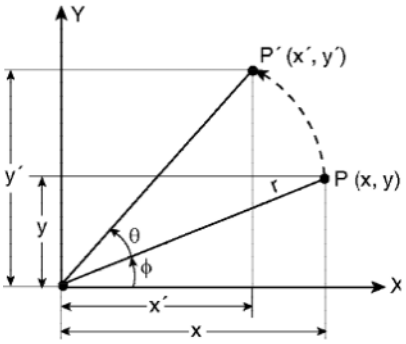
NOTES

For changing the position of a circle or ellipse, we translate the centre coordinates and redraw the figure in the new location.

Note from Figures 5.5 and 5.6 that we are transforming the objects without distorting the original shape size and orientation.

Thus we can define *Translation as a rigid body transformation that moves objects without deformation. Every point on the object is translated by the same amount and there exists a one to one correspondence between the transformed points and original points.*

5.4 ROTATION



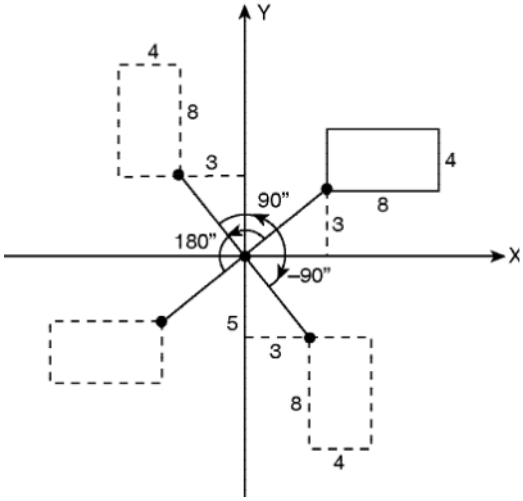


Figure 5.7: *Rotation of a Point about the Origin.* ***Figure 5.8:*** *Rotation of a Rectangle about the Origin.*

This transformation is used to rotate objects about any point in a reference frame. Unlike translation rotation brings about changes in position as well as orientation. The point about which the object is rotated is called the *pivot point or rotation point*. Conventionally anti-clockwise rotation about the pivot point is represented by positive angular value. Such transformation can also be visualized as rotation about an axis that is perpendicular to the reference plane and passes through the pivot point.

5.4.1 Rotation About Origin

Consider a trial case where the pivot point is the origin as shown in Figure 5.7. Then the point to be rotated $P(x, y)$ can be represented as

$$x = r\cos\phi \quad y = r\sin\phi$$

where (r, θ) is the polar coordinate of P. When this point P is rotated through an angle θ in anti- clockwise direction, the new point $P'(x', y')$ becomes,

$$x' = r\cos\ (\theta + \phi) \quad y' = r\sin\ (\theta + \phi)$$

Rewriting the above equations using laws of sines and cosines from trigonometry,

$$\left.\begin{aligned} x' &= r\ \cos\theta\ \cos\phi - r\sin\theta\sin\phi \\ y' &= r\ \sin\theta\ \cos\phi + r\cos\theta\sin\phi \end{aligned}\right\} \tag{2}$$

Replacing $r\cos\phi$ and $r\ \sin\phi$ with x and y respectively in (2) we get the simplified form,

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

In matrix rotation the above relation can be more compactly expressed as,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (3)$$

Symbolically $[X] = [T_R]\,[X]$ where $[T_R]$ is the transformation matrix for rotation.

NOTES

5.4.2 Rotation About an Arbitrary Pivot Point

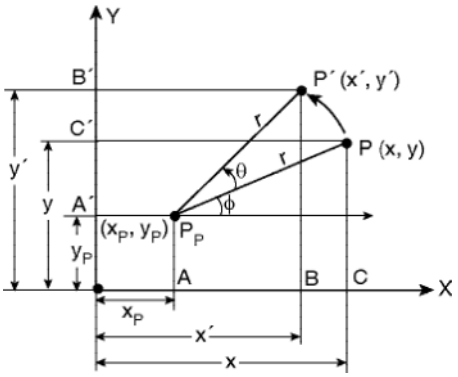
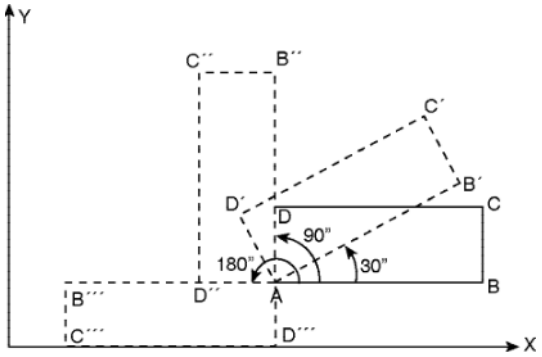


Figure 5.9: *CCW Rotation of Point P about P_P*



Figrue 5.10: *CCW Rotation of Rectangle ABCD About One of Its Corner Point A*

Eqn. (3) is applicable only for rotation about the origin. But in many applications the pivot point will not be the origin. It may be any point lying on the object(s) to be rotated or any point outside the object simply anywhere in the same 2D plane. For example consider the cases when we want to rotate any st.line about one of its end points or any rectangle about one of the corner points or any object lying on a circle about its centre.

Refer Figure 5.9. The Pivot Point is an arbitrary point P_P having coordinates (x_P, y_P). After rotating $P\,(x, y)$ through a positive θ angle its new location is $x'y'$ (P')

Here $x' = OB$

$= OA + AB$

$= x_P + r\cos(\theta + \phi)$

$= x_P + r\cos\phi\cos\theta - r\sin\phi\sin\theta$

$= x_P + (x - x_P)\cos\theta - (y - y_P)\sin\theta$

$\because \quad r\cos\phi = AC$

$= OC - OA$

$= x - x_P$

and $r\sin\phi = A'C'$

$= OC' - OA'$

$= y - y_P$

Also, $y' = OB'$

$= OA' + A'B'$

$= y_P + r\sin(\theta + \phi)$

$= y_P + r\cos\phi\sin\theta + r\sin\phi\cos\theta$

$= y_P + (x - x_P)\sin\theta + (y - y_P)\cos\theta$

Now,
$$\left.\begin{aligned} x' &= x_P + (x - x_P)\cos\theta - (y - y_P)\sin\theta \\ y' &= y_P + (x - x_P)\sin\theta - (y - y_P)\cos\theta \end{aligned}\right\} \quad (4)$$

NOTES

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta - \sin\theta \\ \sin\theta \quad \cos\theta \end{pmatrix} \begin{pmatrix} x - x_P \\ y - y_P \end{pmatrix} + \begin{pmatrix} x_P \\ y_P \end{pmatrix}$$

If we rearrange eqn. (4) by grouping the $(x_{P,} y_P)$ and (x, y) related terms we get

$$\begin{aligned} x' &= (x_P - x_P \cos\theta + y_P \sin\theta) + (x\cos\theta - y\sin\theta) \\ &= \{x_P(1 - \cos\theta) + y_P \sin\theta\} + (x\cos\theta - y\sin\theta) \end{aligned}$$

Similarly,

$$y' = \{(-x_P \sin\theta) + y_P(1 - \cos\theta)\} + (x\sin\theta + y\cos\theta)$$

This grouping allows us to express x' y' matrix, in terms of x_P, y_P matrix and x, y matrix as,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 - \cos\theta & \sin\theta \\ -\sin\theta & 1 - \cos\theta \end{pmatrix} \begin{pmatrix} x_P \\ y_P \end{pmatrix}$$

Symbolically $[X'] = [T_R]\ [X] + [T_P]$ (5)

Thus we see that the general equation for rotation, i.e., eqn. (5) differs from (3) by an additive term $[T_P]$* involving pivot point coordinates. In a later section we will convert such an expression into a more convenient format $[X'] = [T]\ [X]$ involving no additive terms.

Evaluation of the determinant of the general rotation matrix

$$[T_R] = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \text{ yields } \det [T_R] = \cos^2\theta + \sin^2\theta = 1$$

In general transformations with a determinant identically equal to + 1 yield pure rotation.

In Figure 5.7 P is transformed to P' through a (+) ve θ rotation. If we wish to bring back P' to P, we have to apply an inverse transformation, i.e., a (–) ve θ rotation. According to eqn. (3) the required transformation matrix to obtain P from P' is,

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

If we symbolize the above matrix as $[T_R]^{inv}$, then we find

$$\begin{aligned} [T_R][T_R]^{inv} &= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \\ &= \begin{pmatrix} \cos^2\theta + \sin^2\theta & \cos\theta\sin\theta - \sin\theta\cos\theta \\ \sin\theta\cos\theta - \cos\theta\sin\theta & \sin^2\theta + \cos^2\theta \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = [\mathrm{I}] \text{ where [I] is the identity matrix.} \end{aligned}$$

This implies that the inverse rotation matrix $[T_R]^{inv}$ for pure rotation is identical to the inverse of the rotation matrix, i.e., $[T_R]^{-1}$

The above is true because from matrix properties we know only $[T_R]\ [T_R]^{-1} = [I]$

Interestingly enough the transpose of the rotation matrix $[T_R]$

i.e., $[T_R]^T = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = [T_R]^{-1}$

Thus we can infer that *the inverse of any pure rotation matrix, i.e., one with a determinant equal to +1, is its transpose.*

NOTES

5.5 SCALING

5.5.1 Scaling with Respect to the Origin

Scaling is a transformation that changes the size or shape of an object. Scaling with reference to origin can be carried out by multiplying the coordinate values (x, y) of each vertex of a polygon, or each endpoint of a line or arc or the centre point and peripheral definition points of closed curves like a circle by scaling factors s_x and s_y respectively to produce the coordinates $(x' y')$.

The mathematical expression for pure scaling is,

$$\left.\begin{aligned} x' &= s_x \cdot x \\ y' &= s_y \cdot y \end{aligned}\right\} \Rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (6)$$

or symbolically, $[X'] = [T_S]\,[X]$

s_x expands or shrinks object dimensions along X direction whereas s_y affects dimensions along Y direction.

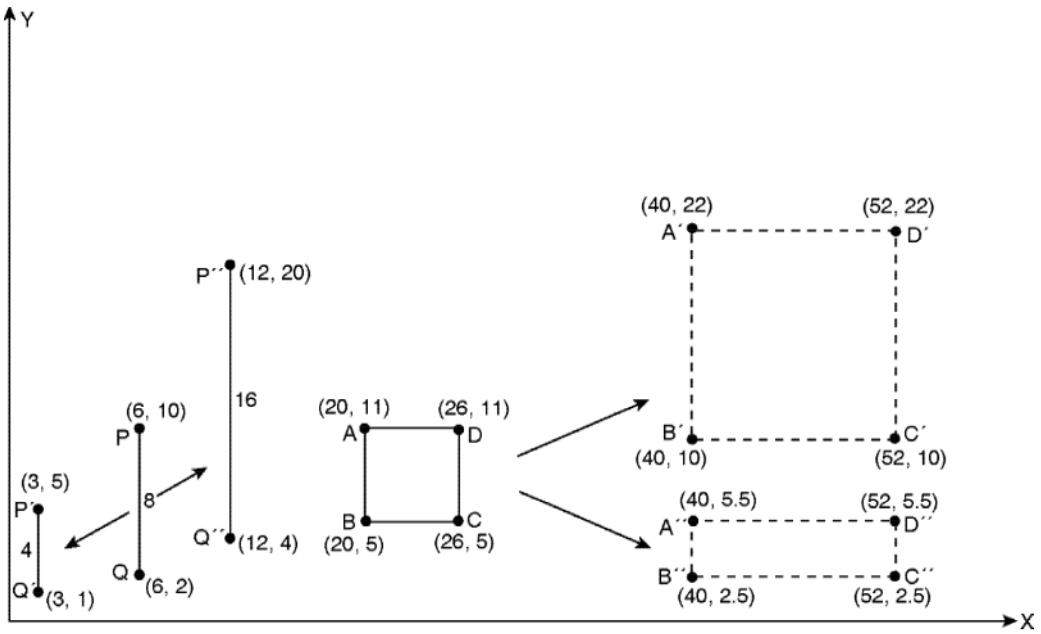


Figure 5.11: *Line PQ scaled to P′Q′ ($s_x = s_y = 0.5$) and to P′′Q′′ ($s_x = s_y = 2$). Square ABCD (each side 6 units) uniformly scaled ($s_x = s_y = 2$) producing a bigger square A′B′C′D′ with each side 12 units. But with non-uniform scaling ($s_x = 2$, $s_y = 0.5$) the square changes to a 12 by 3 rectangle A′′B′′C′′D′′.*

We can represent the scaling transformation carried out on square *ABCD* in Figure 5.11 as,

NOTES

$$[A'\,B'\,C'\,D'] = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} [A\,B \quad CD]$$

and $$[A''\,B''\,C''\,D''] = \begin{pmatrix} 2 & 0 \\ 0 & 0.5 \end{pmatrix} [A\,B\,C\,D]$$

Notice that in the 2nd case where $s_x \neq s_y$, a distortion in shape has occurred - the square has transformed into a rectangle.

In general, for *uniform scaling*, if $s_x = s_y > 1$, a uniform expansion occurs; i.e. the object becomes larger. If $s_x = s_y < 1$, then a uniform compression occurs; i.e. the object gets smaller. Non-uniform expansions or compressions occur, depending on whether s_x and s_y are individually > 1 or < 1 but unequal, such scaling is also known as *differential scaling*. While the basic object shape remains unaltered in uniform scaling, the shape and size both changes in differential scaling. Another interesting point to be noted is that, there is always a translation associated with scaling. Look at the figures, irrespective of the nature of scaling and the scale factors the scaled objects have substantially moved from their respective original positions. This is easily understood if we recall that during scaling transformation, the position vectors of the definition points are actually scaled with respect to the origin. For example, consider the position vector of point Q (6, 2) of line PQ. Its magnitude with respect to origin is $\sqrt{6^2 + 2^2} = 2\sqrt{10}$. The magnitude of the position vectors of the scaled points Q' (3, 1), Q'' (12, 4) are respectively $\sqrt{10}$ and $4\sqrt{10}$ implying uniform scaling with scale factors 0.5 and 2 respectively. Also note that the direction of the position vectors of the original point $\left(\tan^{-1}\frac{2}{6}\right)$ and the scaled points $\left(\tan^{-1}\frac{1}{3} \text{ and } \tan^{-1}\frac{4}{12}\right)$ remains same. Thus, quite obviously, *pure uniform scaling with factors < 1 moves objects closer to the origin while factors > 1 moves objects farther from origin, at the same time decreasing or increasing the object size.*

5.5.2 Scaling with Respect to Any Arbitrary Point

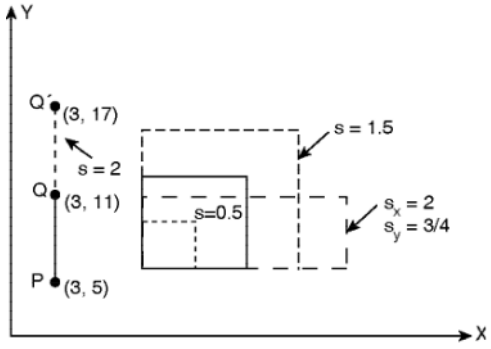


Figure 5.12

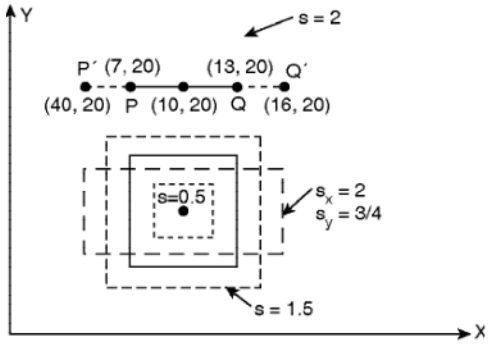


Figure 5.13

By scaling an object we generally expect it to grow or shrink in size or shape based on its original position (as shown in Figures 5.12 and 5.13). Apart from scaling, the movement of the object (which is intrinsically associated in scaling) with respect to origin, is mostly unwanted and can be eliminated by scaling the object with respect to a point conveniently chosen on the object itself. The point so chosen, called the *fixed point* remains unaltered in position after the scaling transformation when the scale factors are applied on the objects dimensions relative to that fixed point. As a result the object seems to expand or shrink in size or change in shape without any displacement of the object as a whole.

For example, consider the scaling transformation of PQ to PQ' (not $P'Q'$) in Figure 5.12. Here P is considered as the fixed point. So, instead of multiplying the scale factor ($s = 2$) to both P & Q position vectors it is multiplied with $Q - P$ and then added with P to obtain shifted Q' and then the line PQ' is reconstructed.

$$Q' = P + s(Q - P) = (3,5) + 2\{(3-3), (11-5)\}$$
$$= (3,5) + (0,12)$$
$$= (3, 17)$$

Note that $Q - P$ is the dimension of the line PQ relative to P. Now compare the result with the scaling of line PQ as shown in Figure 5.11 Instead of endpoint P if we consider the midpoint of PQ as the fixed point it will expand symmetrically in both direction forming $P'Q'$ (Figure 5.13), the midpoint remaining unchanged in position.

For scaling of a rectangle shown in Figure 5.13 the fixed point is the centroid of the rectangle. Thus the fixed point need not necessary lie on the object—it can be any point either on, within or outside the object. Here we derive the most generalized expression for scaling any object (P) coordinate say (x, y) with respect to any arbitrary point say, P_f (x_f, y_f).

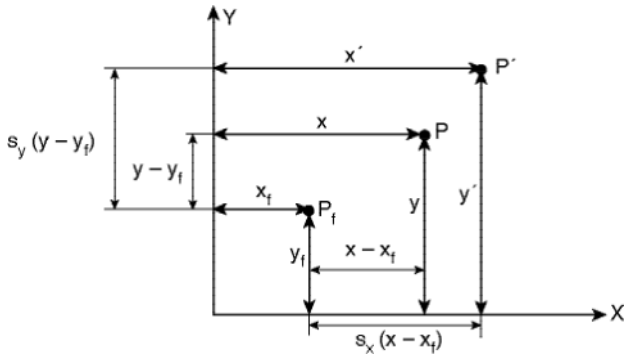


Figure 5.14: *Scaling of a Point with Respect to a Fixed Point s_x, $s_y > 1$*

As in this case the distance between the point in question $P(x, y)$ and the fixed point $P_f(x_f, y_f)$ is scaled we can write,

$$x' = x_f + (x - x_f)s_x$$
$$y' = y_f + (y - y_f)s_y$$

where (x' y') are the scaled coordinates and s_x, s_y are the scale factors.

We can rewrite these transformation equations to separate the multiplicative and additive terms:

$$x' = s_x\, x + (1 - s_x)\, x_f$$
$$y' = s_y\, y + (1 - s_y)\, y_f$$

In matrix notation,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 1 - s_x & 0 \\ 0 & 1 - s_y \end{pmatrix} \begin{pmatrix} x_f \\ y_f \end{pmatrix}$$

Symbolically,

$$[X'] = [T_s]\,[X] + [T_f]^* \qquad (7)$$

s_x and s_y can be equal or unequal and can be >1, <1, equal to 1 or even negative integer or fraction but never equal to zero.

Comparing eqn. (7) with (5) we find that coordinates for a *fixed point* feature in the scaling equations similar to the coordinates for a *pivot point* in the rotation equations.

NOTES

NOTES

5.6 REFLECTION

A reflection is a transformation that produces a mirror image of an object. In 2D reflection we consider any line in 2D plane as the mirror; the original object and the reflected object are both in the same plane of the mirror line. However we can visualize a 2D reflection as equivalent to a 3D rotation of 180º about the mirror line chosen. The rotation path being in the plane perpendicular to the plane of mirror line. Here we will study some standard 2D reflection cases characterized by the mirror line.

5.6.1 Reflection About X Axis

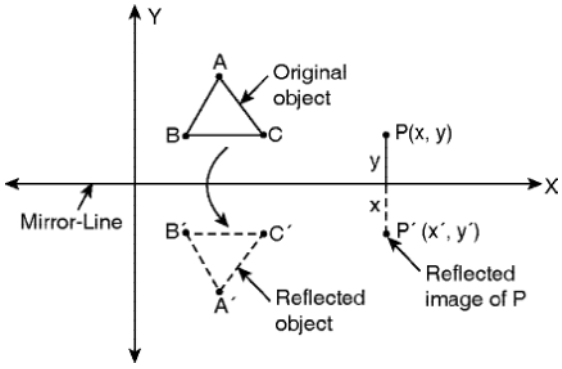


Figure 5.15: *Reflection About X Axis*

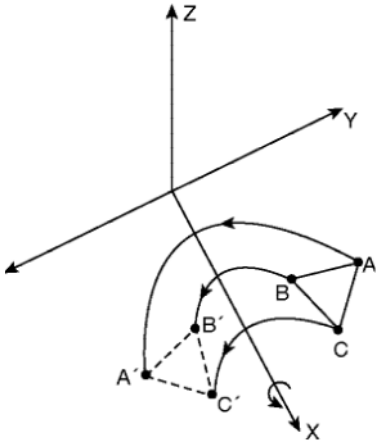


Figure 5.16: *2D Reflection About X Axis as 3D rotation(180^0) about X axis*

Basic Principles of Reflection Transformation

(1) The image of an object is formed on the side opposite to where the object is lying, with respect to the mirror line.

(2) The perpendicular distance of the object (i.e., the object points) from the mirror line is identical to the distance of the reflected image (i.e., the corresponding image points) from the same mirror line. Once again, the two perpendiculars must be along the same straight line.

Therefore the relation between the point $P(x, y)$ and its image $P'(x', y')$ about X axis is simply,

$$x' = x$$
$$y' = -y \quad \text{(Refer Figure 5.15)}$$

So the transformation matrix for reflection about X axis or $y = 0$ axis is,

$[T_M]_{y=0} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ and the transformation is represented as,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{8}$$

i.e. $[X'] = [T_M]_{y=0}[X]$

Consider the 2D reflection of the triangle ABC about X axis (in Figure 5.15) forming the image $\Delta\ A'B'C'$. In the other way we can interpret this event as a 3D rotation–we can think of the ABC triangle moving out of the xy plane and rotating 180º in 3D space about the X axis and back into the xy plane on the other side of the X axis.

NOTES

5.6.2 Reflection about Y axis

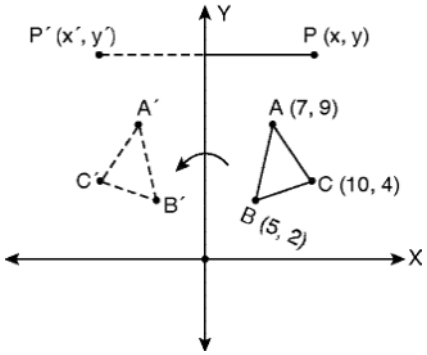


Figure 5.17: *Reflection About Y Axis*

A reflection about Y axis flips x coordinates while y coordinates remains the same. For reflection of $P(x, y)$ to $P'(x', y')$ in Figure 5.17,

$$x' = -x$$
$$y' = y$$

This transformation is identified by the reflection –transformation matrix

$$[T_M]_{x=0} = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad (9)$$

The transformed new vertices $A'B'C'$ of triangle ABC are given by

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 7 & 5 & 10 \\ 9 & 2 & 4 \end{pmatrix} = \begin{pmatrix} -7 & -5 & -10 \\ 9 & 2 & 4 \end{pmatrix}$$

5.6.3 Reflection About the Straight Line $y = x$

To find the relation between a point's coordinates (x, y) and those of its image (x', y'), reflected through $y = x$ straight line, look at Figure 5.18.

Here $PK = P'K$ and both PK and $P'K$ are perpendicular to $y = x$ or OK.

From simple geometry you will find $\Delta\ POK \cong \Delta\ P'OK$

$\therefore\ OP = OP'$ and $\angle\ POK = \angle\ P'OK$ implying also

$\angle\ POM = \angle\ P'ON$ because $y = x$ straight line makes 45º angle with both the axes.

That in turn implies $\Delta POM \cong \Delta P'ON$

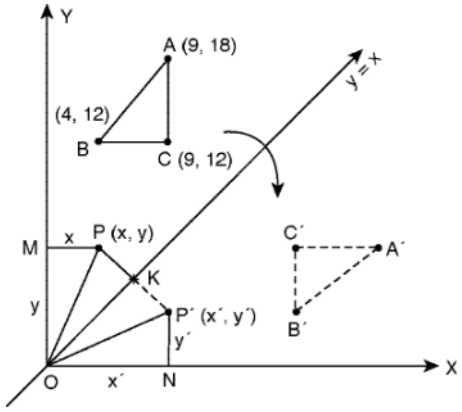


Figure 5.18

NOTES

So, $PM = P'N$ and $OM = ON$
but $PM = x$, $P'N = y'$, $OM = y$, $ON = x'$

Hence $x' = y$
$y' = x$

This can be represented by,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \quad (10)$$

where $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=x}$

Thus the transformed vertices $A'B'C'$ of triangle ABC in Figure 5.18 are given by

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 9 & 4 & 9 \\ 18 & 12 & 12 \end{pmatrix} = \begin{pmatrix} 18 & 12 & 12 \\ 9 & 4 & 9 \end{pmatrix}$$

5.6.4 Reflection About the Straight Line $y = -x$

Considering the experimental point $P(x, y)$, we draw a geometric (Figure 5.19) similar to that in Figure 5.18. Similarly we can prove

$$\left.\begin{aligned} PM &= P'N \\ OM &= ON \end{aligned}\right\}$$

But this time we can not simply state $x' = y$ and $y' = x$ from the above relation, because that is only the equations of magnitude. Looking at the figure we find that P and its image P' both being in the 4th quadrant, x and x' are (+)ve whereas y and y' are –ve. So considering this fact we should write,

$$x' = -y$$
$$y' = -x$$

This can be represented by,

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

where $\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=-x}$

Thus we can infer that unlike in the case of reflection about diagonal axis $y = x$, in reflections about the other diagonal $y = -x$, the coordinate values are interchanged with their signs reversed.

Notice the changes of the vertices of triangle ABC to $A'B'C'$ in obtained the Figure 5.19 given by,

$$\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} 4 & -2 & 4 \\ 6 & 4 & -2 \end{pmatrix} = \begin{pmatrix} -6 & -4 & 2 \\ -4 & 2 & -4 \end{pmatrix}$$

5.6.5 Reflection Relative to the Origin

NOTES

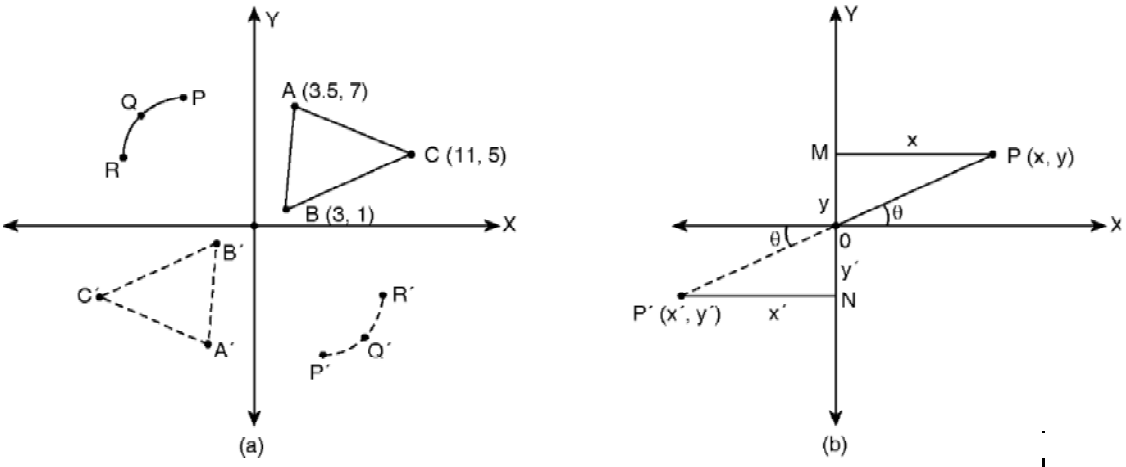


Figure 5.19

In this case we actually choose the mirror-line as the axis perpendicular to the *xy* plane and passing through the origin. After reflection both the *x* and *y* coordinate of the object point is flipped, i.e., *x′* becomes *–x* and *y′* becomes **–*y*.** This can be easily proved from geometry as shown in Figure 5.19(b). The coordinate signs are just opposite because the image is always formed in the quadrant opposite to that of the object.

Thus $x' = -x$

$y' = -y$, which can be represented in matrix form as

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

$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$ is the transformation matrix $[T_M]_{y=x=0}$

The change of the triangle *ABC* to *A′B′C′* in Figure 5.19(a) is given by,

$$\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} 3.5 & 3 & 11 \\ 7 & 1 & 5 \end{pmatrix} = \begin{pmatrix} -3.5 & -3 & -11 \\ -7 & -1 & -5 \end{pmatrix}$$

Similarly the curve *PQR* is changed to *P′Q′R′* represented by,

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

Notice that the radial distances of all the object points, before and after reflection, remains unaltered.

In the previous sections we have discussed reflection transformation about axes or lines passing through origin. The question, is how to find the transformation equations for reflections about any arbitrary line $y = mx + c$ in 2D *xy* plane. We cannot derive those equations as easily as we did for rotation about any arbitrary pivot point or scaling about any arbitrary fixed point. However, we can accomplish such reflections with a combination of basic translate-rotate–reflect transformations, which may be easier to understand after we establish a convenient method of combining consecutive basic transformations on an object in the next section.

Before that it is important to note that all the standard reflection transformations discussed so far could have been achieved, alternatively, by scaling with appropriate choice of scale factors (+ve, or –ve). In that sense reflection is not a basic transformation. But interestingly all these reflection matrices have a determinant equal to -1.

In general, if the determinant of a transformation matrix is identically –1, then the transformation produces a pure reflection.

NOTES

5.7 HOMOGENEOUS COORDINATES AND COMBINATION OF TRANSFORMATION

We have seen how the shape, size, position and orientation of 2D objects can be controlled by performing matrix operation on the position vectors of the object definition points. In some cases, however a desired orientation of an object may require more than one transformation to be applied successively. For illustration let us consider a case which requires 90° rotation of a point $\begin{pmatrix} x \\ y \end{pmatrix}$ about origin followed by reflection through the line $y = x$.

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

This $\begin{pmatrix} x' \\ y' \end{pmatrix}$ then undergoes reflection to produce $\begin{pmatrix} x'' \\ y'' \end{pmatrix}$

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

These successive matrix operations can be symbolically expressed as,

$$[X''] = [T_M]_{y=x} [X']$$

i.e., $[X''] = [T_M]_{y=x} \{[T_R]_{90^\circ} [X]\}$

As we know matrix multiplication is associative we can first perform $[T_M]_{y=x} [T_R]_{90^\circ}$ instead of performing $[T_R]_{90^\circ} [X]$ first (but not $[T_R]_{90^\circ} [T_M]_{y=x}$, thereby maintaining the right to left order of succession) to form a *resultant transformation matrix*. This matrix when multiplied with the original point coordinates will yield the ultimate transformed coordinates.

$$\text{Thus, } [T_M]_{y=x} [T_R]_{90^\circ} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$= [T_{COMB}] \ (say)$$

$$\text{Now, } [T_{COMB}][X] = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix}$$

Which shows the same result as before.

This process of calculating the product of matrices of a number of different transformations in a sequence is known as *concatenation* or, *combination* of transformations and the resultant product matrix is referred to as *composite or concatenated transformation matrix*. Application of concatenated transformation matrix on the object coordinates eliminates the calculation of intermediate coordinate values after each successive transformation.

But the real problem arises when there is a translation or rotation or scaling about an arbitrary point other than the origin involved among several successive transformations. The reason being the general form of expression of such transformations is not simply, $[X'] = [T]\ [X]$ involving the 2 by 2 array $[T]$ containing multiplicative factors, rather it is in the form, $[X'] = [T_1]\ [X] + [T_2]$, where $[T_2]$ is the additional two element column matrix containing the translational terms. Such transformations cannot be combined to form a single resultant representative matrix. This problem can be eliminated if we can combine

$[T_1]$ and $[T_2]$ into a single transformation matrix. This can be done by expanding the usual 2×2 transformation matrix format into 3×3 form. The general 3×3 form will be something like,

NOTES

$\begin{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} & \begin{matrix} m \\ n \end{matrix} \\ \begin{matrix} 0 & 0 \end{matrix} & 1 \end{pmatrix}$ where the elements a, b, c, d of the upper left 2×2 sub matrix are the multiplicative factors of $[T_1]$ and m, n are the respective x, y translational factors of $[T_2]$.

But such 3×3 matrices are not conformable for multiplication with 2×1 2D position vector matrices. Herein lies the need to include a *dummy coordinate* to make 2×1 position vector matrix $\begin{pmatrix} x \\ y \end{pmatrix}$ to a 3×1 matrix $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ where the third coordinate is dummy.

Now if we multiply, $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$ with a non-zero scalar 'h' then the matrix it forms is $\begin{pmatrix} x\,h \\ y\,h \\ h \end{pmatrix}$ or symbolically, say, $\begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix}$ which is known as the *homogeneous coordinates* or homogeneous position vector of the same point $\begin{pmatrix} x \\ y \end{pmatrix}$ in 2D plane.

$$\begin{pmatrix} x \\ y \end{pmatrix} \rightarrow \begin{pmatrix} x_h \\ y_h \\ h \end{pmatrix} = h \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

The extra coordinate h is known as a weight, which is homogeneously applied to the cartesian components.

Thus a general homogeneous coordinate representation of any point $P(x, y)$ is (x_h, y_h, h) or (xh, yh, h) that implies, $x = \frac{x_h}{h}$, $y = \frac{y_h}{h}$

As 'h' can have any non-zero value, there can be infinite number of equivalent homogeneous representation of any given point in space. For example, (6, 4, 2), (12, 8, 4), (3, 2, 1), (1/2, 1/3, 1/6), (–3, –2, –1) all represent the physical point (3, 2).

But so far as geometric transformation is concerned our choice is simply $h = 1$ and the corresponding homogeneous coordinate triple $(x, y, 1)$ for representation of point positions (x, y) in xy-plane. Other values of parameter 'h' are needed frequently in matrix formulation of three dimensional viewing transformation. Though we have introduced homogeneous coordinates just as a tool for making our transformation operations easier, they have their own significance as the coordinates of points in projective space (not a subject we will pursue here).

Expressing positions in homogeneous coordinates allows us to represent all geometric transformation equations uniformly as matrix multiplication. Coordinates are represented with three element column vectors and transformation operations are written in form of 3 by 3 matrices. Thus, for translation, we now have,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{11}$$

NOTES

or,

Symbolically, $[X'] = [T_T]\,[X]$

Compare eqn.(11) with eqn. (1) both yield the same result, $x' = x + \Delta x$ and $y' = y + \Delta y$

Equations of rotation and scaling with respect to coordinate origin (derived earlier as (3) and (6) respectively) may be modified as,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{12}$$

and

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{13}$$

respectively.

Similarly the modified general expression for reflection may be

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{14}$$

where values of a, b, c, d depend upon choice of coordinate axes or diagonal axes as mirror line.

The simplest $[X'] = [T]\,[X]$ form representing rotation about any arbitrary pivot point (x_P, y_P) as modified from eqn. (5) is,

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_P + \sin\theta \cdot y_P \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta \cdot x_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{15}$$

and for scaling with respect to any arbitrary fixed point (x_f, y_f) the modified form of eqn. (7) is

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} s_x & 0 & (1-s_x)\cdot x_f \\ 0 & s_y & (1-s_y)\cdot y_f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{16}$$

It is now for you to do some matrix multiplications to compare results of eqn. (15) and (16) with those of eqn. (5) and eqn. (7) respectively. However we will show how to derive (15) and (16) using the theory of composite transformation in the next section.

But before we move to the next section just pause few minutes. Have you noticed that in all the above 3 × 3 transformation matrices the bottom corner element on the diagonal line is always 1 as the bottom row is always [0 0 1]. Now what happens if we force this corner element to be anything other than 1, say '*s*'? The effect is interesting. If the diagonal becomes [1 1 *s*] and all the other elements are zero, i.e., if

NOTES

$$[T] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix}$$

$$\text{then } [T][X] = [T] = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ s \end{pmatrix} = \begin{pmatrix} x' \\ y' \\ h \end{pmatrix} = [X']$$

Here $x' = x, y' = y, h = s$. Normalizing this yields the transformed coordinates as x/s and y/s which implies that overall scaling has occurred, s being the scale factor.

If $s < 1$, then an uniform expansion occurs whereas

If $s > 1$ an uniform compression occurs.

5.8 COMPOSITE TRANSFORMATION

Since we are now aware of representing all kinds of transformation, in a homogeneous manner, we can reduce a long sequence of transformations to a series of matrix multiplications, with little difficulty. No matrix addition is required. If $[T_1]$, $[T_2]$, $[T_3]$ be any three transformation matrices then the matrix product, $[T_1]\ [T_2]\ [T_3] = ([T_1]\ [T_2])\ [T_3] = [T_1]\ ([T_2]\ [T_3])$ which implies we can evaluate matrix product using either a left to right or a right to left associative grouping. But we must be extremely careful about the order in which transformations actually takes place and the order we follow while multiplying the corresponding transformation matrices. *For column-matrix representation of coordinate positions, we form composite transformations by multiplying matrices in order from right to left***.** Different order of multiplication of matrices will give different results. Because we all know that matrix product may not always be commutative, i.e., $[T_1]\ [T_2] \neq [T_2]\ [T_1]$.

For example, if we first rotate an object (counter-clockwise 90° about origin) and then translate the rotated object (by 2,1) then the representative matrix expression is,

$$[X'] = [T_T]_{2,1}\ \ [T_R]_{90°}\ \ [X]$$

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

(Here $[X]$ represents the rectangle shown in Figure 5.20(a))

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

NOTES

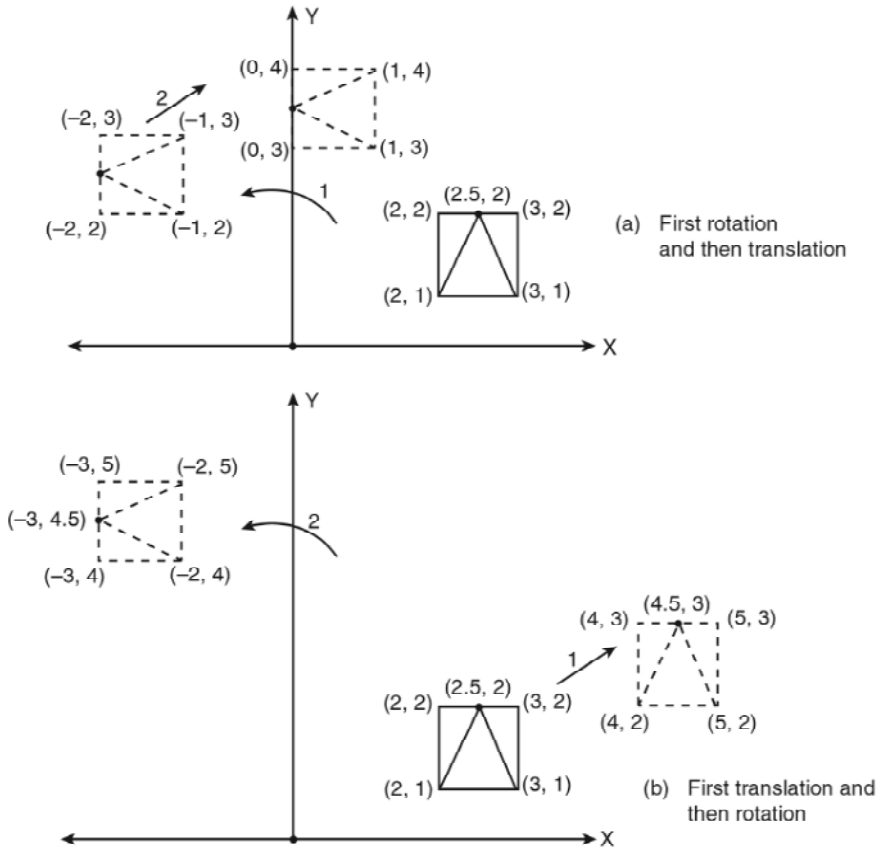


Figure 5.20

Now if we reverse the order of multiplication of $[T_T]_{2,1}$ and $[T_R]_{90^\circ}$ implying first translation and then rotation, then,

$$[X'] = [T_R]_{90^\circ} \quad [T_T]_{2,1} \quad [X]$$

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

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

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

The transformed object matrix $[X']$ is different in the above two cases.

However, *multiplication of transformation matrices is commutative for a sequence of transformations, which are of the same kind.*

As an example two successive rotations could be performed in either order and the final result would be the same. If the successive rotations are θ_1 and θ_2 about the origin to finally transform the point P to P' then,

NOTES

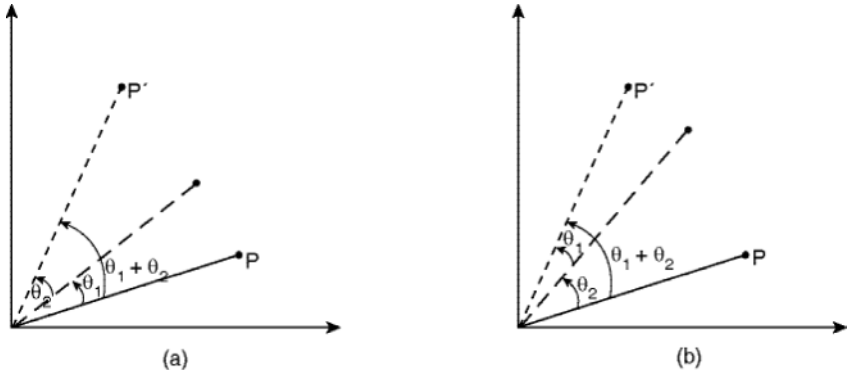
$$P' = [T_R]_{\theta 2}\,[T_R]_{\theta 1}\,P$$

$$= \begin{pmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= \begin{pmatrix} (\cos\theta_1\cos\theta_2 - \sin\theta_1\sin\theta_2) & -(\sin\theta_1\cos\theta_2 + \cos\theta_1\sin\theta_2) & 0 \\ (\cos\theta_1\sin\theta_2 + \sin\theta_1\cos\theta_2) & (-\sin\theta_1\sin\theta_2 + \cos\theta_1\cos\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= \begin{pmatrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix} P$$

$$= [T_R]_{\theta 1}\,[T_R]_{\theta 2}\,P = [T_R]_{\theta 1+\theta 2}\,P$$



Successive rotation: first by θ_1 and then by θ_2

First θ_2 rotation followed by θ_1 rotation; result same

Figure 5.21

The commutative property also holds true for successive translations or scaling or successive reflection about coordinate axes (excluding the diagonals).

For two successive translations of $(\Delta x_1, \Delta y_1)$ and $(\Delta x_2, \Delta y_2)$

$$[T]_{\Delta x2,\,\Delta y2}\,[T]_{\Delta x1,\,\Delta y1} = [T]_{\Delta x1,\,\Delta y1}\,[T]_{\Delta x2,\,\Delta y2}$$

$$= \begin{pmatrix} 1 & 0 & \Delta x_1 \\ 0 & 1 & \Delta y_1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x_2 \\ 0 & 1 & \Delta y_2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & \Delta x_1 + \Delta x_1 \\ 0 & 1 & \Delta y_1 + \Delta y_2 \\ 0 & 0 & 1 \end{pmatrix} = [T]_{\Delta x1+\Delta x2,\,\Delta y1+\Delta y2}$$

For two successive scaling by (s_{x1}, s_{y1}) and (s_{x2}, s_{y2}),

NOTES

$$[T_s]_{sx2,\ sy2} \quad [T_s]_{sx1,\ sy1} = \begin{pmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= [T_s]_{s_{x1},\ s_{y1}} \quad [T_s]_{sx2,\ sy2}$$

$$= [T_s]_{(sx1,\ sx2),\ (sy1,\ sy2)}$$

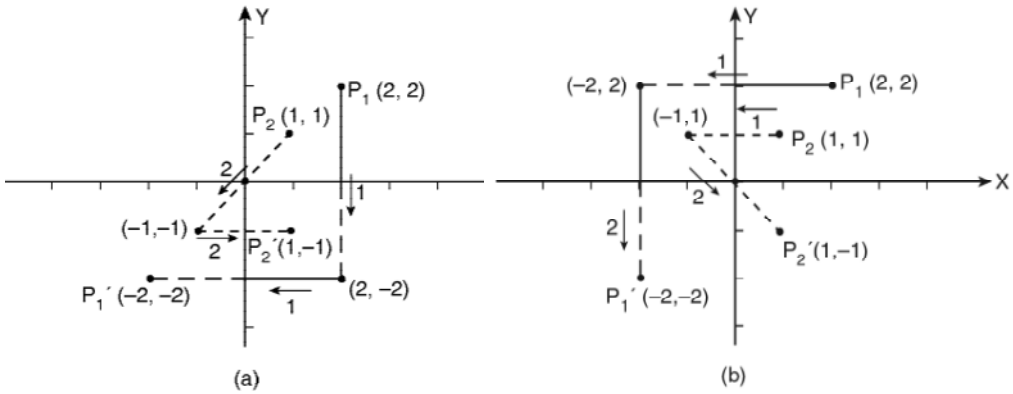


Figure 5.22: *Two commutative pairs of reflection transformation $P_1 \rightarrow P'_1$: about X axis and Y axis, $P_2 \rightarrow P'_2$: about origin and Y axis, operation sequences are shown by arrows and numbers.*

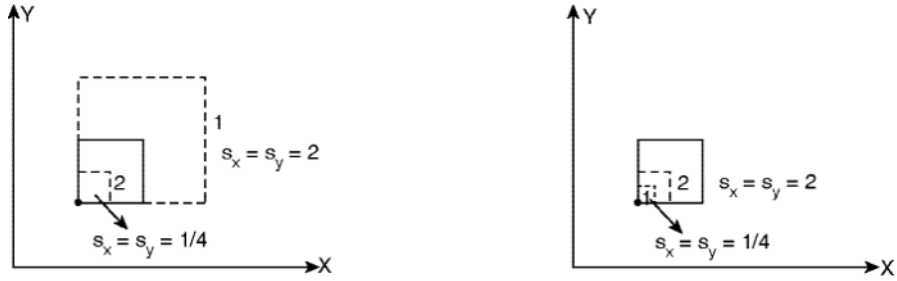
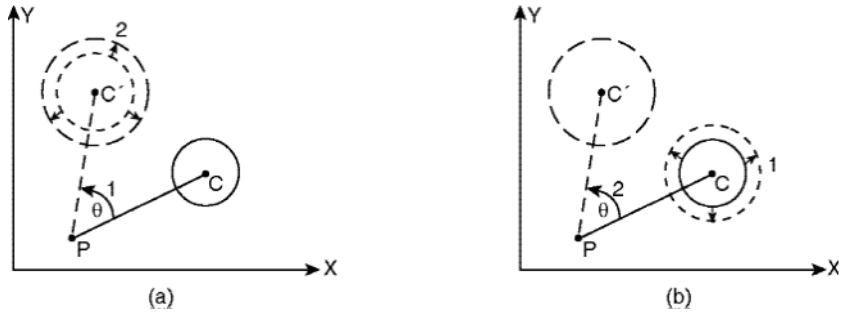


Figure 5.23: *Commutative pair of fixed point scaling; irrespective of the order of application of scale factors 2 and ¼ the final size of the square is 2 × ¼, i.e., half the size of the original square.*

One more example of commutative pair of operations is rotation and uniform scaling.



First rotation of circle about P, then scaling of circle w.r.t. its centre.

First scaling about centre, C followed by rotation about P; the result is same.

Figure 5.24

NOTES

5.8.1 Inverse Transformation

For each geometric transformation there exists an inverse transformation which describes just the opposite operation of that performed by the original transformation. Any transformation followed by its inverse transformation keeps an object unchanged in position, orientation, size and shape. For example, if an object is translated 3 units in the (+)ve X direction and then 3 units in the (–)ve X direction, the object comes back to its initial position implying no resultant transformation. We will use inverse transformation to nullify the effects of already applied transformation.

We will designate inverse of any transformation $[T]$ as $[T]^{-1}$. Listed below are the notations of inverse transformation matrices of some standard transformations.

Translation : $[T_T]^{-1}{}_{\Delta x, \Delta y} = [T_T]_{-\Delta x, -\Delta y}$

Rotation : $[T_R]^{-1}{}_{\theta} = [T_R]_{-\theta}$

Scaling : $[T_S]^{-1}{}_{Sx,\ Sy} = [T_S]_{1/Sx,\ 1/Sy}$

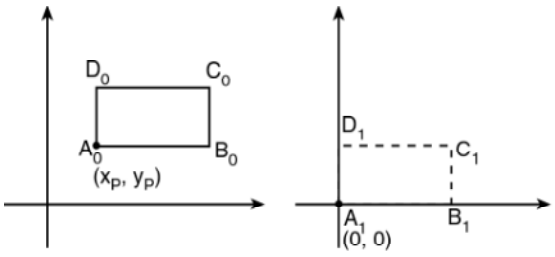
Reflection : $[T_M]^{-1}{}_{y=0} = [T_M]_{y=0}$ and $[T]^{-1}{}_{x=0} = [T_M]_{x=0}$

In the following sections we will derive the transformation matrices for general pivot point rotation, general fixed point scaling and general reflection (about any line), using the basic matrices for pure transformation, (such as rotation and scaling with respect to coordinate origin, reflection about coordinate axes) and translation. The technique here is based on the concept of composite transformation and inverse transformation.

5.8.2 General Pivot Point Rotation

Apart from solving a problem from the most basic theory another alternative approach is to derive the solution with reference to the familiar solution of a simpler problem of its kind. The standard solution (transformation matrix) of rotation about origin is much simpler and easy to remember compared to that of rotation about an arbitrary point other than origin. Hence in an alternative approach we will convert our present problem to a simple problem of rotation about the origin with the help of some additional translations. The sequential steps to be performed in this regard are as follows.

1. Translate the pivot point (x_p, y_p) and the object by the same amount such that the pivot point moves to the coordinate origin while the relative distances between the pivot point and the object points remains unchanged.
2. Rotate the object about the origin (by θ).
3. Translate the rotated object by an amount such that the pivot point comes back to its original position.



(a) Rectangle $A_0\ B_0\ C_0\ D_0$ to be rotated about pivot-pt. $A_0\ (x_p, y_p)$

(b) The rectangle & P.Pt. translated by $-x_p, -y_p$ so that A_0 becomes $A_1\ (0, 0)$ i.e. the origin

NOTES

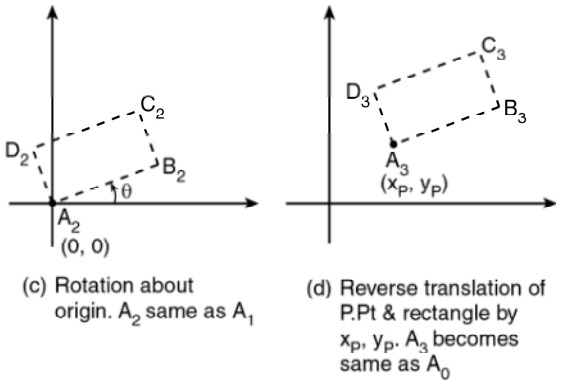


Figure 5.25

Now combining the matrices corresponding to above transformations sequence-wise the resultant transformation matrix is, $[T_{\text{COMB}}] = [T_T]^{-1}_{-x_P,\,-y_P}[T_R]_\theta\,[T_T]_{-x_P,\,-y_P}$

$$= \begin{pmatrix} 1 & 0 & x_P \\ 0 & 1 & y_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_P \\ 0 & 1 & -y_P \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_P + \sin\theta\ y_P \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta\ x_P \\ 0 & 0 & 1 \end{pmatrix}$$

This is the same transformation matrix we obtained for general pivot point rotation in eqn. (15).

The same principle,

Reducing a complex transformation problem to its simplest basic form at the cost of some additional to and fro translational or rotational movement is followed while deriving the transformation matrix for general fixed-point scaling or general reflection.

5.8.3 General Fixed Point Scaling

Here the problem is to scale an object with respect to a fixed point other than the origin. We will reduce this to the basic and simplest form of scaling with respect to origin through the following steps. (Figure 5.28)

1. Translate the object and the fixed point (x_f, y_f) equally so that the fixed point coincides with the coordinate origin.
2. Scale the translated object with respect to the coordinate origin (with scale factors s_x, s_y).
3. Use the inverse translation of step(1) to return the object and the fixed point to their original positions.

NOTES

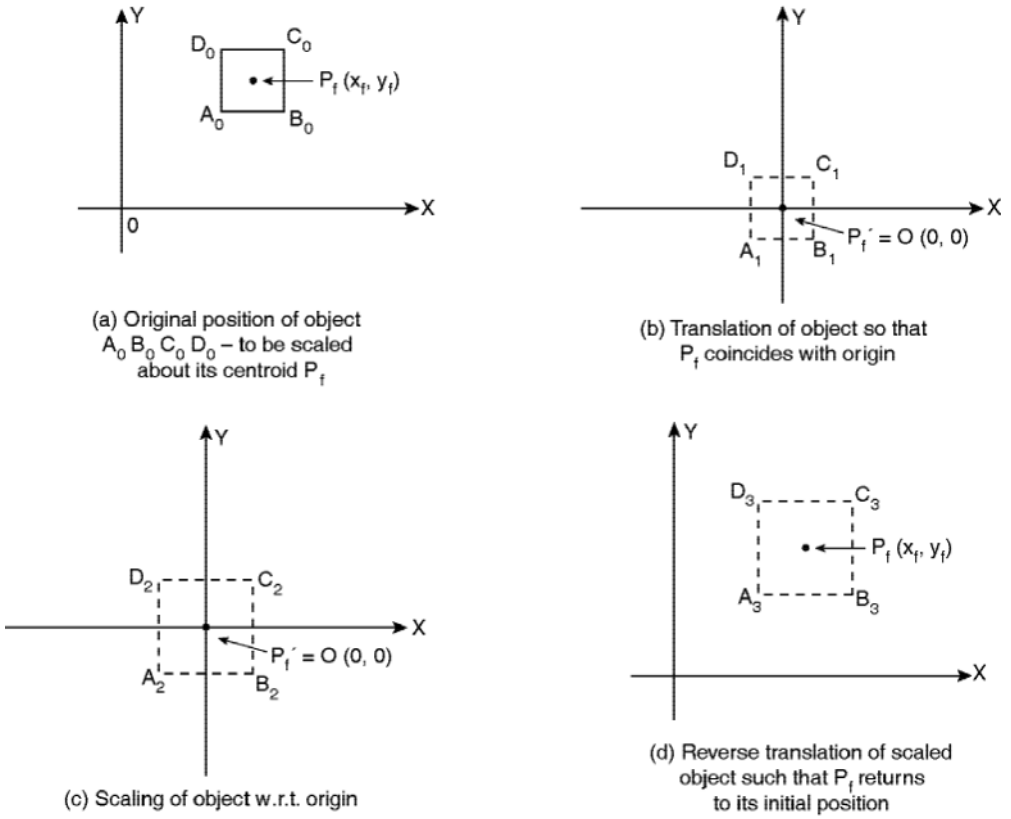


Figure 5.26

The composite transformation matrix resulting from combination of transformations as per above sequence is given by $[T_{\text{COMB}}] = [T_T]^{-1}_{-x_f, -y_f} [T_s]_{sx, sy} [T_T]_{-xf, -yf}$

$$= \begin{pmatrix} 1 & 0 & x_f \\ 0 & 1 & y_f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_f \\ 0 & 1 & -y_f \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x & 0 & (1-s_x)x_f \\ 0 & sy & (1-s_y)y_f \\ 0 & 0 & 1 \end{pmatrix}$$

Compare this with eqn. (16).

5.8.4 Reflection Through an Arbitrary Line

The problem of reflection of an object through a line that neither passes through the origin nor is parallel to the coordinate axes can be solved using the following steps.

1. Translate the line and the object so that the line passes through the origin.
2. Rotate the line and the object about the origin until the line is coincident with one of the coordinate axes about which we are familiar to perform reflection.
5. Reflect the object about that coordinate axis.
4. To the objects, apply the inverse rotation about the origin
5. Translate the object back to the original location.

In matrix notation $[T_{COMB}] = [T_T] [T_R] [T_M] [T_R]^{-1} [T_T]^{-1}$

If the arbitrary mirror line is given by $y = mx + c$ where m = slope of the line = tan θ, θ = zan$^{-1}(m)$ being the angle the line makes with the X axis.

NOTES

c = intercept on the Y axis made by the line (implying (0, c) is a point on the line).

Then the amount of translation in step (1) should be (0, –c) whereas the amount of rotation in step (2) should be $-\theta$, i.e., –tan^{-1}(m) about the origin if we prefer to merge the line with the X axis.

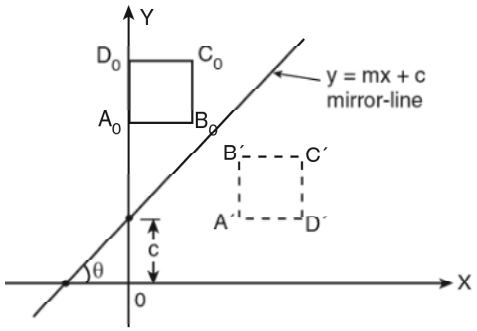
So we can say

$$[T_{COMB}] = [T]^{-1}_{0,-c}\ [T_R]^{-1}_{-\theta}\ [T_M]_{y=0}\ [T_R]_{\theta}\ [T]_{0,-c}$$

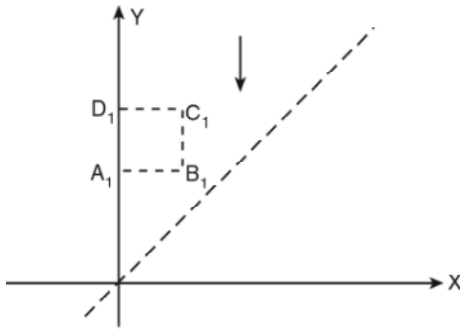
$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -c \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} (\cos^2\theta - \sin^2\theta) & 2\sin\theta\cos\theta & -2c\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & (\sin^2\theta - \cos^2\theta) & c(\cos^2\theta - \sin^2\theta) + c \\ 0 & 0 & 1 \end{pmatrix} \quad (17)$$

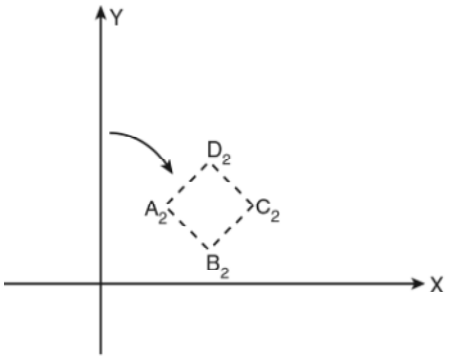
Put θ = tan^{-1}(m)



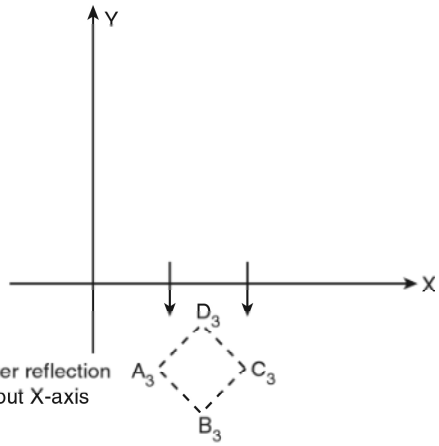
$A_0\ B_0\ C_0\ D_0$: original object
$A'\ B'\ C'\ D'$: object after reflection about y = mx + c



(a) After translation by $\Delta y = c$



(b) After rotation by – θ



(c) After reflection about X-axis

NOTES

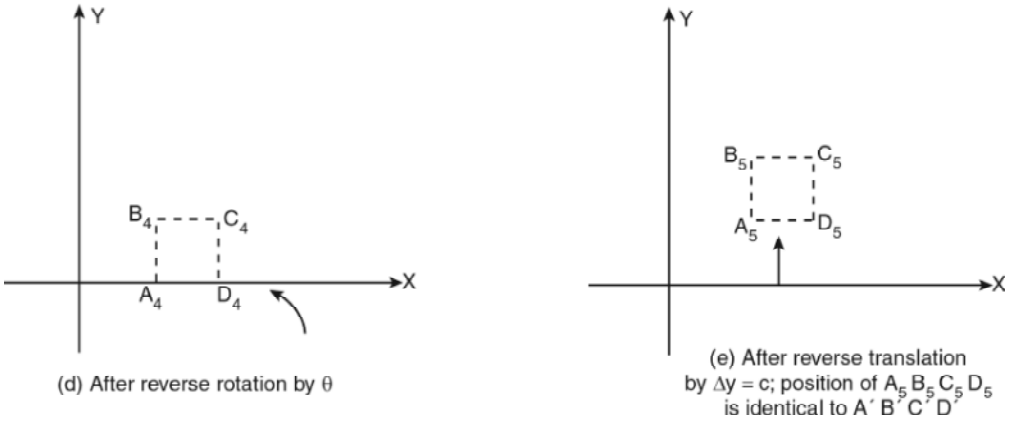


Figure 5.27

Note that eqn. (17) does not hold good for a mirror line parallel to the Y axis, say, $x = k$ straight line where $K \neq 0$. The reason is simple; we assumed that the mirror line $y = mx + c$ makes a finite intercept 'c' on the Y axis, but the line $x = k$ does not intersect the Y axis at all. For such cases we will first translate the object and mirror line $-k$ units in X direction to merge the mirror line with Y axis, then reflect the object about Y axis any finally translate the reflected object k units in X direction.

Accordingly, $[T_M]_{x=k} = [T_T]^{-1}_{-k,0}\,[T_M]_{x=0}\,[T_T]_{-k,0}$

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

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

5.9 TRANSFORMATION OF COORDINATE SYSTEM

So far we have discussed *geometric transformation* of *2d* objects which are well defined with respect to a *global coordinate system*, also called the *world coordinate system(WCS)*– the principal frame of reference. But it is often found convenient to define quantities (independent of the WCS) with respect to a *local coordinate system*, also called the *model coordinate system* or the *user coordinate system(UCS)*. While the UCS may vary from entity to entity and as per convenience, the WCS being the master reference system remains fixed for a given display system. Once you define an object 'locally' you can place it in the global system simply by specifying the location of the origin and orientation of the axes of the local system within the global system, then mathematically transforming the point coordinates defining the object from local to global system. Such transformations are known as the transformation between the coordinate systems. Here we will briefly discuss only the transformations between two cartesian frames of reference.

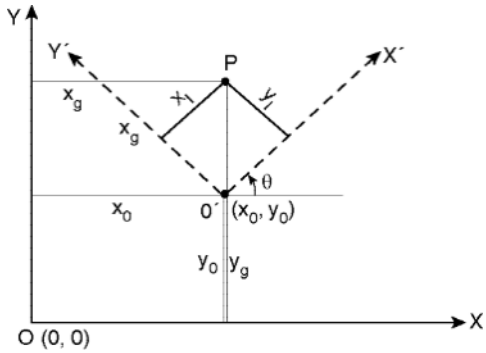


Figure 5.28: *The Coordinates of Point P w.r.t. Local cood.sys X′O′Y′ is, $(x_p y_p)$ While Its Coordinates w.r.t. WCS XOY is (x'_g, y'_g)*

Local to Global Figure 5.28 shows two cartesian systems global XOY and local X′OY′ with coordinate origins at (0, 0) and (x_0, y_0)

NOTES

respectively and with an of angle θ between the X and X′ axes. To transform object descriptions (x_l, y_l) w.r.t. local system to that x_g, y_g w.r.t. global system we need to follow the following two steps that superimpose the XOY frame to the X′O′Y′ frame.

Step 1

Translation: So that origin '*O*' of the XY system moves to origin '*O*' sof the X′Y′ system.

Transformation matrix:

$$[T_T]_{x_0,\, y_0} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 2

Rotation: So that X aligns with X′ and Y aligns with Y′

Transformation matrix:

$$[T_T]_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

So the global coordinates (x_g, y_g) of point *P* are expressed in terms of its known local coordinate (x_1, y_1) as,

$$\begin{pmatrix} x_g \\ y_g \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

Global to Local: Unlike the previous case, if the object is originally defined in the global XY system then to transform its description in local X′Y′ system (Figure 5.30). we have to superimpose local X′O′Y′ frame to the global XOY frame. So the transformations required this time are,

1. $[T_T]_{-x_0,\, -y_0} = \begin{pmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{pmatrix}$ and

2. $[T_R]_{-\theta} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

The description of point *P* in the local system is given by

$$\begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -x_0 \\ 0 & 1 & -y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_g \\ y_g \\ 1 \end{pmatrix}$$

Will it make any difference if we first rotate and then translate the objects? (Refer problem 15)

Note:

- **Shear:** A different category of transformation that distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other.

  When an X direction shear relative to the X axis is produced, the original coordinate position (x, y) is then shifted by an amount proportional to its distance 'y' from the X axis $(y = 0)$, i.e.,

$x' = x + sh_x \cdot y$ where sh_x = shear parameter in the positive X direction.

$y' = y$

the corresponding transformation matrix is,

$$\begin{pmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

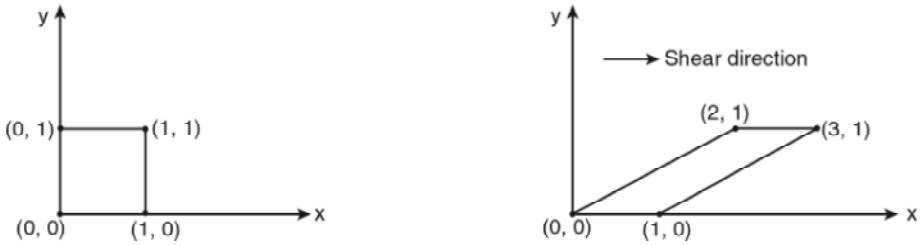


Figure 5.29: *A Unit Square is Transformed to a Parallelogram Following a X Direction Shear with $sh_x = 2$*

Based on the same principle, –shear in the X direction relative to a line parallel to the X axis, ($y = k$) –shear in the Y direction relative to Y axis or parallel to Y axis ($x = h$) can be formulated.

- **Affine Transformation:** All the two-dimensional transformations where each of the transformed coordinates x' and y' is a linear function of the original coordinates x and y as

$$\left.\begin{aligned} x' &= A_1 x + B_1 y + C_1 \\ y' &= A_2 x + B_2 y + C_2 \end{aligned}\right\}$$ where A_i, B_i, C_i are parameters fixed for a given transformation type.

2D transformation of coordinate systems, translation, rotation, scaling, reflection, shear- all are examples of 2D affine transformations and exhibit the general property, that parallel lines transform to parallel lines and finite points map to finite points. An affine transformation involving only translation, rotation and reflection preserves the length and angle between two lines.

5.10 SOLVED PROBLEMS

1. A unit square is transformed by a 2 × 2 transformation matrix. The resulting position vectors are

$$\begin{pmatrix} 0 & 2 & 8 & 6 \\ 0 & 3 & 4 & 1 \end{pmatrix}$$

What is the transformation matrix?

Solution

Let the unit square have coordinates (x, y), $(x + 1, y)$, $(x + 1, y + 1)$, $(x, y + 1)$ and let the transformation matrix be

$$\begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

so

$$\begin{pmatrix} 0 & 2 & 8 & 6 \\ 0 & 3 & 4 & 1 \end{pmatrix} = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} x & x+1 & x+1 & x \\ y & y & y+1 & y+1 \end{pmatrix}$$

$$= \begin{pmatrix} ax+cy & a(x+1)+cy & a(x+1)+c(y+1) & ax+c(y+1) \\ bx+dy & b(x+1)+dy & b(x+1)+d(y+1) & bx+d(y+1) \end{pmatrix}$$

Equating L.H.S. and R.H.S. element by element we get the following 4 sets of equations

NOTES

NOTES

$$\left.\begin{aligned} ax + cy &= 0 \\ bx + dy &= 0 \end{aligned}\right\} \quad \text{(i)}$$

$$\left.\begin{aligned} a(x+1) + cy &= 2 \\ b(x+1) + dy &= 3 \end{aligned}\right\} \quad \text{(ii)}$$

$$\left.\begin{aligned} a(x+1) + c(y+1) &= 8 \\ b(x+1) + d(y+1) &= 4 \end{aligned}\right\} \quad \text{(iii)}$$

$$\left.\begin{aligned} ax + c(y+1) &= 6 \\ bx + d(y+1) &= 1 \end{aligned}\right\} \quad \text{(iv)}$$

From (i) and (ii) we get, $a = 2$; $b = 3$
From (iii) and (iv) we get, $c = 6$; $d = 1$

Hence the transformation matrix is $\begin{pmatrix} 2 & 6 \\ 3 & 1 \end{pmatrix}$

2. In 2D graphics obtain the 3 × 3 transformation matrix for translating a point by –1,2. Calculate the inverse of this matrix and show that the result is a matrix that translates a point by 1, –2.

Solution

The 3 × 3 transformation matrix for translating a point by –1, 2 in the column matrix format is,

$$\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} = T(\text{say})$$

$$\text{Now adjoint } T = \begin{pmatrix} \begin{vmatrix} 1 & 2 \\ 0 & 1 \end{vmatrix} & -\begin{vmatrix} 0 & 2 \\ 0 & 1 \end{vmatrix} & \begin{vmatrix} 0 & 1 \\ 0 & 0 \end{vmatrix} \\ -\begin{vmatrix} 0 & -1 \\ 0 & 1 \end{vmatrix} & \begin{vmatrix} 1 & -1 \\ 0 & 1 \end{vmatrix} & -\begin{vmatrix} 1 & 0 \\ 0 & 0 \end{vmatrix} \\ \begin{vmatrix} 0 & -1 \\ 1 & 2 \end{vmatrix} & -\begin{vmatrix} 1 & -1 \\ 0 & 2 \end{vmatrix} & \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \end{pmatrix}^T$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -2 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$|T| = 1\begin{vmatrix} 1 & 2 \\ 0 & 1 \end{vmatrix} - 0\begin{vmatrix} 0 & 2 \\ 0 & 1 \end{vmatrix} + (-1)\begin{vmatrix} 0 & 1 \\ 0 & 0 \end{vmatrix}$$

$$= 1.1 - 0 + 0 = 1$$

$$T^{-1} = \frac{Adj.T}{|T|} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

This matrix is the transformation matrix for translation by 1, –2.

3. A triangle is defined by $\begin{pmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \end{pmatrix}$

Find the transformed coordinates after the following transformations.

(1) 90° rotation about origin.
(2) reflection about line $y = -x$.

Solution

After 90° rotation about origin, the transformed coordinate are,

$$= \begin{pmatrix} \cos 90 & -\sin 90 & 0 \\ \sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \\ 1 & 1 & 1 \end{pmatrix}$$

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

Finally after reflection about line $y = -x$, the transformed coordinates are,

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

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

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

4. Give the explicit form of the 3 × 3 matrix representing the transformation: Scaling by a factor of 2 in the X direction and then rotation about (2 , 1).

Solution

For scaling by a factor of 2 in the X direction the transformation matrix is,

$$[T_s]_{2,1} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For rotation about the point (2,1) we first translate the point so as to make it coincident with the origin, perform rotation about the origin and then give it reverse translation.

Here the matrix product that gives the transformation matrix for rotation about (2,1) is

$$[T_T]_{2,1}\,[T_R]_\theta\,[T_T]_{-2,-1}$$

$$= \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & 2 - 2\cos\theta + \sin\theta \\ \sin\theta & \cos\theta & 1 - 2\sin\theta - \cos\theta \\ 0 & 0 & 1 \end{pmatrix}$$

The final transformation matrix for the entire series of transformations is,

$$[T_T]_{2,1}\,[T_R]_\theta\,[T_T]_{-2,-1}\,[T_s]_{2,1}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & 2 - 2\cos\theta + \sin\theta \\ \sin\theta & \cos\theta & 1 - 2\sin\theta - \cos\theta \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2\cos\theta & -\sin\theta & 2 - 2\cos\theta + \sin\theta \\ 2\sin\theta & \cos\theta & 1 - 2\sin\theta - \cos\theta \\ 0 & 0 & 1 \end{pmatrix}$$

NOTES

NOTES

5. A polygon has 4 vertices located at $A(20, 10)$, $B(60, 10)$, $C(60, 30)$, $D(20, 30)$. Indicate a transformation matrix to double the size of the polygon with point A located at the same place.

Solution

This is a case of fixed point scaling where the fixed point is A. So for the desired transformation first translate the object such that point A coincides with origin. Then scale the object with respect to the origin with $s_x = s_y = 2$ and finally perform inverse translation such that the point A comes back to the original position (20, 10). So the concatenated transformation matrix is given by,

$$[T_{COMB}] = [T_T]_{20,10}[T_S]_{2,2}[T_T]_{-20,-10}$$

$$= \begin{pmatrix} 1 & 0 & 20 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -20 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 0 & -20 \\ 0 & 2 & -10 \\ 0 & 0 & 1 \end{pmatrix}$$

Tips: We can bypass the time consuming matrix multiplications to get the product matrix $[T_{COMB}]$ if we can afford to remember the fixed-point scaling matrix, i.e.,

$$\begin{pmatrix} s_x & 0 & (1-s_x)x_f \\ 0 & s_y & (1-s_y)y_f \\ 0 & 0 & 1 \end{pmatrix};$$

put the values of $s_x = s_y = 2$, $x_f = 20$, $y_f = 10$ and get the answer for this problem.

From the given coordinates it is clear that the polygon is a rectangle of size, $(60 - 20) \times (30 - 10)$, i.e., 40 units × 20 units implying area = 800 sq. units.

On application of the scaling transformation the rectangle will become,

$$= \begin{pmatrix} 2 & 0 & -20 \\ 0 & 2 & -10 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 20 & 60 & 60 & 20 \\ 10 & 10 & 30 & 30 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 20 & 100 & 100 & 20 \\ 10 & 10 & 50 & 50 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

After scaling the size of the rectangle is $(100 - 20) \times (50 - 10)$, i.e., 80 units × 40 units, i.e., twice that of the original size and the area = 3200 sq.units, i.e. 4 times that of the original area.

6. A triangle PQR has its vertices located at $P(80,50)$, $Q(60,10)$, $R(100,10)$. It is desired to obtain its reflection about an axis, parallel to the Y axis and passing through the point A (30,10). Work out the necessary transformation matrix and also the coordinates of the vertices of the reflected triangle.

Solution

The equation of the mirror line in this case is $x = 30$. So we first translate the PQR triangle and the mirror line +30 units in (–) ve X axis direction to merge the mirror line with Y axis. The translation matrix is therefore

$$\begin{pmatrix} 1 & 0 & -30 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Next we apply the standard equations of reflection about Y axis to the vertices of the triangle for which the transformation matrix is

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

NOTES

Finally we reverse translate the reflected triangle 30 units in (+)ve X axis direction using

$$\begin{pmatrix} 1 & 0 & 30 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The compound matrix is therefore,

$$[T_{\text{COMB}}] = \begin{pmatrix} 1 & 0 & 30 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -30 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -1 & 0 & 60 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The coordinates of the vertices of the reflected triangle is given by,

$$\begin{pmatrix} -1 & 0 & 60 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 80 & 60 & 100 \\ 50 & 10 & 10 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -20 & 0 & -40 \\ 50 & 10 & 10 \\ 1 & 1 & 1 \end{pmatrix}$$

7. A mirror is vertically placed such that it passes through (20,0) and (0,20). Find the reflected view of a triangle with vertices (30,40), (50,50) and (40,70) in this mirror.

Solution

We plot the mirrorline passing through (20,0) and (0, 20). From figure we easily get $\tan\theta = 20/20 = 1$ which implies $\theta = 45°$. To make the line coincident with the X axis we first translate it to make it pass through origin and then rotate it by $\theta = 45°$ about origin. The transformation is given by

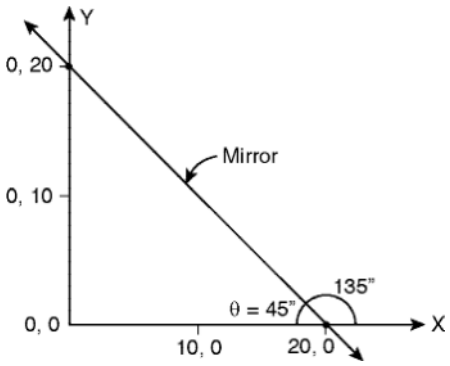


Figure 5.30

$$[T_R]_{45°}\ [T_T]_{-20,0} = \begin{pmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -20 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

These two transformations are also applied on the triangle given. Next we perform reflection of this triangle about the transformed mirrorline i.e. the X axis, using the standard transformation matrix.

$$[T_M]_{y=0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Finally, we apply $[T_R]^{-1}_{45°}$ and $[T_T]^{-1}_{-20,0}$ successively on the reflected triangle to get the desired effect.

NOTES

Thus the resultant compound matrix is

$$[T_{COMB}] = [T_T]^{-1}_{-20,\,0} \quad [T_R]^{-1}_{45^\circ} \quad [T_M]_{y=0} \quad [T_R]_{45^\circ} \quad [T_T]_{-20,\,0}$$

$$= \begin{pmatrix} 1 & 0 & 20 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -20 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & -1 & 20 \\ -1 & 0 & 20 \\ 0 & 0 & 1 \end{pmatrix}$$

The reflected view of the triangle is given by

$$[T_{COMB}]\,[X] = \begin{pmatrix} 0 & -1 & 20 \\ -1 & 0 & 20 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 30 & 50 & 40 \\ 40 & 50 & 70 \\ 1 & 1 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} -20 & -30 & -50 \\ -10 & -30 & -20 \\ 1 & 1 & 1 \end{pmatrix}$$

8. Show that a 2D reflection through X axis followed by a 2D reflection through the line $y = -x$ is equivalent to pure rotation about the origin.

Solution

The homogeneous transformation matrices for reflection about X axis and reflection through $y = -x$ are

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{ respectively.}$$

When applied successively the combination matrix is

$$[T_{COMB}] = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The transformation matrix for rotation about origin by an angle $\theta = 270°$ is

$$[T_R]_{270^\circ} = \begin{pmatrix} \cos 270 & -\sin 270 & 0 \\ \sin 270 & \cos 270 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

This implies $[T_{COMB}] = [T_R]_{270^\circ}$.

Again we find the determinant of $[T_{COMB}]$ here is $1 - (-1) = 1$. Considering the fact that the determinant of any pure rotation transformation matrix is equal to +1 we can conclude from the $[T_{COMB}]$ that the successive translations given here are equivalent to pure rotation, i.e., rotation about origin.

In general, if two pure reflection transformation about line passing through the origin are applied successively, the result is a pure rotation about origin.

NOTES

9. The reflection along the line $y = x$ is equivalent to the reflection along the X axis followed by counter clockwise rotation by θ degrees. Find the value of θ.

Solution

The transformation matrix for reflection about the line $y = x$ is

$$[T_M]_{y=x} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For reflection about X axis, the matrix is

$$[T_M]_{y=0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{(i)}$$

For counter clockwise rotation by θ degrees about origin, the transformation matrix is given by

$$[T_R]_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{(ii)}$$

For successive application of (i) and (ii) the resultant transformation matrix is given by,

$$[T_{COMB}] = [T_R]_\theta [T_M]_{y=0} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ \sin\theta & -\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

As required $[T_{COMB}] = [T_M]_{y=x}$

i.e., $$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ \sin\theta & -\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\Rightarrow \cos\theta = 0; -\cos\theta = 0 \text{ and } \sin\theta = 1$$

$$\Rightarrow \theta = 90^\circ$$

10. In 2D graphics the following transformation matrix would reflect a point about the diagonal line passing through the origin and (10,10)

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

Show that this is same as coordination of matrix for 45 degree clockwise rotation followed by reflection about X axis and finally by counter clockwise rotation by 45 degrees (about origin).

Solution

For 45° clockwise rotation the transformation matrix is

$$[T_R]_{-45^\circ} = \begin{pmatrix} \cos(-45) & -\sin(-45) & 0 \\ \sin(-45) & \cos(-45) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

NOTES

For reflection about X axis the transformation matrix is

$$[T_M]_{y=0} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

For 45° anticlockwise rotation the transformation matrix is

$$[T_R]_{45^o} = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The combination of these matrices yields the transformation matrix

$$[T_{COMB}] = \begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

which is seen to be same as transformation matrix that would reflect a point about the diagonal line passing through origin and (10, 10).

11. Prove that if rotation angle is θ the transformation matrix formed when multiplied by the transformation matrix formed when angle is $-\theta$ is equal to identity matrix.

Solution

$$[T_R]_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$[T_R]_{-\theta} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

$$\text{Now } [T_R]_{-\theta}\,[T_R]_\theta = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos^2\theta + \sin^2\theta & -\cos\theta\sin\theta + \sin\theta\cos\theta & 0 \\ -\sin\theta\cos\theta + \cos\theta\sin\theta & \sin^2\theta + \cos^2\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I, \text{ the identity matrix}$$

12. Show that the 2 × 2 matrix $[T] = \begin{pmatrix} \frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2} \\ \frac{-2t}{1+t^2} & \frac{1-t^2}{1+t^2} \end{pmatrix}$ represents pure rotation.

NOTES

Solution

We know that for pure rotational transformation determinant of the transformation matrix is always equal to 1.

Now determinant of $[T] = \left\{\frac{1-t^2}{1+t^2}\right\}2 - \left\{-\left\{\frac{2t}{1+t^2}\right\}^2\right\}$

$$= \frac{(1-t^2)^2}{(1+t^2)^2} + \frac{4t^2}{(1+t^2)^2} = \frac{(1-t^2)^2 + 4t^2}{(1+t^2)^2} = \frac{1-2t^2+t^4+4t^2}{(1+t^2)^2}$$

$$= \frac{1+2t^2+t^4}{(1+t^2)^2} = \frac{(1+t^2)^2}{(1+t^2)^2} = 1. \quad \text{Hence proved.}$$

13. Prove that for $x = t^2, y = t$ the transformation $[x \quad y \quad 1]\begin{pmatrix} 0 & -2 & 2 \\ -2 & 2 & -2 \\ 1 & 0 & 1 \end{pmatrix}$ yields points that lie on a unit circle.

Solution

The product of the matrices yields the matrix

$$[X'] = [x \quad y \quad 1]\begin{pmatrix} 0 & -2 & 2 \\ -2 & 2 & -2 \\ 1 & 0 & 1 \end{pmatrix}$$

$$= [-2y+1 \quad -2x+2y \quad 2x-2y+1]$$

Replacing x by t^2 and y by t yields

$$[X'] = [-2t+1 \quad -2t^2+2t \quad 2t^2-2t+1]$$

But this is a homogeneous representation of the transformed points,

where $h = 2t^2 - 2t + 1$, $x_h = -2t + 1$, $y_h = -2t^2 + 2t$

So the actual transformed coordinates are those corresponding to $h = 1$ i.e. $\frac{x_h}{h}$ and $\frac{y_h}{h}$

$$\text{i.e. } \frac{-2t+1}{2t^2-2t+1} \text{ and } \frac{-2t^2+2t}{2t^2-2t+1}$$

To prove that these points $[X']$ lie on a unit circle we have to show that $\left(\frac{x_h}{h}\right)^2 + \left(\frac{y_h}{h}\right)^2 = 1$

$$\text{i.e. } \left(\frac{-2t+1}{2t^2+2t+1^2}\right)^2 + \left(\frac{-2t^2+2t}{2t^2-2t+1}\right)^2 = 1$$

Let $2t - 1 = P$

Therefore, $\text{L.H.S.} = \left\{\frac{-(2t-1)}{2t(t-1)+1}\right\}^2 + \left\{\frac{-2t(t-1)}{2t(t-1)+1}\right\}^2$

NOTES

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

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

$$= \frac{(P^4+2P^2+1)}{(P^2+1)^2} = \frac{(P^2+1)^2}{(P^2+1)^2} = 1 \quad \text{Hence proved.}$$

14. Applying a 2D rotation followed by a scaling transformation is the same as applying first the scaling transformation and then the rotation. Justify.

Solution

For rotation, $[T_R]_\theta = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$

For scaling, $[T_s]s_x, s_y = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(a) When rotation is followed by scaling,

$$[T_{COMB}]_{SR} = [T_s]_{s_x, s_y}\,[T_R]_\theta = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x\cos\theta & -s_x\sin\theta & 0 \\ s_y\sin\theta & s_y\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(b) When scaling is followed by rotation,

$$[T_{COMB}]_{RS} = [T_R]_\theta\,[T_S]_{s_x, s_y} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} s_x\cos\theta & -s_x\sin\theta & 0 \\ s_x\sin\theta & s_y\cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Comparing $[T_{COMB}]_{RS}$ with $[T_{COMB}]_{SR}$ we find that they are not identical if $s_x \neq s_y$ i.e., for non-uniform scaling. However for uniform scaling ($s_x = s_y$), the resultant matrices become identical and the comment is justified. Also for $\theta = n\pi$, n being any integer, $\sin\theta$ is 0, $\cos\theta$ is 1 which implies $[T_{COMB}]_{RS} = [T_{COMB}]_{SR}$ irrespective of $s_x = s_y$ or not.

15. 'Successive translation and rotation are not commutative'. Justify. For the given rectangle with corner point A at (x_P, y_P) in Figure 5.31 following two transformation sequences yield identical results.

(a) First translation by ($-x_P$, $-y_P$) and then rotation by θ about A.

(b) First rotation by θ about A and then translation by ($-x_P$, $-y_P$).

Does this contradict the non-commutativity of 'translation – rotation' pair?

Solution

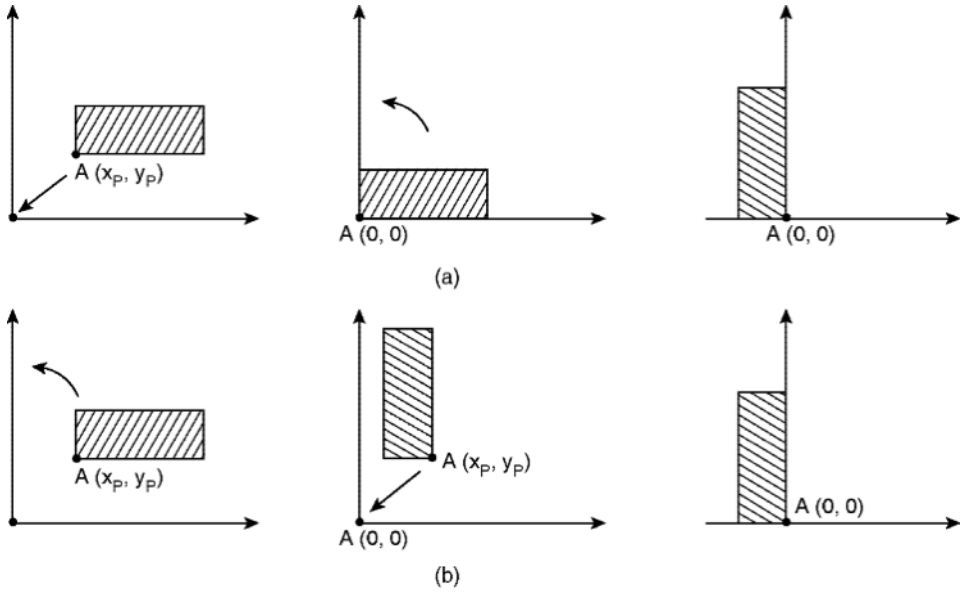


Figure 5.31

First we have to show that changing the order of successive translation and rotation transformation yields non-identical resultant transformation matrix.

Case 1

Translation by $(\Delta x, \Delta y)$ followed by rotation about pivot point (x_P, y_P) by θ

The resultant transformation matrix

$$[T_{COMB}]_{RT} = [T_R]_\theta \; [T_T]_{\Delta x,\, \Delta y}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_P + \sin\theta\, y_P \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta\, x_P \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & \Delta x\cos\theta - \Delta y\, \sin\theta + (1-\cos\theta)x_P + \sin\theta\, y_P \\ \sin\theta & \cos\theta & \Delta x\sin\theta + \Delta y\, \cos\theta + (1-\cos\theta)y_P - \sin\theta\, x_P \\ 0 & 0 & 1 \end{pmatrix}$$

Case 2

Rotation about pivot point (x_P, y_P) by θ followed by translation by $(\Delta x, \Delta y)$

The resultant transformation matrix

$$[T_{COMB}]_{TR} = [T_T]_{\Delta x,\, \Delta y} [T_R]_\theta$$

$$= \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)x_P + \sin\theta\, y_P \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta\, x_P \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & -\sin\theta & (1-\cos\theta)\, x_P + \sin\theta\, y_P + \Delta x \\ \sin\theta & \cos\theta & (1-\cos\theta)y_P - \sin\theta\, x_P + \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

It is clear that $[T_{COMB}]_{TR} \neq [T_{COMB}]_{RT}$ as $\theta \neq 0$

Hence the comment is justified (even for $(x_P = 0, y_P = 0)$ or $\Delta x = \Delta y = 0$).

NOTES

Check Your Progress

1. State true (T) or false (F)
   (a) A pair of parallel lines remain parallel after any 2D transformation.
   (b) In general rotation and scaling are non-commutative.
   (c) X-direction shear w.r.t. X-axis or w.r.t. a line parallel to X-axis doesn't make any difference.
   (d) 'h' can have any value in homogeneous coordinate (xh, yh, h).
   (e) A composite transformation always involves inverse transformation.

NOTES

Translation–rotation non-commutativity holds good for rotation about a fixed pivot point. But in the two sequences sited here, the pivot-point for rotation varies. In sequence (a) the pivot point i.e. the translated point A is the origin whereas in sequence (b) the pivot point is the initial point A i.e. (x_P, y_P). Herein lies the fallacy. Had we rotated the figure about (x_P, y_P) after translation in sequence (a) the finally transformed figure should have been different from that in (b).

16. An observer standing at the origin sees a point $P(1, 1)$. If the point is translated one unit in the vector direction $\boldsymbol{v} = \mathbf{j}$, its new coordinate position is $P'(1, 2)$. Suppose instead that the observer stepped back one unit along the Y axis. What would be the apparent coordinates of P with respect to the observer?

Solution

The problem can be set up as a transformation of coordinate systems.

If we translate the origin O in the direction $\boldsymbol{v} = -\mathbf{j}$ (as the observer stepped back one unit along Y axis). The coordinate of P in this translated system can be found from the following transformations.

$$[T_T]_{0,1}\, P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

So the new coordinates are (1, 2). Also note that from observer's point of view, moving the object forward or stepping back from it (by the same amount) has same effect.

17. An object point $P(x, y)$ is translated in the direction $\boldsymbol{v} = a\mathbf{i} + b\mathbf{j}$ and simultaneously an observer moves in the direction $\boldsymbol{v}$. Show that there is no apparent motion (from the point of view of the observer) of the object point.

Solution

After undergoing the translation the transformed coordinates of p are,

$$P'_{0,0} = [T_T]_{a,b}\, P_{0,0} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+a \\ y+b \\ 1 \end{pmatrix}$$ [The suffix 0,0 implies point coordinates w.r.t global origin]

As the observer has also moved at the same time by $\boldsymbol{v}$, from the observer's frame of reference the coordinates of the translated point are

$$P'_{a,b} = [T_T]_{-a,-b}\, P'_{0,0} = \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x+a \\ y+b \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$ [The suffix a, b implies point coordinates w.r.t origin (a, b) of the moving system]

Thus $P'_{a,b} = P_{0,0}$ which implies that from the observers point of view there is no apparent motion of the object point. In real life we often experience such situation while travelling in a car. When a passing by vehicle travels at same speed in the same direction then it seems to be static to the observer in the car. The motion of the vehicle is noticeable only if there is difference of speed and/or direction of motion between the car and the other vehicle.

18. An object ($ABCD$ rectangle) is defined with respect to a coordinate system whose units are measured in inches. If a local coordinate system (which uses mm as the basic unit) is used to describe the object details ($abcd$ rectangle) as shown in the figure, then indicate the necessary transformation matrix for describing the object in the local coordinate system.

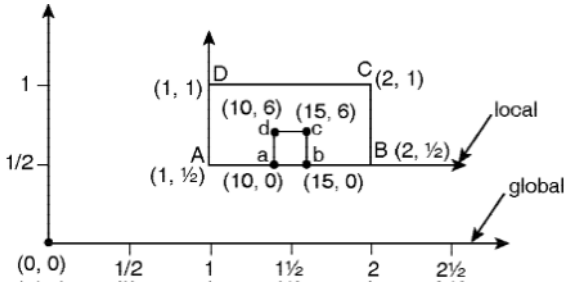


Figure 5.32

NOTES

Solution

This is a case of transformation of coordinate system from global to local involving both translation and scaling. As the origin of the local system shown is (1, 1/2) with respect to global system hence we need to translate the local system by (–1, –1/2) with no change in axis orientation. This should be followed by a coordinate scaling transformation so that each unit (mm) of the local system is scaled to match with that (1 inch) of the global system. Since there are about 25mm to an inch, the scale factors should be $s_x = 25$ and $s_y = 23$. Thus the necessary transformation matrix is given by

$$[T_s]_{25,25}\,[T_T]_{-1,-1/2} = \begin{pmatrix} 25 & 0 & 0 \\ 0 & 25 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1/2 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 25 & 0 & -25 \\ 0 & 25 & -25/2 \\ 0 & 0 & 1 \end{pmatrix}$$

Remember, if you reverse the order of translation and scaling you will lose the advantage of scaling with respect to origin. Because if we first perform scaling then it has to be performed about (1,1/2) instead of origin.

19. Suppose we want to scale a unit square by a factor 2 along one of its diagonal as shown in the figure. Indicate a single transformation matrix required to achieve this. Also draw a rough sketch of the transformed square.

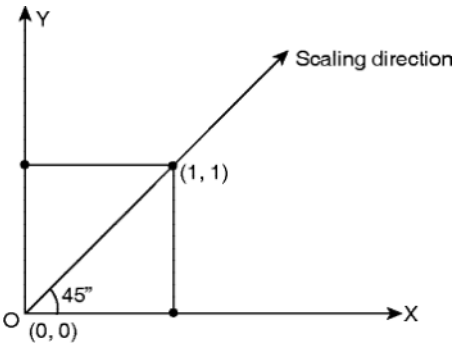


Figure 5.33

Solution

As this is a case of scaling in non-standard direction we first perform a –45° rotation so that the direction of scaling, X' coincides with the X axis while Y' (axis at 90° to X') coincides with Y axis. Then the scaling transformation is applied (with $s_x = 2$, $s_y = 1$) followed by an opposite rotation to return points to their original orientation. The single composite transformation matrix representing these three transformations is given by,

$$[T_{COMB}] = [T_R]_{45^\circ}\,[T_S]_{2,1}\,[T_R]_{-45^\circ}$$

$$= \begin{pmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \cos(-45) & -\sin(-45) & 0 \\ \sin(-45) & \cos 45 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

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

On application of this matrix the transformed vertices of the square are (0,0), (3/2,1/2), (2,2) and (1/2,3/2) forming a parallelogram as shown in the sketch below.

NOTES

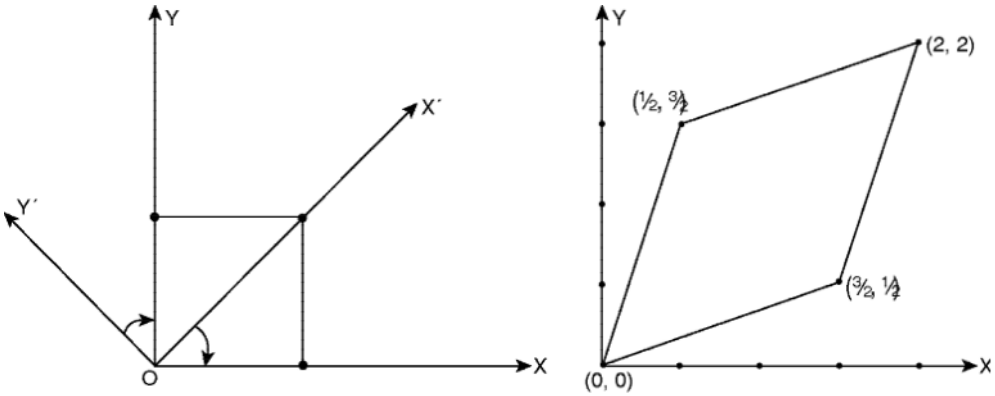


Figure 5.34

20. A triangle is defined by the vertices $\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}$ and the 2×2 translation matrix is $\begin{pmatrix} 3 & 2 \\ 1 & 2 \end{pmatrix}$

(i) Find the area of the triangle.

(ii) Find the vertices of the transformed triangle.

(iii) Find the area of the transformed triangle.

(iv) Find the relationship between area original, area transformed and the determinant of the transformation matrix.

Solution

(i) Area of the original triangle

$$= \frac{1}{2}\begin{vmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix}$$

$$= \frac{1}{2}[1(1-0) - 0(0-0) + (-1)(-1)]$$

$$= \frac{1}{2} \times 2 = 1$$

(ii) Vertices of the transformed triangle are

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

(iii) Area of the transformed triangle

$$= \frac{1}{2}\begin{vmatrix} 3 & 2 & -3 \\ 1 & 2 & -1 \\ 1 & 1 & 1 \end{vmatrix}$$

$$= \frac{1}{2}[3(2+1) - 2(1+1) + (-3)(1-2)]$$

$$= \frac{1}{2} \times 8 = 4$$

(iv) The determinant of the transformation matrix

$$= \begin{vmatrix} 3 & 2 \\ 1 & 2 \end{vmatrix} = (6-2) = 4$$

Hence, area of transformed triangle = (area original) × (determinant of the transformation matrix)

It can be proved that the area of a unit square after being operated by any 2 × 2 transformation matrix becomes equal to the determinant of the transformation matrix.

Since the area of a general figure is the sum of unit squares, the area of any transformed figure A_t is related to the area of the initial figure A_i by $A_t = A_i$ det [T], where [T] is the concerned 2 × 2 transformation matrix.

NOTES

21. P_1P_2 and P_3P_4 are two intersecting straight lines where $P_1 = (1, -1)$, $P_2 = (-1, -2)$, $P_3 = (1, -1)$, $P_4 = (-2, 1)$. Discuss the effect on the length of the lines and the angle between them when operated on by the following 2D transformation matrices respectively.

(i) Translation matrix $\begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$

(ii) 30° rotation matrix $\begin{pmatrix} \frac{\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$

(iii) Non-uniform scaling matrix $\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{pmatrix}$

(iv) Reflection (about $y = -x$) matrix $\begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}$

Solution

If we consider P_1P_2 and P_3P_4 as vectors then

$$\overrightarrow{P_1P_2} = [1-(-1)]\mathbf{i} + [1-(-2)]\mathbf{j} = 2\mathbf{i} + 3\mathbf{j}$$
$$\overrightarrow{P_3P_4} = [1-(-2)]\mathbf{i} + [-1-(-1)]\mathbf{j} = 3\mathbf{i} - 2\mathbf{j}$$

Now length of these lines are $\sqrt{(2)^2 + 3^2}$ and $\sqrt{(3)^2 + (-2)^2}$ respectively, i.e., $\sqrt{13}$ units each.

Now if θ be the angle between the lines then

$$\overrightarrow{P_1P_2} \cdot \overrightarrow{P_3P_4} = |\overrightarrow{P_1P_2}| \; |\overrightarrow{P_3P_4}| \cos\theta = (2)(3) + (3)(-2) = 0$$
$$\Rightarrow \cos\theta = 0 \quad \Rightarrow \theta = 90^\circ$$

Case 1

The given translation matrix operates on the line end points matrix to produce the translated end points given by,

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

so the transformed vectors are

$$\overrightarrow{P_1'P_2'} = [0-(-2)]\mathbf{i} + [3-0]\mathbf{j} = 2\mathbf{i} + 3\mathbf{j}$$

and $$\overrightarrow{P_3'P_4'} = [0-(-3)]\mathbf{i} + [1-3]\mathbf{j} = 3\mathbf{i} - 2\mathbf{j}$$

which are equal and collinear to the original vectors $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_3P_4}$ respectively. Hence the length and internal angle of the lines remains unchanged in translation.

NOTES

Case 2

Here we can directly multiply the transformation matrix with the vector component matrix to get the transformed vector components as

$$\begin{pmatrix} \frac{\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} 2 & 3 \\ 3 & -2 \end{pmatrix} = \begin{pmatrix} \left(\sqrt{3}-\frac{3}{2}\right) & \left(\frac{3\sqrt{3}}{2}+1\right) \\ \left(1+\frac{3\sqrt{3}}{2}\right) & \left(\frac{3}{2}-\sqrt{3}\right) \end{pmatrix}$$

the transformed vectors are

$$\overrightarrow{P_1'P_2'} = \left(\sqrt{3}-\frac{3}{2}\right)\mathbf{i} + \left(1+\frac{3\sqrt{3}}{2}\right)\mathbf{j}$$

$$\overrightarrow{P_3'P_4'} = \left(\frac{3\sqrt{3}}{2}+1\right)\mathbf{i} + \left(\frac{3}{2}-\sqrt{3}\right)\mathbf{j}$$

Now if α be the angle between $\overrightarrow{P_1'P_2'}$ and $\overrightarrow{P_3'P_4'}$ then

$$\begin{aligned} \overrightarrow{P_1'P_2'}\ \overrightarrow{P_3'P_4'} &= |\ \overrightarrow{P_1'P_2'}\ |\ |\ P_3'P_4'\ |\cos\alpha \\ &= \left\{\left(\sqrt{3}-\frac{3}{2}\right)\left(\frac{3\sqrt{2}}{2}+1\right)+\left(1+\frac{3\sqrt{3}}{2}\right)\left(\frac{3}{2}-\sqrt{3}\right)\right\} \\ &= 0 \Rightarrow \cos\alpha = 0 \ \Rightarrow \alpha = 90^\circ = \theta \end{aligned}$$

$$\text{Again length of } \overrightarrow{P_1'P_2'} = |\ \overrightarrow{P_1'P_2'}\ | = \sqrt{\left\{\left(\sqrt{3}-\frac{3}{2}\right)^2+\left(1+\frac{3\sqrt{3}}{2}\right)^2\right\}} = \sqrt{13}$$

$$\text{Length of } \overrightarrow{P_3'P_4'} = |\ \overrightarrow{P_3'P_4'}\ | = \sqrt{\left\{\left(\frac{3\sqrt{3}}{2}+1\right)^2+\left(\frac{3}{2}-\sqrt{3}\right)^2\right\}} = \sqrt{13}$$

Hence in 30° rotation the internal angle and length remain unaltered.

Case 3

Upon scaling the vector component matrix is

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 2 & -3 \\ 3 & -2 \end{pmatrix} = \begin{pmatrix} 1 & \frac{3}{2} \\ 6 & -4 \end{pmatrix}$$

The transformed vectors, $\overrightarrow{P_1'P_2'} = \mathbf{i} + 6\mathbf{j}$ and $\overrightarrow{P_3'P_4'} = \frac{3}{2}\mathbf{i} - 4\mathbf{j}$

If β be the angle between $\overrightarrow{P_1'P_2'}$ and $\overrightarrow{P_3'P_4'}$ then

$$\overrightarrow{P_1'P_2'} \cdot \overrightarrow{P_3'P_4'} = |\ \overrightarrow{P_1'P_2'}\ |\ |\ \overrightarrow{P_3'P_4'}\ |\cos\beta = \frac{3}{2} + (6)(-4) = \frac{-45}{2}$$

$$\Rightarrow \cos\beta = \frac{-45/2}{|\ \overrightarrow{P_1'P_2'}\ |\ |\ \overrightarrow{P_3'P_4'}\ |} = \frac{-45}{\sqrt{37\times 73}}$$

$$\Rightarrow \beta \neq 90^\circ$$

Check Your Progress

2. Which of the following is not a basic transformation?
   (a) Rotation
   (b) Scaling
   (c) Reflection
   (d) None of the above
3. Scaling a point
   (a) doesn't produce any change
   (b) scales the points position vector
   c) makes the point move w.r.t origin
   (d) both (b) and (c) are true
4. For column matrix representation of object points composite transformation is obtained by multiplying matrices in order
   (a) from right to left
   (b) from left to right
   (c) from right to left or left to right
   (d) random (order)

NOTES

$$\text{Length of } \overrightarrow{P_1'P_2'} = |\overrightarrow{P_1'P_2'}| = \sqrt{(1)^2 + (6)^2} = \sqrt{(37)} \neq |\overrightarrow{P_1P_2}|$$

$$\text{Length of } \overrightarrow{P_3'P_4'} = |\overrightarrow{P_3'P_4'}| = \sqrt{\left(\frac{3}{2}\right)^2 + (-4)^2} = \sqrt{\frac{73}{4}} \neq |\overrightarrow{P_3P_4}|$$

So non-uniform scaling doesn't preserve the angle between the lines nor their lengths.

(However it can be proved that for uniform scaling the angle is preserved even though the lengths may vary).

Case 4

Upon reflection about $y = -x$ straight line, the transformed vectors are given by

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

so transformed $\overrightarrow{P_1'P_2'} = 3\mathbf{i} - 2\mathbf{j}$

and $\overrightarrow{P_3'P_4'} = 2\mathbf{i} - 3\mathbf{j}$

If r be the angle between $\overrightarrow{P_1'P_2'}$ and $\overrightarrow{P_3'P_4'}$ then

$$\cos\gamma = \frac{\overrightarrow{P_1'P_2'} \cdot \overrightarrow{P_3'P_4'}}{|\overrightarrow{P_1'P_2'}|\,|\overrightarrow{P_3'P_4'}|}$$

$$= \frac{(-3)(2) + (-2)(-3)}{|\overrightarrow{P_1'P_2'}| \cdot |\overrightarrow{P_3'P_4'}|} = 0$$

$$\Rightarrow \quad = \gamma = 90^\circ = \theta$$

$$\text{Length of } \overrightarrow{P_1'P_2'} = |\overrightarrow{P_1'P_2'}| = \sqrt{(-3)^2 + (-2)^2} = \sqrt{13}$$

$$\text{Length of } \overrightarrow{P_3'P_4'} = |\overrightarrow{P_3'P_4'}| = \sqrt{(2)^2 + (-3)^2} = \sqrt{13}$$

So this reflection preserves length we well as angle (strictly speaking the angle is not preserved in reflection; it can be proved that actually $r = 2\pi - 90°$, i.e., 270°).

5.11 SUMMARY

This unit shows that any 2D geometric transformation can be expressed as 3 by 3 matrix operators so that any sequence of multiple transformations can be concatenated into a single composite matrix. This is an efficient formulation for reducing computation. However, faster transformation can be performed by moving blocks of pixels (representing a displayed object) in the frame buffer using bitBLT raster operations. This doesn't entail matrix multiplication to find transformed coordinates and applying scan conversion functions to display transformed object at new position. A thorough understanding of 2D transformation mathematics is essential for graphics programmers as it shows how complex 3D viewing and geometric transformations should be carried out. The discussions in this unit have been restricted to planar models. Real life effects can be brought using 3D transformations.

5.12 ANSWERS TO 'CHECK YOUR PROGRESS'

1. (a) T, (b) T, (c) F, (d) F (e) F
2. (c)
3. (d)
4. (a)

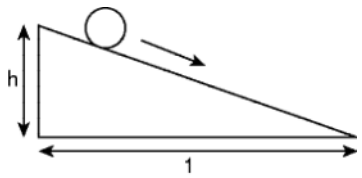
NOTES

5.13 EXERCISES AND QUESTIONS

1. Why are homogeneous coordinates used for transformation computations in Computer Graphics?
2. Show how reflections in the line $y = x$ and in the line $y = –x$ can be performed by a scaling operation followed by a rotation.
3. A triangle is located at $P(10, 40)$, $Q(40, 40)$, $R(40, 30)$. Work out the transformation matrix which would rotate the triangle by 90 degrees (*ccw*) about the point Q. Find the coordinates of the rotated triangle.
4. Consider a standard ellipse centred at (0, 0) where 2a is the length of the major axis along the X axis and 2b is the length of the minor axis along the Y axis. Indicate the necessary transformation that would place the ellipse totally in first quadrant with the major axis passing through (0,0) and (10,10).
5. What is the homogeneous equation of a line? Write the formula for the intersection of two lines specified in homogeneous form. What is the homogeneous equation of a plane?
6. (a) Prove or disprove the following statement:

   'To perform general 2D transformation on a straight line, it is sufficient to transform the end points individually and then draw a straight line between these transformed points.'

   (b) Show that a pair of parallel straight lines remain parallel even after transformation by the general 2 × 2 transformation matrix.
7. Describe the steps needed to stretch a unit square, located at position (4, 5) along its main diagonal from (4, 5) to (6, 10). Provide the transformation matrix.
8. Describe the steps needed to tilt a 2 × 2 square, located at position (4, 2) so that its bottom edge is oriented parallel to the (1, 2) vector. Provide the transformation matrix. Show how you derived the matrix.
9. A circular disc of diameter '*d*' is rolling down the inclined plane starting from rest. Assume there is no slip and develop the set of transformations required to produce this animation.



10. Describe the steps and provide the transformation matrix to convert objects to a cartesian coordinate system in which the origin is located at position (4, 3) w.r.t to the original system and the X axis is now parallel to the line [1, 4].
11. What are the properties for concatenation of transformations? What is the sequence of transformation required to change the position of object shown in Figure 1 to Figure 2.

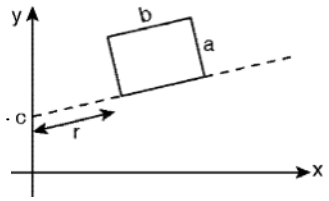


Figure 1

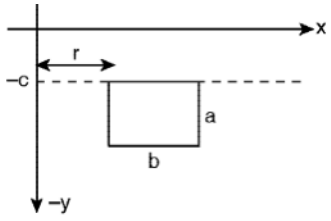
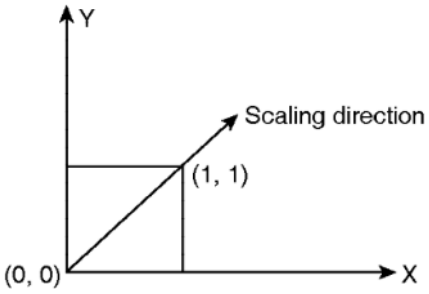
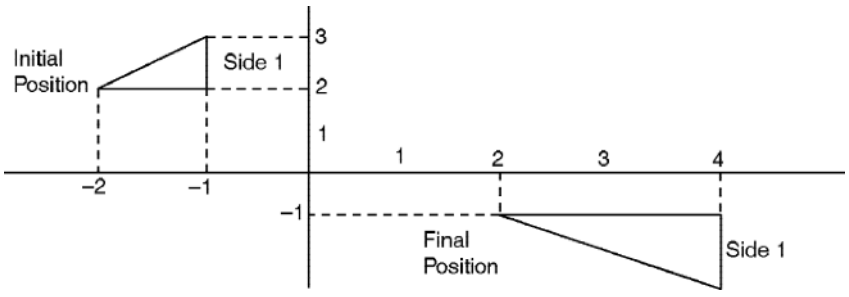


Figure 2

12. The triangle ABC with position vectors [1 0], [0 1], [–1 0] respectively is transformed by the matrix $T = \begin{pmatrix} 3 & 2 \\ -1 & 2 \end{pmatrix}$ to get the triangle A* B* C*. Now do the following:
    (i) Compute area of the transformed triangle without using the position vectors A*, B* or C*.
    (ii) Determine position vectors for A*, B* and C*, find area of transformed triangle from these vectors and verify the area value obtained in i) above.
13. Find the transformation that converts a square with diagonal vertices (0, 3) and (–3, 6) into a unit square at the origin.
14. Prove that 2D rotation and scaling commute if $S_x = S_y$ or if $\theta = n\pi$ for integer n.
15. Let (x_1, y_1, w_1) and $(x_2, y_2, 1)$ be the homogeneous coordinate representations of points P_1 and P_2, respectively, and let $(x_1', y_1', 1)$ be the homogeneous coordinates of P_1', a point obtained after rotating P_1 around P_2 by α degrees. Express x_1' and y_1' in terms of x_1, y_1, w_1, x_2, y_2 and α.
16. Write the transformation for a shear of 45 degrees in the X direction and show that it has the desired effect on the unit square whose vertices are (0, 0), (1, 0), (0, 1) and (1, 1) by transforming its vertices to positions (0, 0), (1, 0), (1, 1) and (2, 1) respectively. Write a similar transformation matrix for a similar shear of 45 degrees in the negative Y direction and, by considering the effect of the transformations on the unit square, show how a clockwise rotation of 90 degrees can be implemented as a succession of shears.
17. Determine a sequence of basic transformations that are equivalent to the *X* direction shearing matrix.
18. In 2D, what angle of rotation is equivalent to a reflection through the origin, meaning a reflection through both the X axis and the Y axis?
19. Find a single transformation matrix to scale a unit square placed at the origin along its diagonal (0, 0) – (1, 1) by a scale factor of 2.



20. A polygon defined by vertices: (x, y), $(3x, y)$, $(3x, 3y)$, $(x, 3y)$ in that order is to be transformed so that its area is reduced by half (without changing its aspect ratio or centroid) and the diagonal passing through $(x, 3y)$, $(3x, y)$ becomes parallel to the x axis. Derive the transformation matrix needed to do this.
21. For the following figure generate transformation matrix.



NOTES

NOTES

22. Write a procedure to compute the elements of the matrix for transforming object descriptions from one cartesian coordinate system to another. The second coordinate system is to be defined with an origin point P_0 and a vector V that gives the direction for the positive Y axis of this system.
23. Determine the homogeneous transformation matrix for reflection about the line $y = mx + b$, or $y = 2x - 6$.
24. Express $P^1 = (x^1, y^1, 1)^T$ in terms of $P = (x, y, 1)^T$ where P is a point of the rectangle whose corner points (x_1, y_1) and $(x_1 + 2a, y_1 + 2b)$ and axes are parallel to coordinate axes and the rectangle is scaled w.r.t centre of mass having scale factors $S_x = 2$ and $S_y = 3$.

5.14 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

UNIT 6 3-DIMENSIONAL GRAPHICS

Structure

NOTES

6.0 INTRODUCTION

For a graphics programmer the real challenge is to image a 3D object in a 2D display plane with realistic effect. It is the third dimension or depth of objects that needs to be handled properly to produce the desired effect. This is achieved by a suitable combination of 3D geometric transformation and 3D viewing transformation where projection plays the key role.

3D geometric transformation is just an extension of the 2D transformation. Though the basic transformation mathematics is easily understandable the important aspect in 3D transformation is proper visualization of object movement in space. Even the object may remain stationary and the viewer's point of view might change. All these factors taken together 3D graphics packages are developed that models real-life 3D objects and allow the user to move, resize or reorient them or even dynamically updates the object's views while simulating users movement within the scene. This unit addresses the cases and mathematics behind the transformation of geometry and projected views of objects from a 3D perspective.

6.1 UNIT OBJECTIVES

- Understanding 3D geometric transformation matrices
- Analyzing the case study of complex 3D geometric transformations derived from basic transformations
- Understanding the classification of projections and studying each projection case with specific examples
- Understanding the stages of 3D viewing transformations and clipping

6.2 3D GRAPHICS

In the previous unit we confined our discussion to two-dimensional (zero thickness) graphic objects only. The present unit is an extension of the previous one in the sense that it now takes z coordinates into consideration apart from the x, y and homogeneous coordinates while the basic approach for representing and manipulating objects remains same. As done in case of 2D we express 3D object transformation in matrix form and any sequence of transformations is represented as a single matrix formed by concatenating the matrices for individual transformations in the same sequence. However, this chapter takes us closer to reality, because real objects are all three dimensional.

NOTES

Just as 2D any 3D transformation can be represented by

$$[X'] = [T][X]$$

where

$[X']$ represents transformed homogeneous coordinates $\begin{pmatrix} x' \\ y' \\ z' \\ h \end{pmatrix}$

$[X]$ represents the untransformed coordinates $\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$

and $[T]$ is the 4×4 transformation matrix (unlike the 3×3 transformation matrix used in 2D).

Again the transformation from homogeneous coordinates to ordinary coordinates is given by

$$\begin{pmatrix} x* \\ y* \\ z* \\ 1 \end{pmatrix} = \begin{pmatrix} x'/h \\ y'/h \\ z'/h \\ h/h \end{pmatrix}$$

6.3 TRANSLATION

If any point $P(x, y, z)$ in 3D space is moved to position $P'(x', y', z')$ such that

$$\begin{aligned} x' &= x + \Delta x \\ y' &= y + \Delta y \\ z' &= z + \Delta z \end{aligned} \tag{1}$$

Δx, Δy, Δz being the displacement of P in three principal directions x, y, z respectively.

Then we can express this 3D translation in homogeneous matrix form as

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

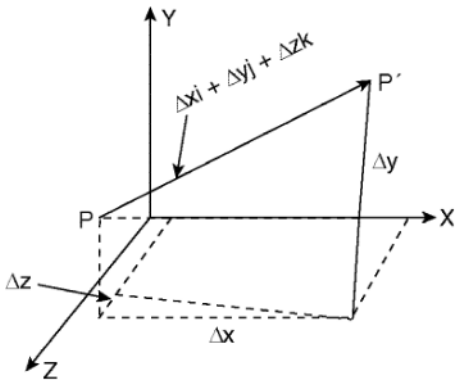


Figure 6.1

NOTES

6.4 SCALING

The matrix expression for scaling transformation, relative to the coordinate origin, will be,

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

$$\Rightarrow \left.\begin{aligned} x' &= s_x\, x \\ y' &= s_y\, y \\ z' &= s_z\, z \end{aligned}\right\} \begin{array}{l} \text{where } s_x,\ s_y,\ s_z \text{ are scale factors in} \\ x, y \text{ and } z \text{ direction respectively.} \end{array} \quad (2)$$

But as we already know this type of scaling repositions an object relative to origin and also changes the shape (i.e. relative dimensions of the object) if the scaling parameters (s_x, s_y, s_z) are not equal.

To induce scaling without allowing any shape change or position change, we have to perform scaling w.r.t. a point lying on the object itself, and that too, using uniform scale factors ($s_x = s_y = s_z$).

Applying methods similar to those used in 2D the transformation matrix for scaling w.r.t. a selected fixed point (x_f, y_f, z_f) can be obtained as,

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

6.5 ROTATION

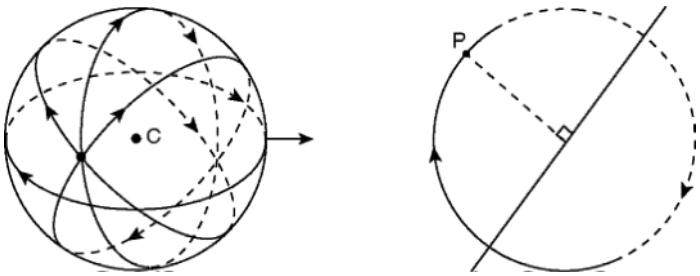


Figure 6.2

We have seen that any 2D-rotation transformation is uniquely defined by specifying a centre of rotation and amount of angular rotation. But these two parameters do not uniquely define a rotation in 3D space because an object can rotate along different circular paths centering a given rotation centre and thus forming different planes of rotation. We need to fix the plane of rotation and that is done by specifying an axis of rotation instead of a centre of rotation. The radius of rotation-path is always perpendicular to the axis of rotation. Before considering three-dimensional rotation about an arbitrary axis we examine rotation about each of the coordinate axes. By convention positive rotation angles produce counter-clockwise rotations about a coordinate axis when looking from positive axis towards the coordinate origin.

NOTES

6.5.1 Rotation about Z Axis

If the rotation is carried out about the Z axis, the z coordinates remains unchanged (because rotation occurs in planes perpendicular to the Z axis) while x and y coordinates behave exactly the same way as in two dimensions. Rotation of any point $P(x, y, z)$ about Z axis by an amount θ is represented by

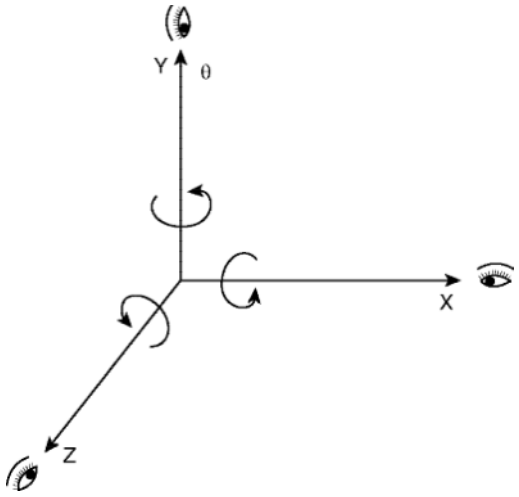


Figure 6.3

$$\left.\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \\ z' &= z \end{aligned}\right\} \tag{3}$$

The corresponding transformation matrix in 4 × 4 form is,

$$[T_R]_{Z,\,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.5.2 Rotation about X Axis

Here rotation takes place in planes perpendicular to X axis hence x coordinate does not change after rotation while y and z coordinates are transformed. The expression can be derived similarly as before if we replace the Z axis in Figure 6.3 with X axis and the other two axes accordingly, maintaining right-handed coordinate system.

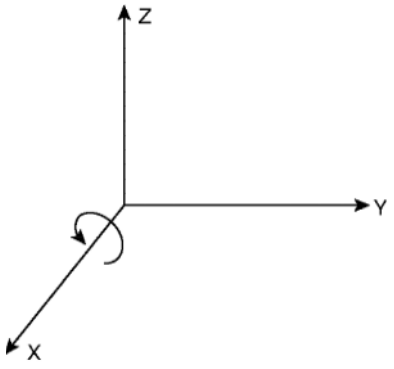


Figure 6.4

Looking at the figure replace z with x, y with z and x with y in eqn. (3) to obtain the required expressions as,

$$\left.\begin{aligned} y' &= y\cos\theta - z\sin\theta \\ z' &= y\sin\theta + z\cos\theta \\ x' &= x \end{aligned}\right\} \tag{4}$$

The transformation matrix is thus

$$[T_R]_{X,\,\theta} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

NOTES

6.5.3 Rotation about Y Axis

This time we replace Z axis with Y axis and X and Y axis with Z and X axis respectively, in Figure 6.3 to yield Figure 6.5. Accordingly modifying eqn. (3), we get

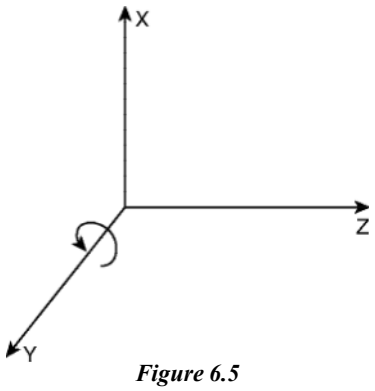


Figure 6.5

$$\begin{aligned} z' &= z\cos\theta - x\sin\theta \\ x' &= z\sin\theta + x\cos\theta \\ y' &= y \end{aligned} \qquad (5)$$

The transformation matrix is now

$$[T_R]_{Y,\,\theta} = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.5.4 Rotation about any Arbitrary Axis in Space

To perform rotation about a line that is not parallel to one of the coordinate axes we first need to align the line with one of the coordinate axes through some intermediate transformation. Then the actual specified rotation is performed about that coordinate axis, the expressions for which are known standards.

Assuming an arbitrary axis in space passing through the point (x_0, y_0, z_0) and having direction cosines (C_x, C_y, C_z), rotation about this axis by some angle θ is accomplished through the following steps:

Step 1 Translate so that the point (x_0, y_0, z_0) is at the origin of the coordinate system. The required matrix is,

$$[T_T] = \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 2 Perform appropriate rotations to make the axis of rotation coincident with the Z axis. In fact we can align the rotation axis with any of the three coordinate axes. Here our choice

NOTES

of Z axis is arbitrary. In general, making an arbitrary axis passing through the origin coincide with one of the coordinate axes requires two successive rotations about the other two coordinate axes.

A. Here first perform rotation about X axis until the rotation axis is in the ZX plane.

B. Then perform rotation about Y axis until the rotation axis coincides with the Z axis.

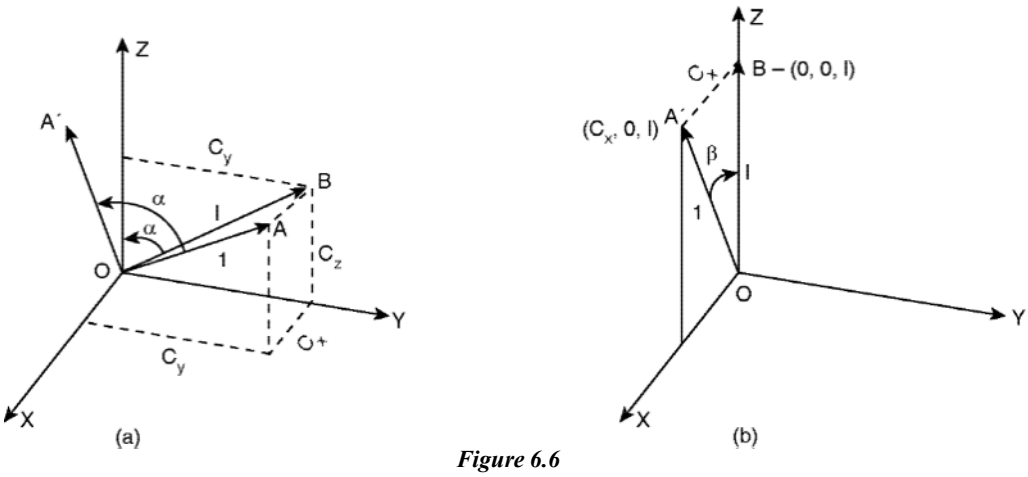


Figure 6.6

To obtain the transformation matrix for step 2A and 2B let us consider Figure 6.6(a) where $\overrightarrow{OA}$ represents the unit vector along the given axis of rotation and $\overrightarrow{OB}$ is the projection of $\overrightarrow{OA}$ on the *YZ* plane.

The amount of rotation required in step 2A to bring $\overrightarrow{OA}$ in the ZX plane is equal to α, the angle between $\overrightarrow{OB}$ and Z axis.

If $|\overrightarrow{OB}| = l$, then

$$\cos\alpha = \frac{C_z}{l}, \sin\alpha = \frac{C_y}{l}$$

Again $l = \sqrt{(C_z^2 + C_y^2)}$

So for step 2A, i.e., for rotation about X axis by angle α, the matrix is,

$$[T_R]_{X,\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{l} & \frac{-C_y}{l} & 0 \\ 0 & \frac{C_y}{l} & \frac{C_z}{l} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

After step 2A, $\overrightarrow{OA}$ becomes $\overrightarrow{OA'}$ and $\overrightarrow{OB}$ becomes $\overrightarrow{OB'}$ both being in the ZX plane. Also the angle between OA' and Z axis is say β in the ZX plane and this is the angle through which the step 2A – transformed rotation axis ($\overrightarrow{OA'}$) should be rotated about Y axis in clockwise direction in order to coincide with Z axis.

From Figure 6.6(b) $\cos\beta = \frac{l}{1}$; $\sin\beta = \frac{C_x}{1} = C_x$

Hence rotation matrix for step 2B is,

$$[T_R]_{y,\beta} = \begin{pmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

Step 3 Rotate about Z axis by the angle θ, the transformation matrix being,

$$[T_R]_{Z,\theta} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Step 4 A. Reverse the rotation about Y axis , the matrix being $[T_R]^{-1}_{Y,\theta}$

B. Reverse the rotation about X axis; matrix $[T_R]^{-1}_{X,\theta}$

Step 5 Reverse the translation carried out in step 1 with the matrix $[T_T]^{-1}$

Putting together all these steps the complete transformation is then,

$$[T_{COMB}] = [T_T]^{-1}\ [T_R]^{-1}_{X,\theta}\ [T_R]^{-1}_{Y,\theta}\ [T_R]_{Z,\theta}\ [T_R]_{Y,\theta}\ [T_R]_{X,\theta}\ [T_T]$$

6.6 PROJECTION

3D objects or scenes which are defined and manipulated using actual physical units of measurement in a 3D space, have to be transformed at one stage from a 3D representation to a 2D representation. Because the image is finally viewed on a 2D plane of the display device. Such 3D-to-2D transformation is called *Projection*. 2D projected images are formed by the intersection of lines called *Projectors* with a plane called the *projection plane*. Projectors are lines from an arbitrary point called the *centre of projection* through each point in an object. Refer Figure 6.7.

There are different categories of projection depending on the direction of projectors and also the relative position of the centre of projection and plane of projection.

6.6.1 Parallel Projection

When the centre of projection is situated at an infinite distance such that the projectors are parallel to each other, the type of projection is called *parallel projection*. In parallel projection, image points are found at the intersection of the view plane with parallel projectors drawn from the object points in a fixed direction. Different parallel projection of the same object results on the same view plane for different direction of projectors.

6.6.2 Orthographic Projection

Orthographic projection occurs when the direction of projection is perpendicular to the plane of projection. Refer Figure 6.7.

NOTES

NOTES

Axonometric projections are orthographic projections in which the direction of projection is not parallel to any of the three principal axes.

The sub-categories of axonometric projections are:

Isometric projection The direction of projection makes equal angles with all three principal axes.

Dimetric projection The direction of projection makes equal angles with exactly two of the principal axes.

Trimetric projection The direction of projection makes unequal angles with the three principal axes.

6.6.3 Oblique Projection

When the angle between the projectors and the plane of projection is not equal to 90° then the projection is called *oblique projection*. Refer Figure 6.7.

Some common sub-categories of oblique projections are:

Cavalier projection The direction of projection is so chosen that there is no foreshortening (refer Figure 6.7) of lines perpendicular to the plane of projection.

Cabinet projection The direction of projection is so chosen that lines perpendicular to the plane of projection are foreshortened by half their lengths.

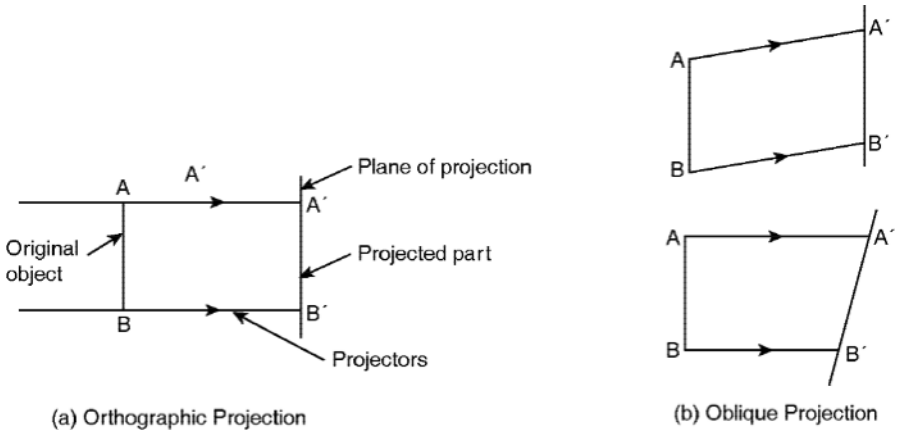


Figure 6.7

6.6.4 Multi-view Orthographic Projection

An orthographic projection always shows the true size and shape of a single plane face (of an object) parallel to the projection plane. Transformation matrices and corresponding expressions for orthographic projection on to the coordinate planes are given below.

For $z = 0$ plane

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (6)$$

For $y = 0$ plane

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (7)$$

For $x = 0$ plane

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad (8)$$

Here (x_P, y_P, z_P) is assumed to be the coordinates of the projected point.

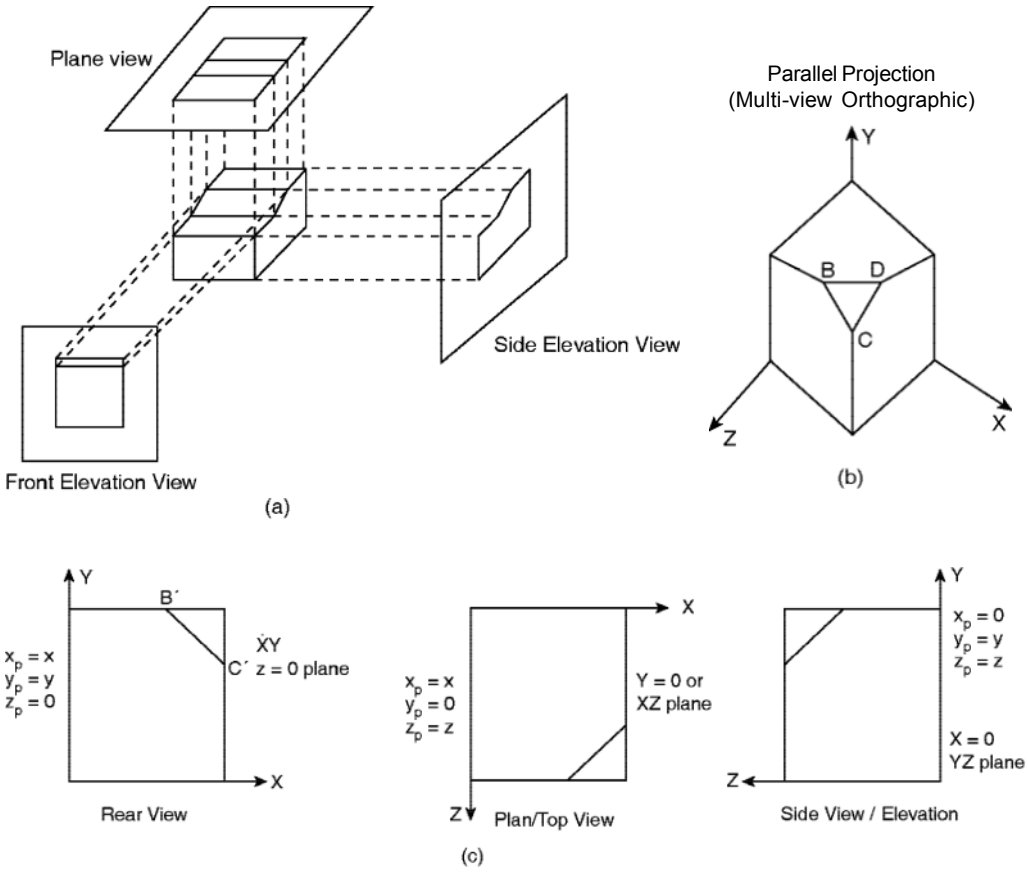


Figure 6.8: *Orthographic projections of an object, displaying plan and elevation views*

Using the above-mentioned equations we can get the projected images of the planes (object faces) which are parallel to the coordinate planes. So instead of a single orthographic projection multiple orthographic projections (i.e., multiview orthographic projection) are necessary to show the true shape and size of different faces of an object. By combining multiple views like top and bottom view, front and rear view, right side and left side view of the same object, the whole object can be visually reconstructed.

The front, right side and top views are obtained by orthographic projection onto the $z = 0$, $x = 0$ and $y = 0$ coordinate planes from centers of projection at infinity on the (+)ve Z, (+)ve X and (+)ve Y axes respectively. Similarly the rear, left side and bottom views are obtained by orthographic projection onto the $z = 0$, $x = 0$ and $y = 0$ planes from centres of projections at infinity on the (–)ve Z, (–)ve X and (–)ve Y axes respectively.

For planes (like plane BCD in Figure 6.8 (b) which are not parallel to any of the coordinate planes we cannot get the true picture by using any of the above transformation equations. To get a true projected picture of the plane BCD we orient the normal to the plane so that it becomes parallel to one of the coordinates axes. This makes the plane BCD parallel to one of the coordinate planes and then it can be orthographically projected to that coordinate plane. Such projected views are known as *auxiliary views*.

NOTES

NOTES

6.6.5 Isometric Projection

We have already mentioned that Isometric projection is a special case of Axonometric projection when the parallel projectors make equal angles with all three principal axes. Isometric projection of an object is obtained by first manipulating (rotating) the object about two axes (X and Y) by a fixed amount followed by orthographic projection onto the projection plane (z = 0). Now we have to derive mathematically the degree of rotation needed about the two axes.

Let us consider a unit cube *OADCEFGB* such that three of its edges *OA*, *OB* and *OC* are along three principal axes X, Y and Z respectively. Now to obtain Isometric projection the object is to be manipulated such that after orthographic projection these three edges are equally foreshortened. *This implies the foreshortening ratios for each projected principal axes are same in isometric projection.*

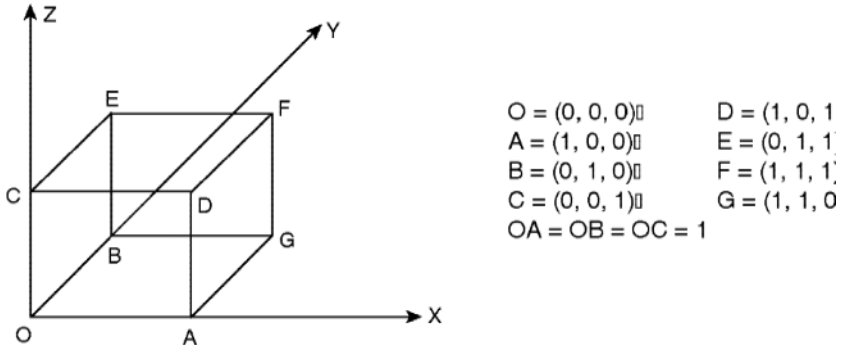


Figure 6.9(a)

Let us also assume that after the resultant transformation matrix for isometric projection is applied on the unit cube the points A (1, 0, 0), B (0, 1, 0) and C (0, 0, 1) transform respectively to A' (x_x, y_x, 0), B' (x_y, y_y, 0) and C' (x_z, y_z, 0). This happens because the X, Y and Z axis on which the points lie transforms respectively to X′, Y′ and Z′ axis. The z coordinates of all the transformed points are 0 because finally the points are projected on z = 0 plane.

So length of the edge along X axis after projection is $OA' = \sqrt{\left(x_x^2 + y_x^2\right)}$

Length of the edge along X axis before projection is OA = 1

So foreshortening factor along X axis is $f_x = \dfrac{OA'}{OA} = \sqrt{(x_x^2 + y_x^2)}$

Similarly foreshortening factors along Y and Z axis are

$$f_y = \frac{OB'}{OB} = \sqrt{(x_y^2 + y_y^2)} \text{ and } f_z = \frac{OC'}{OC} = \sqrt{(x_z^2 + y_z^2)} \text{ respectively.}$$

If, for isometric projection the cube is first rotated by an angle θ_y about Y axis, then through θ_x about by X axis followed by orthographic projection on z = 0 plane then the transformation expression becomes

$$[X'] = [T_{PR}]_{z=0}\ [T_R]_{x,\theta_x}\ [T_R]_{y,\theta_y}\ [X],$$

where $$[T_R]_{x,\theta_x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$[T_R]_{Y,\theta_y} = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$[T_{PR}]_{z=0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The concatenated transformation matrix for isometric projection $[T_{PR}]_{ISO}$ is, thus, given by,

$$[T_{PR}]_{ISO} = [T_{PR}]_{z=0}\,[T_R]_X\,[T_R]_Y$$

$$= \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ \sin\theta_y \sin\theta_x & \cos\theta_x & -\cos\theta_y \sin\theta_x & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

On being operated by the above isometric projection matrix we get the actual coordinate values of isometric points A', B', C' as,

$$A' \equiv x_x = \cos\theta_y,\ y_x = \sin\theta_y \sin\theta_x,\ 0$$
$$B' \equiv x_y = 0,\ y_y = \cos\theta_x,\ 0$$
$$C' \equiv x_z = \sin\theta_y,\ y_z = -\cos\theta_y \sin\theta_x,\ 0$$

Thus, $$f_x^2 = x_x^2 + y_x^2 = \cos^2\theta_y + \sin^2\theta_y \sin^2\theta_x$$
$$f_y^2 = x_y^2 + y_y^2 = \cos^2\theta_x$$
$$f_z^2 = x_z^2 + y_z^2 = \sin^2\theta_y + \cos^2\theta_y \sin^2\theta_x$$

As the foreshortening factors are all equal,

$$f_x^2 = f_y^2 \text{ implying } \cos^2\theta_y + \sin^2\theta_y \sin^2\theta_x = \cos^2\theta_x$$

Replacing $\cos^2\theta_y$ with $1 - \sin^2\theta_y$ and $\cos^2\theta_x$ with $1 - \sin^2\theta_x$ in the above equation we get,

$$\sin^2\theta_y = \frac{\sin^2\theta_x}{(1 - \sin^2\theta_x)}$$

Similarly, $f_z^2 = f_y^2$

$$\Rightarrow \quad \sin^2\theta_y + \cos^2\theta_y \sin^2\theta_x = \cos^2\theta_x$$

$$\Rightarrow \quad \sin^2\theta_y = \frac{1 - 2\sin^2\theta_x}{1 - \sin^2\theta_x}$$

NOTES

NOTES

Equating both the $\sin^2\theta_y$ thus obtained we further get,

$$\sin^2\theta_x = 1 - 2\sin^2\theta_x$$

$$\Rightarrow \quad \sin^2\theta_x = \frac{1}{3} \text{ i.e. } \sin\theta_x = \pm\sqrt{\frac{1}{3}}$$

$$\Rightarrow \quad \theta_x = \pm 35.26^\circ$$

Now
$$\sin^2\theta_y = \frac{\sin^2\theta_x}{1-\sin^2\theta_x}$$

$$= \frac{\frac{1}{3}}{1-\frac{1}{3}}$$

$$= \frac{1}{2}$$

$$\Rightarrow \quad \sin\theta_y = \pm\sqrt{\frac{1}{2}}$$

$$\Rightarrow \quad \theta_y = \pm 45^\circ$$

Therefore there are four possible isometric projections.

(i) $\theta_y = -45^\circ,\ \theta_x = +35.26^\circ,$ (ii) $\theta_y = -45^\circ, \theta_x = -35.26^\circ,$

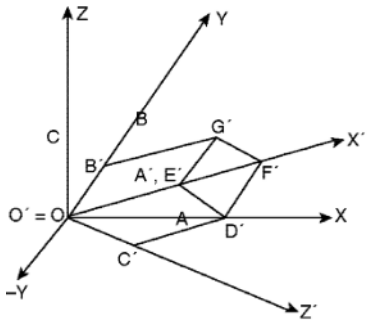
(iii) $\theta_y = +45^\circ, \theta_x = +35.26^\circ,$ (iv) $\theta_y = +45^\circ, \theta_x = -35.26^\circ$

For each case the foreshortening factor is $f = f_x = f_y = f_z = \sqrt{\cos^2}\theta_x = \sqrt{\frac{2}{3}} = 0.8165$

Considering $\theta_y = +45^\circ, \theta_x = +35.26^\circ$ as a sample case for isometric projection the transformation matrix becomes,

$$[T_{PR}]_{ISO} = \begin{pmatrix} \cos 45^\circ & 0 & \sin 45^\circ & 0 \\ \sin 45^\circ \sin 35.26^\circ & \cos 35.26^\circ & -\cos 45^\circ \sin 35.26^\circ & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{or, } [T_{PR}]_{ISO} = \begin{pmatrix} 0.7 & 0 & 0.7 & 0 \\ 0.4 & 0.8 & -0.4 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Refer Figure 6.9(a). Multiplying the original vertices of the cube with $[T_{PR}]_{ISO}$ the transformed isometric vertices are

O′ = O = (0, 0, 0) D′ = (1.4, 0, 1)
A′ = (0.7, 0.4, 0) E′ = (0.7, 0.4, (
B′ = (0, 0.8, 0) F′ = (1.4, 0.8, (
′C = (0.7, –0.4, 1) G′ = (0.7, 1.2, (
OA′ = OB′ = OC′ = .8165

Figure 6.9(b): *The isometric projection of the unit cube OADCEFGB with $\theta_y = +45^\circ, \theta_x = +35.26^\circ$ as viewed from the +ve Z direction on the view plane XY*

NOTES

Figure 6.9 (a) (b) displays the projected (isometric) view of the cube and principal axes w.r.t the original reference frame after $[T_{PR}]_{ISO}$ is applied on the scene. See how the directions of X and Z axis have changed after isometric projection. All the axial lengths have now reduced to 0.8 (strictly .8165) of the original length along the respective transformed axis. *OA* which was 1 unit along X axis has now become $OA' = \sqrt{(0.7^2 + 0.4^2)} = 0.8$ along *X′*. Similarly $OB = 1$ transforms to $OB' = 0.8$, and $OC = 1$ transforms to $OC' = \sqrt{(0.7^2 + 0.4^2)} = 0.8$.

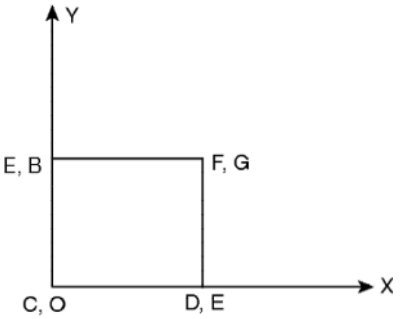


Figure 6.9(c): *The unit cube OADCEFGB when now viewed from +ve Z direction on the view plane XY will look like this. Compare with Figure 6.9(a).The plane CEFD is in front of the plane OBGE. In fact this is a case of orthographic projection with only one side of the cube visible.*

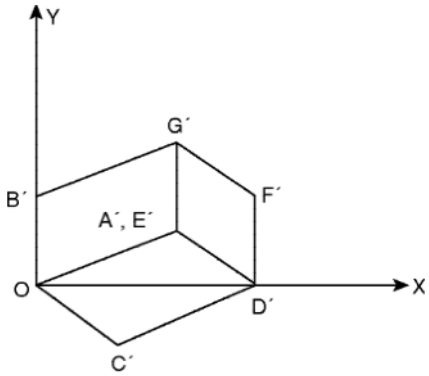


Figure 6.9(d): *Notice that three sides of the cube are visible as compared to only one side in Figure 6.9(c) Isometric projection of the same cube with other combinations of θ_y and θ_x will make the other sides visible on XY plane.*

6.6.6 Oblique Projection

An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane. Figure 6.10 shows an oblique projection of a point $P(x, y, z)$ on the $z = 0$ plane. $P_{ob}(x_p, y_p)$ is oblique projection of the point P and $P_{or}(x, y)$ is the orthographic projection of P on the $z = 0$ plane. Let L be the length of the line joining points $P_{ob}(x_p, y_p)$ and $P_{or}(x, y)$, θ being the angle made by the line L with X axis.

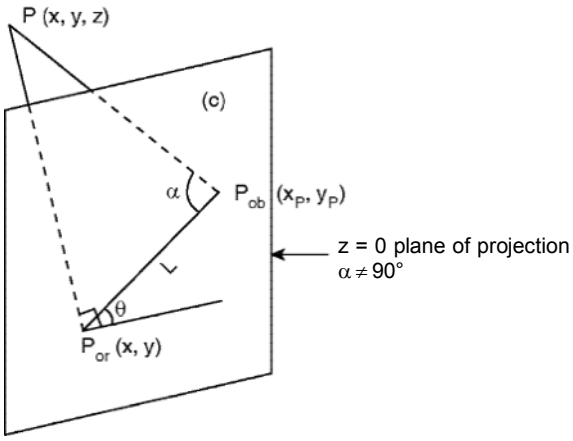


Figure 6.10

Then x_p and y_p, can be expressed as:

$$\left.\begin{aligned} x_P &= x + L\cos\theta \\ y_P &= y + L\sin\theta \end{aligned}\right\} \quad (9)$$

Here the length L is a function of z coordinate. If α is the direction of projection, i.e., the angle made by the projector with the projection plane then,

NOTES

$$\tan\alpha = \frac{z}{L}$$

Thus $$L = \frac{z}{\tan\alpha}$$

$$= zL_1 \quad \left(\text{where } L_1 = \frac{1}{\tan\alpha}\right)$$

When $z = 1,\ L_1 = L$

$$x_p = x + z(L_1 \cos\theta)$$

$$y_p = y + z(L_1 \sin\theta)$$

Considering $z_p = 0$ we get

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & L_1\cos\theta & 0 \\ 0 & 1 & L_1\sin\theta & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

This is the standard matrix expression for oblique projection onto $z = 0$ plane.

When $L_1 = 0$ (implying $\alpha = 90°$) it will represent an orthographic projection on $z = 0$ plane (see eqn. (6))

Commonly used values for angle θ are 30° and 45°. Two commonly used values of α are those for which $\tan\alpha = 1$ and $\tan\alpha = 2$. For the first case, $\alpha = 45°$ and the views obtained are called *cavalier projections*. All lines perpendicular to the projection plane are projected with no change in length.

For $\tan\alpha = 2$, $\alpha = 63.4°$ and the resulting view is called a *cabinet projection*. All lines perpendicular to the viewing surface are projected at one half their length. Cabinet projections appear more realistic than cavalier projections because of the reduction in the length of perpendiculars (i.e., depths of objects).

Foreshortening factors The projected length divided by the actual length is called foreshortening factor. The lines or edges, which are perpendicular to the plane of projection, suffer different shortening in lengths for different values of α. In Figure 6.10 the length of the perpendicular $P P_{or}$ is z. As the oblique projection of P is P_{ob}, the projected length of z is PP_{ob}, i.e., L. Here the foreshortening factor is $\frac{L}{z} = \frac{1}{\tan\alpha}$. For $\alpha = 45°$ foreshortening factor $1/\tan\alpha = 1$ and for $\alpha = 63.4°$, foreshortening factor $1/\tan\alpha = 1/2$. In case of orthographic projection ($\alpha = 90°$) foreshortening factor is 0 as in this case the projected length (L) is equal to 0. In all the above cases value of θ may vary.

Example 6.1 Derive the equation of parallel projection onto the XY plane in the direction of projection V = $a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$

Let $A(x, y, z)$ be any point and $B(x_p, y_p, z_p)$ is the parallel projection of A on XY plane.

The vector $\overrightarrow{AB}$ is defined as

$$\overrightarrow{AB} = (x_P - x)\mathbf{i} + (y_P - y)\mathbf{j} + (z_P - z)\mathbf{k}$$

While the vector V is defined as V = $a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$

Since $\overrightarrow{AB}$ is parallel to V, so

$(x_p - x)\mathbf{i} + (y_p - y)\mathbf{j} + (z_p - z)\mathbf{k} = t(a\mathbf{i} + b\mathbf{j} + c\mathbf{k})$ where t is a constant

NOTES

$$\left.\begin{aligned} \text{i.e.}\quad x_p - x &= at \\ y_p - y &= bt \end{aligned}\right\} \qquad (10)$$

Since $B(x_p, y_p, z_p)$ falls on xy plane, so $z_p = 0$

$$\Rightarrow z = -ct \text{ or } t = -\frac{z}{c}$$

Putting the value of $t = -\frac{z}{c}$ in eqn. (10), we have

$$x_p - x = -a\frac{z}{c}$$

$$\Rightarrow x_p = x - \frac{az}{c}$$

and $$y_p - y = -b\frac{z}{c}$$

$$\Rightarrow y_p = y - \frac{bz}{c}$$

and we have $z_p = 0$

So the transformation matrix is

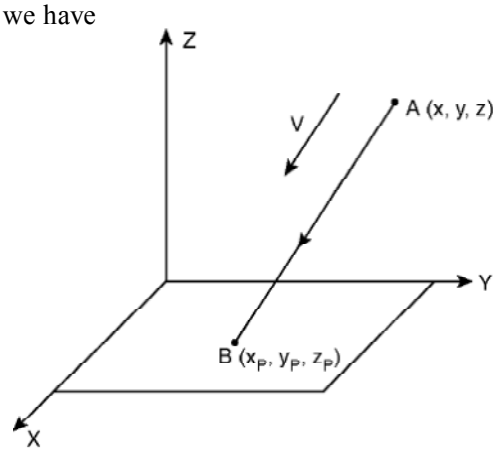


Figure 6.11

$$[T] = \begin{pmatrix} 1 & 0 & \frac{-a}{c} & 0 \\ 0 & 1 & \frac{-b}{c} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

and the equation becomes

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{-a}{c} & 0 \\ 0 & 1 & \frac{-b}{c} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Example 6.2 Derive the general equation of parallel projection onto a given view plane in the direction of a given projector $v = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$. The view plane is defined by a reference point $R(x_0, y_0, z_0)$ and normal vector $N = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$.

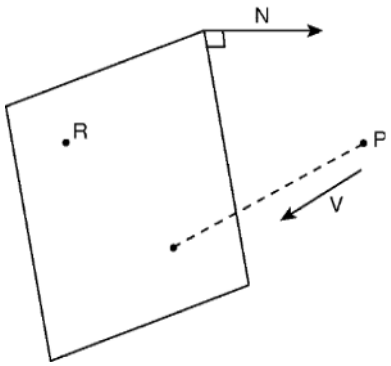


Figure 6.12

Let the point to be projected be $P(x, y, z)$.

(i) First we have to give a translation so that the reference point (x_0, y_0, z_0) becomes the origin, for which the required translation matrix

$$[T_T] = \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(ii) Then we have to rotate the view plane such that the normal vector N becomes parallel to the Z axis. For this first we have to give a rotation about X axis and then a rotation about Y axis.

NOTES

$$N = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$$

$$\therefore \quad N = \frac{N}{|N|} = \frac{n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}}{\sqrt{n_1^2 + n_2^2 + n_3^2}}$$

$$= \frac{n_1}{n'}\mathbf{i} + \frac{n_2}{n'}\mathbf{j} + \frac{n_3}{n'}\mathbf{k} \quad \text{where } n' = \sqrt{\left(n_1^2 + n_2^2 + n_3^2\right)}$$

$$= C_x\mathbf{i} + C_y\mathbf{j} + C_z\mathbf{k}$$

where C_x, C_y, C_z are the direction cosines of the normal and $C_x = \frac{n_1}{n'}$, $C_y = \frac{n_2}{n'}$ $C_z = \frac{n_3}{n'}$

$$\text{Let } d = \sqrt{(C_y^2 + C_z^2)} = \frac{\sqrt{\left(n_2^2 + n_3^2\right)}}{\sqrt{\left(n_1^2 + n_2^2 + n_3^2\right)}}$$

The transformation matrix for rotation about the X axis is:

$$[T_R]_{X,\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{d} & \frac{-C_y}{d} & 0 \\ 0 & \frac{C_y}{d} & \frac{C_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ where α is the rotation angle about X axis

The transformation matrix for rotation about the Y axis is

$$[T_R]_{Y,\beta} = \begin{pmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 1 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} d & 0 & -C_x & 0 \\ 0 & 1 & 0 & 0 \\ C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ where β is the rotation angle about Y axis

(iii) Now that the view plane has transformed to the XY plane, we have to project the transformed point (x, y, z) on the XY plane in the direction of $V = a\mathbf{i} + b\mathbf{j} + c\mathbf{k}$. Let the transformed (x, y, z) point be $P'(x', y', z')$ and the projected point be $P_P(x_p, y_p, z_p)$. So $\overrightarrow{P'P_p} \parallel V$

$\therefore\ (x_p - x')\mathbf{i} + (y_p - y')\mathbf{j} + (z_p)\mathbf{k} = t(a\mathbf{i} + b\mathbf{j} + c\mathbf{k})$, where t is a scalar multiple

$$\Rightarrow \quad x_p - x' = at$$
$$y_p - y' = bt$$
$$z_p - z' = ct$$

since $z_P = 0$ [as the point lies on XY plane]

$$\therefore\ t = \frac{-z'}{c}$$

Substituting the value of t in the other expressions we have

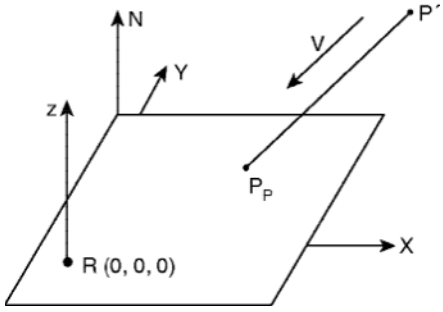


Figure 6.13

$$x_p = x' - \frac{a\,z'}{c} \text{ and } y_p = y' - \frac{b\,z'}{c}$$

$$\begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \frac{-a}{c} & 0 \\ 0 & 1 & \frac{-b}{c} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix}$$

NOTES

(iv) Finally we have to give the inverse rotations $[T_R]^{-1}_{Y,\beta}$ and $[T_R]^{-1}_{X,\alpha}$ and inverse translation $[T_T]^{-1}$ successively where

$$[T_R]^{-1}_{Y,\beta} = \begin{pmatrix} d & 0 & C_x & 0 \\ 0 & 1 & 0 & 0 \\ -C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$[T_R]^{-1}_{X,\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{d} & \frac{C_y}{d} & 0 \\ 0 & \frac{-C_y}{d} & \frac{C_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\text{and } [T_T]^{-1} = \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Concatenating all the matrices we get

$$[T_{COMB}] = [T_T]^{-1}\,[T_R]^{-1}_{X,\alpha}\,[T_R]^{-1}_{Y,\beta}\,[T_{PR}]\,[T_R]_{Y,\beta}\,[T_R]_{X,\alpha}\,[T_T]$$

$$= \begin{pmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 0 & z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{d} & \frac{C_y}{d} & 0 \\ 0 & \frac{-C_y}{d} & \frac{C_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} d & 0 & C_x & 0 \\ 0 & 1 & 0 & 0 \\ -C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & \frac{-a}{c} & 0 \\ 0 & 1 & \frac{-b}{c} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} d & 0 & -C_x & 0 \\ 0 & 1 & 0 & 0 \\ C_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{C_z}{d} & \frac{-C_y}{d} & 0 \\ 0 & \frac{C_y}{d} & \frac{C_z}{d} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\therefore$ The projected point is

$$\therefore \quad \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = [T_{COMB}] \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

6.6.7 Perspective Projection

To obtain a perspective projection of a 3D object we transform points along projection lines which are not parallel to each other and converge to meet at a finite point known as the *projection reference point* or the *center of projection*. The projected view is obtained by calculating the intersection of the projection lines with the view plane.

NOTES

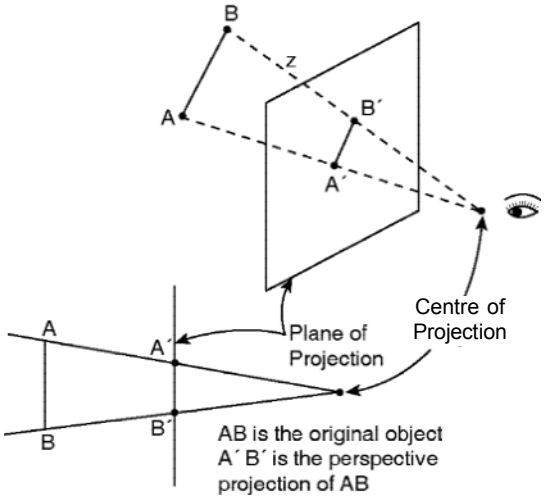


Figure 6.14

By varying the position of projection reference point and the projection plane we will study different standard cases of perspective projection.

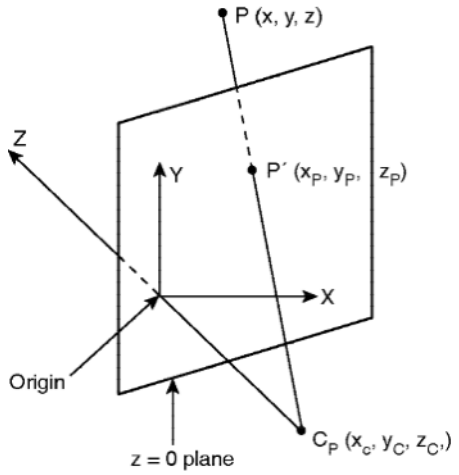


Figure 6.15

Case 1 In Figure 6.15 $P(x, y, z)$ is any point and $C_p(x_c, y_c, z_c)$ is the center of projection. $P_p(x_p, y_p, z_p)$ is the perspective projection of P on $z = 0$ plane with reference to C_P. We have to find the projection matrix

Any point $P'(x', y', z')$ on the line PC_p is given by

$$\left.\begin{aligned} x' &= x + (x_c - x)\,t \\ y' &= y + (y_c - y)\,t \\ z' &= z + (z_c - z)\,t \end{aligned}\right\} \tag{11}$$

where t is the parameter

Eqn. (11) is the parametric equation

For $t = 0$; $x' = x$; $y' = y$; $z' = z$

For $t = 1$; $x' = x_c$; $y' = y_c$; $z' = z_c$

We have to find out for what value of t, $x' = x_p$, $y' = y_p$ and $z' = z_p$. We assume a case where C_P is on the (–)ve Z axis at a distance d from the origin, hence we can write,

$$x_c = 0, y_c = 0, z_c = -d$$

Putting $z' = z_p$ and $z_c = -d$ in eqn. (11)

$$z' = z_p = z + (z_c - z)t$$

$$\Rightarrow \quad z_p = z + (-d - z)t = z - (z + d)t$$

$$\Rightarrow \quad t = \frac{z - z_p}{z + d}$$

As z_p is on XY plane $z_p = 0$, i.e., $t = \frac{z}{z + d}$

Putting this value of t in eqn. (11) we have

$$x_p = x' = x + (x_c - x)t$$

$$= x + (x_c - x)\frac{z}{z + d}$$

but $x_c = 0$ in this case, as C_p is on the Z axis

So $x_p = x - \frac{xz}{z + d} = \frac{xd}{z + d} = \frac{x}{\left(1 + \frac{z}{d}\right)}$

Similarly putting the value of t in eqn. (11) for the value of y_p, we have

$$y_p = y' = y + (y_c - y)t$$

$$= y + (y_c - y)\frac{z}{z + d}$$

Here also $y_c = 0$ as C_p is on the Z axis

So $y_p = y - \frac{yz}{z + d} = \frac{yd}{z + d} = \frac{y}{\left(1 + \frac{z}{d}\right)}$

So the transformation equation involved when the point (x, y, z) is projected on the $z = 0$ plane with C_p on the Z axis

$$\begin{pmatrix} x_h \\ y_h \\ z_h \\ h \end{pmatrix} = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Here h is the homogeneous factor and is equal to $z + d$.

x_p, y_p and z_p can be found by $x_p = \frac{x_h}{h}$, $y_p = \frac{y_h}{h}$ and $z_p = \frac{z_h}{h}$

Case 2 To find the transformation matrix for perspective projection of any point $P(x, y, z)$ onto the $z = k$ plane from the centre of projection C_p at $(0, 0, -d)$. Refer Figure 6.14.

We know that for $z = 0$ plane of projection transformation matrix involved is

$$T = \begin{pmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{pmatrix}$$

In this problem the plane of projection involved is the $z = k$ plane which is parallel to $z = 0$ plane.

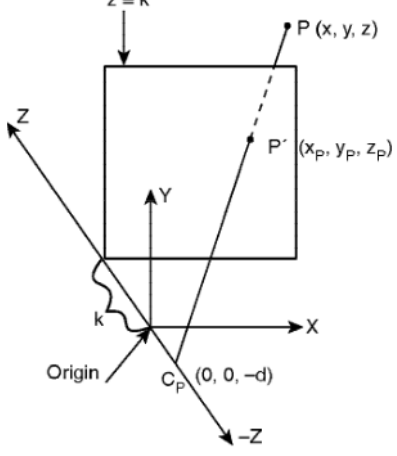


Figure 6.16

NOTES

NOTES

We adopt a similar technique as we have done when the plane of projection was $z = 0$ plane. The position of C_p is unchanged.

If (x', y', z') is any point lying on the line PC_P then

$$\left.\begin{aligned} x' &= x + (0 - x)t \\ y' &= y + (0 - y)t \\ z' &= z + (-d - z)t \end{aligned}\right\} \quad \Rightarrow \quad \left.\begin{aligned} x' &= x - xt \\ y' &= y - yt \\ z' &= z - (z + d)t \end{aligned}\right\}$$

Here t is the parameter.

Now when $z' = z_p$ then,

$$z_p = k = z - (z + d)t$$

$$\Rightarrow \quad t = \frac{z - k}{z + d}$$

Now, $\quad x_p = x' \text{ for } t = \dfrac{z - k}{z + d}$

$$\therefore \quad x_p = x\left\{1 - \frac{z - k}{z + d}\right\} = x\frac{d + k}{z + d}$$

Similarly $\quad y_p = y' \text{ for } t = \dfrac{z - k}{z + d}$

$$\therefore \quad y_p = y\left\{1 - \frac{z - k}{z + d}\right\} = y\frac{d + k}{z + d}$$

Here transformation matrix can be found by

$$x_p = \frac{x_h}{h} = x\frac{d + k}{z + d}$$

$$y_p = \frac{y_h}{h} = y\frac{d + k}{z + d}$$

$$z_p = \frac{z_h}{h} = k\frac{z + d}{z + d} = k$$

where $\quad h = z + d$

$$\therefore \; T = \begin{pmatrix} d + k & 0 & 0 & 0 \\ 0 & d + k & 0 & 0 \\ 0 & 0 & k & kd \\ 0 & 0 & 1 & d \end{pmatrix}$$

Basic relationship between z_p and k is established because any point on $z = k$ plane has a projected value of k.

Case 3 The plane of projection is defined by a reference point $R(x_0, y_0, z_0)$ lying on that plane and a vector normal to that plane $N = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$, the centre of projection is at origin (0, 0, 0). The point $P(x, y, z)$ is to be projected.

NOTES

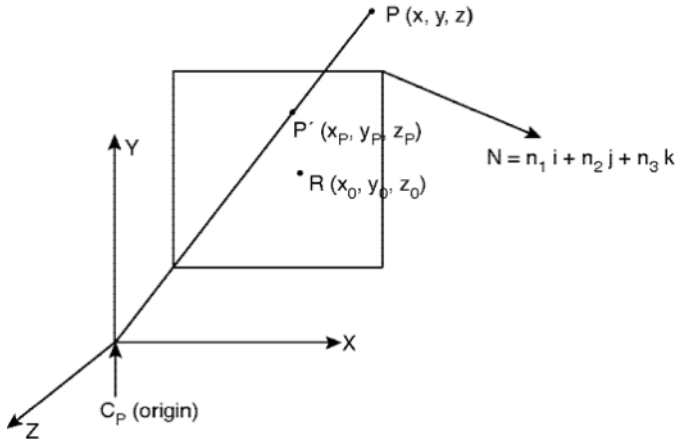


Figure 6.17

Any point x', y', z' lying on the line PC_P is given by

$$x' = 0 + (x - 0)t$$
$$y' = 0 + (y - 0)t$$
$$z' = 0 + (z - 0)t$$

or $x' = xt,\ y' = yt,\ z' = zt$

Let $x' = x_p = xt,\ y' = y_p = yt,\ z' = z_p = zt$

The vector passing through the points $P_p(x_p, y_p, z_p)$ and $R(x_0, y_0, z_0)$ [as both of them lie on the plane of projection] is given by

$$\mathrm{V} = (x_p - x_0)\mathbf{i} + (y_p - y_0)\mathbf{j} + (z_p - z_0)\mathbf{k}$$

The normal vector N is perpendicular to the vector V hence V.N = 0

$$\text{i.e., } n_1(x_p - x_0) + n_2(y_p - y_0) + n_3(z_p - z_0) = 0$$

Now replacing the values of (x_p, y_p, z_p) in the above equation we get,

$$n_1(xt - x_0) + n_2(yt - y_0) + n_3(zt - z_0) = 0$$
$$\Rightarrow \quad t(n_1x + n_2y + n_3z) = n_1x_0 + n_2y_0 + n_3z_0$$
$$\Rightarrow \quad t = \frac{n_1x_0 + n_2y_0 + n_3z_0}{n_1x + n_2y + n_3z}$$

This is the value of t for which x' becomes x_p, y' becomes y_p and z' becomes z_p

so $$t = \frac{d_0}{n_1x + n_2y + n_3z} \quad \text{where} \quad d_0 = (n_1x_0 + n_2y_0 + n_3z_0)$$

since $$x_p = x' = xt \ \text{ for } \ t = \frac{d_0}{n_1x + n_2y + n_3z}$$

so $$x_p = \frac{x\,d_0}{n_1x + n_2y + n_3z}$$

Similarly,, $$y_p = \frac{y\,d_0}{n_1x + n_2y + n_3z}$$

$$z_p = \frac{z\,d_0}{n_1x + n_2y + n_3z}$$

NOTES

so the transformation matrix is

$$T = \begin{pmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & 0 \\ n_1 & n_2 & n_3 & 0 \end{pmatrix}$$

Case 4 Same as case 3, only difference being the centre of projection is not at origin but at (a, b, c). This is the most generalised case of perspective tranformation.

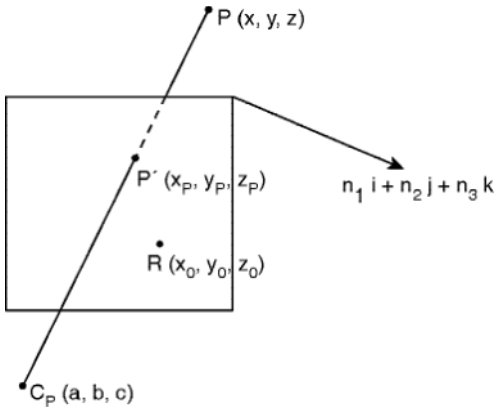


Figure 6.18

Any point (x', y', z') passing through line joining PC_P is given by

$$x' = a + (x - a)t$$
$$y' = b + (y - b)t$$
$$z' = c + (z - c)t$$

Any vector V passing through P_p and R (as they lie on the same plane of projection) is given by

$$\mathrm{V} = (x_p - x_0)\mathbf{i} + (y_p - y_0)\mathbf{j} + (z_p - z_0)\mathbf{k}$$

Now since N is perpendicular to V so N.V = 0.

So $(x_p - x_0)n_1 + (y_p - y_0)n_2 + (z_p - z_0)n_3 = 0$

Considering $x_p = x'$, $y_p = y'$ and $z_p = z'$ for a particular value of t and substituting the values in the above equation we get,

$$[\{a + (x - a)t\} - x_0]n_1 + [\{b + (y - b)t\} - y_0]n_2 + [\{c + (z - c)t\} - z_0]n_3 = 0$$
$$\Rightarrow \quad (x - a)n_1 t + (y - b)n_2 t + (z - c)n_3 t = (x_0 - a)n_1 + (y_0 - b)n_2 + (z_0 - c)n_3$$
$$\Rightarrow \quad t = (x_0 n_1 - a n_1 + y_0 n_2 - b n_2 + z_0 n_3 - c n_3)\ \{(x - a)n_1 + (y - b)n_2 + (z - c)n_3\}$$

Let $\quad d_0 = x_0 n_1 + y_0 n_2 + z_0 n_3$

$$d_1 = a n_1 + b n_2 + c n_3$$

and $\quad d = d_0 - d_1 = x_0 n_1 - a n_1 + y_0 n_2 - b n_2 + z_0 n_3 - c n_3$

$$\therefore \quad t = \frac{d}{(x n_1 + y n_2 + z n_3) - d_1}$$

now $\quad x_p = x'$ for $t = \dfrac{d}{(x n_1 + y n_2 + z n_3) - d_1}$

similarly $y_p = y'$ and $z_p = z'$ for the same value of t

$$x_p = x' = a + (x - a)t$$
$$= a + \frac{(x - a)\,d}{x n_1 + y n_2 + z n_3 - d_1}$$
$$= \frac{(x n_1 + y n_2 + z n_3 - d_1)a + (x - a)d}{x n_1 + y n_2 + z n_3 - d_1}$$
$$= \frac{x(n_1 a + d) + a n_2 y + a n_3 z - a d_1 - a d}{x n_1 + y n_2 + z n_3 - d_1}$$
$$x_p = \frac{x(a n_1 + d) + a n_2 y + a n_3 z - a d_0}{x n_1 + y n_2 + z n_3 - d_1}$$

Proceeding similarly for y_p and z_p we have,

$$y_p = \frac{bn_1x + (bn_2 + d)y + bn_3z - bd_0}{xn_1 + yn_2 + zn_3 - d_1}$$

$$z_p = \frac{cn_1x + cn_2 + (cn_3 + d)z - cd_0}{xn_1 + yn_2 + zn_3 - d_1}$$

NOTES

$\therefore$ The transformation matrix is

$$T = \begin{pmatrix} (an_1 + d) & an_2 & an_3 & -ad_0 \\ bn_1 & (bn_2 + d) & bn_3 & -bd_0 \\ cn_1 & cn_2 & (cn_3 + d) & -cd_0 \\ n_1 & n_2 & n_3 & -d_1 \end{pmatrix}$$

Vanishing points Perspective projection produces realistic views but does not preserve relative proportions of object dimensions. Projections of distant objects are smaller than the projections of objects of the same size that are close to the projection plane (or center of projection). This characteristic feature (anomaly) of perspective projection is known as *perspective foreshortening*.

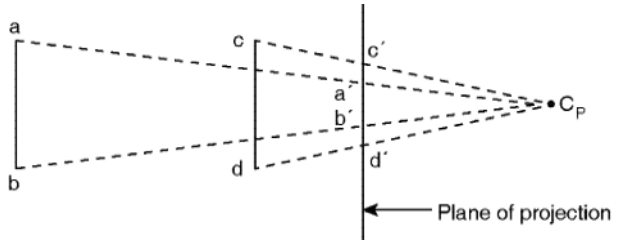


Figure 6.19

Another characteristic feature (anomaly) of perspective projection is the illusion that after projection certain sets of parallel lines appear to meet at some point on the projection plane. These points are called *vanishing points*. For each of the parallel lines we get a vanishing point if they are again not parallel to the plane of projection; parallel lines that are parallel to the view plane will be projected as parallel lines. Principal vanishing points are formed by the apparent intersection of lines parallel to any of the three principal axes X, Y, Z. The number of principal vanishing points is determined by the number of principal axes intersected by the view plane.

Computing vanishing points We will derive the vanishing point for a set of parallel lines (*AB* & *CD* parallel to vector *U*) for perspective projection on a plane defined by normal vector $\mathbf{N} = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$ and in-plane reference point $R_0(x_0, y_0, z_0)$, the centre of projection being $C_p(a, b, c)$.

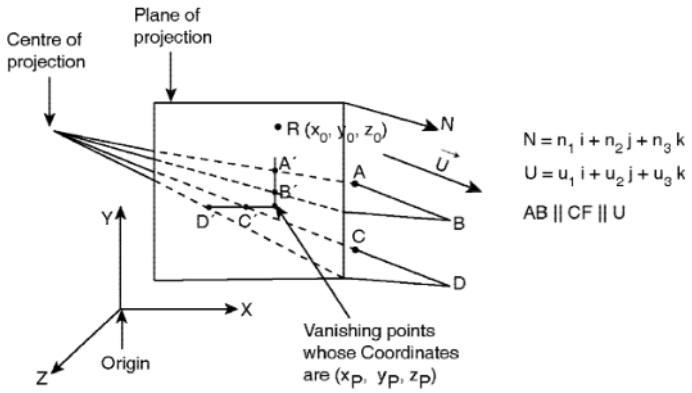


Figure 6.20

NOTES

AB is projected to $A'B'$. So if line AB is extended to infinity, B' meets the vanishing point. Similarly D' becomes the vanishing point if end D of line CD is stretched to infinity. We assume that a point with coordinates (x, y, z) is the variable point on the line AB, i.e., (x, y, z) represents the coordinates of any point on this set of parallel lines to be projected. As this point approaches infinity B' approaches the vanishing point.

We assume another point with coordinates (p, q, r) on any line, say AB of the family of parallel lines.

Line passing through points with coordinates (x, y, z) and (p, q, r) is given by

$(x-p)\mathbf{i} + (y-q)\mathbf{j} + (z-r)\mathbf{k} = t(u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k})$ [since AB is parallel to U – the vector given by $u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}$]

$$\Rightarrow \left.\begin{aligned} x - p &= u_1 t \\ y - q &= u_2 t \\ z - q &= u_3 t \end{aligned}\right\} \Rightarrow \left.\begin{aligned} x &= p + u_1 t \\ y &= q + u_2 t \\ z &= r + u_3 t \end{aligned}\right\}$$

The transformation matrix obtained for perspective projection of any point with coordinate (x, y, z) on the plane of projection defined by a normal vector $\mathbf{N} = n_1\mathbf{i} + n_2\mathbf{j} + n_3\mathbf{k}$ and a reference point (x_0, y_0, z_0) is

$$[T] = \begin{pmatrix} d + an_1 & an_2 & an_3 & -ad_0 \\ bn_1 & d + bn_2 & bn_3 & -bd_0 \\ cn_1 & cn_2 & d + cn_3 & -cd_0 \\ n_1 & n_2 & n_3 & -d_1 \end{pmatrix}$$

when the centre of projection C_P is at (a, b, c).

Now the homogeneous coordinate of the projected point (x_p, y_p, z_p) is given by

$$\begin{aligned} x_h &= (d + an_1)x + an_2 y + an_3 z - ad_0 \\ y_h &= bn_1 x + (d + bn_2)y + bn_3 z - bd_0 \\ z_h &= cn_1 x + cn_2 y + (d + cn_3)z - cd_0 \\ h &= n_1 x + n_2 y + n_3 z - d_1 \end{aligned}$$

where $d_0 = n_1 x_0 + n_2 y_0 + n_3 z_0$

$d_1 = n_1 a + n_2 b + n_3 c$

$d = d_0 - d_1$

Now $x_p = x_h / h$

$$= \frac{(d + an_1)x + an_2 y + an_3 z - ad_0}{n_1 x + n_2 y + n_3 z - d_1}$$

or $$x_p = \frac{a(n_1 x + n_2 y + n_3 z) + dx - ad_0}{n_1 x + n_2 y + n_3 z - d_1}$$

$$= \frac{a}{1 - \dfrac{d_1}{n_1 x + n_2 y + n_3 z}} + \frac{dx}{n_1 x + n_2 y + n_3 z - d_1} - \frac{ad_0}{n_1 x + n_2 y + n_3 z - d_1} \qquad (12)$$

The vanishing point occurs corresponding to the projection of a point (x, y, z) which is at infinite distance, i.e., when

$$\left.\begin{aligned} x &\to \infty \\ y &\to \infty \\ z &\to \infty \end{aligned}\right\}$$

i.e., when $t \to \infty$

In eqn. (12), the term $\dfrac{a}{\dfrac{1-d_1}{n_1x+n_2y+n_3z}} = \dfrac{a}{\dfrac{1-d_1}{\infty}}$ [Since $x, y, z \to \infty$]

$$= \frac{a}{1-0} = a$$

In eqn. (12) the term $= \dfrac{dx}{n_1x+n_2y+n_3z-d_1}$

$$= \frac{d(p+u_1t)}{[\{n_1(p+u_1t)+n_2(q+u_2t)+n_3(r+u_3t)\}-d_1]}$$

$$= \frac{dp}{[\{n_1(p+u_1t)+n_2(q+u_2t)+n_3(r+u_3t)\}-d_1]} + \frac{du_1t}{[\{n_1(p+u_1t)+n_2(q+u_2t)+n_3(r+u_3t)\}-d_1]}$$

$$= \frac{dp}{\infty} + \frac{du_1}{[\{n_1(p+u_1t)+n_2(q+u_2t)+n_3(r+u_3t)\}-d_1]/t} \text{[since } t \to \infty]$$

$$= 0 + \frac{du_1}{[(nu_1+n_2u_2+n_3u_3)+\{(n_1p+n_2q+n_3r)/t\}-(d_1/t)]}$$

$$= \frac{du_1}{n_1u_1+n_2u_2+n_3u_3} \quad \left[\because (n_1p+n_2q+n_3r)/t = 0 \text{ and } (d_1/t) = 0 \text{ when } t \to \infty\right]$$

In eqn. (12) the term $\dfrac{ad_0}{(n_1x+n_2y+n_3z)-d_1} = \dfrac{ad_0}{\infty} = 0$ [since $x, y, z \to \infty$]

In the limiting condition the expression for x_p, i.e., the coordinate of the vanishing point becomes,

$$x_p = a + \frac{du_1}{n_1u_1+n_2u_2+n_3u_3} = a + du_1/k$$

where k is the dot product of vectors **N** and **U**

Similarly we can find the limits of y_p and z_p as $x, y, z, \ t \to \infty$

So the coordinate of the vanishing point for perspective projection in the direction **U** are

$$\begin{aligned} x_{\mathbf{VP}} &= a + du_1/k \\ y_{\mathbf{VP}} &= b + du_2/k \\ z_{\mathbf{VP}} &= c + du_3/k \end{aligned} \tag{13}$$

Note that when **U** is parallel to the plane of projection, k = **N**. **U** = 0 and $x_{\mathbf{VP}}$, $y_{\mathbf{VP}}$, $z_{\mathbf{VP}}$ becomes undefined. This points to the fact that there will be no vanishing point for a set of parallel lines which are parallel to the projection plane.

Principal vanishing points Principal vanishing points correspond to the vector direction parallel to the three principal axes directions. There are three principal vanishing point P_{VP1}, P_{VP2} and P_{VP3}.

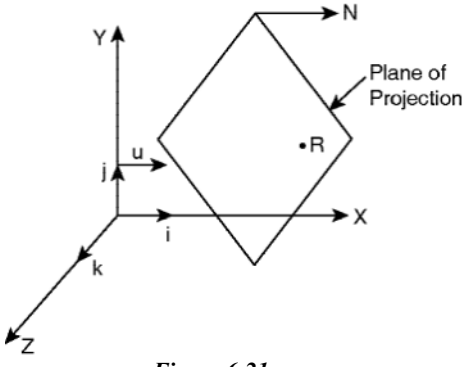


Figure 6.21

NOTES

Check Your Progress

1. To align an arbitrary axis in space with Z axis requires
   (a) rotation of the axis about X-axis
   (b) rotation of the axis about Y-axis
   (c) successive rotation of the axis about X and Y axis
   (d) rotation of the axis by the angle made by the axis with XY-plane
2. Most realistic projection is produced by
   (a) isometric projection
   (b) perspective projection
   (c) cavalier projection
   (d) trimetric projection
3. Foreshortening factor for any orthographic projection is
   (a) 0
   (b) 1
   (c) 0 for perpendicular lines and 1 for parallel lines w.r.t. projection plane
   (d) can't be said in general

NOTES

For $\mathbf{U} = \mathbf{i}$, $P_{VP} = P_{VP1}$

For $\mathbf{U} = \mathbf{j}$, $P_{VP} = P_{VP2}$

For $\mathbf{U} = \mathbf{k}$, $P_{VP} = P_{VP3}$

This is true only if the vector direction is not parallel to the plane of projection .

When **U** is parallel to **i** and not parallel to the plane of projection, then,

$$\mathbf{U} = \mathbf{i}$$

$$\therefore u_1 = 1,\ u_2 = 0,\ u_3 = 0$$

So from eqn. (13)
$$\left.\begin{aligned} x_{VP1} &= a + d/n_1 \\ y_{VP1} &= b \\ z_{VP2} &= c \end{aligned}\right\} P_{VP1}$$

When **U** is parallel to **j** and not parallel to the plane of projection.

$$u_1 = 0,\ u_2 = 1,\ u_3 = 0$$

$$\left.\begin{aligned} x_{VP2} &= a \\ y_{VP2} &= b + d/n_2 \\ z_{VP2} &= c \end{aligned}\right\} P_{VP2}$$

If **U** is parallel to **k** and not parallel to the plane of projection

$$\left.\begin{aligned} X_{VP3} &= a \\ Y_{VP3} &= b \\ Z_{VP3} &= c + d/n_3 \end{aligned}\right\} P_{VP3}$$

Depending on the number of principal axes intersecting the plane of projection, the number of principal vanishing points on that plane of projection vary from a minimum of one to a maximum of three. Accordingly the perspective projection is classified as one principal vanishing point, two principal vanishing point or three principal vanishing point projections.

One point or one principal vanishing point perspective projection occurs when any one principal axes intersects the plane of projection. For example only a Z axis vanishing point occurs when, the Z axis intersects the plane of projection whereas X, Y axes and XY plane remains parallel to the projection plane. The only vanishing point is P_{VP3}: $x = a$, $y = b$, $z = c + d/n_3$

Two point or two principal vanishing point perspective projection occurs when the plane of projection intersects exactly two of the principal axes. For example when the projection plane intersects the X and Y axes and the Z axis remains parallel to the projection plane. The two principal vanishing points formed in X and Y axis direction are,

$$P_{VP1} : x = a + d/n_1,\ y = b,\ z = c$$

$$P_{VP2} : x = a,\ y = b + d/n_2,\ z = c$$

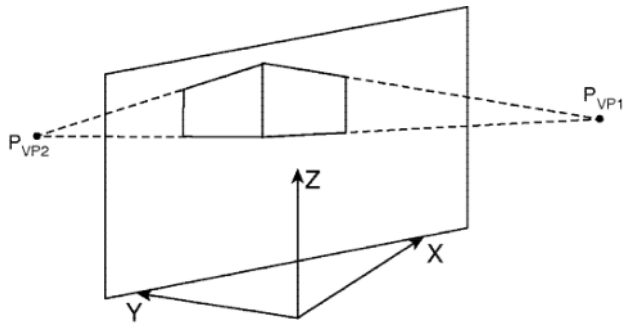


Figure 6.22: *Cube-edges parallel to X and Z axis form 2 vanishing points P_{VP1} and P_{VP3} .*

Three point or three principal vanishing point projection occurs when the projection plane intersects all three of the principal axes, i.e., none of the principal axes is parallel to the projection plane. Here the principal vanishing points are

$$P_{VP1} : x = a + d/n, \quad y = b, \quad z = c$$
$$P_{VP2} : x = a, \quad y = b + d/n_2 \quad z = c$$
$$P_{VP3} : x = a, \quad y = b \quad z = c + d/n_3$$

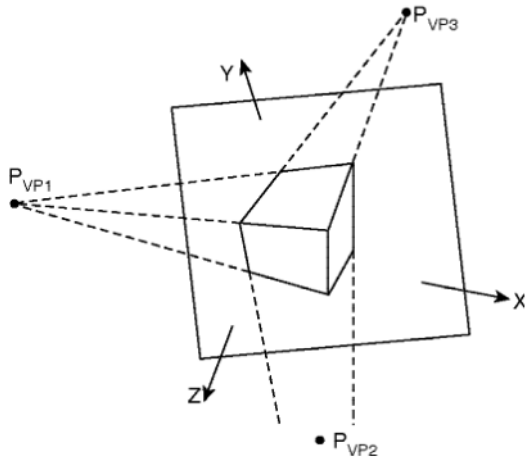


Figure 6.23: *As all the three principal axes intersects the projection plane cube-edges parallel to X, Y, and Z axes form vanishing points P_{VP1}, P_{VP2} and P_{VP3} respectively.*

Example 6.3 (Perspective Projection)

A cube has its vertices located at *A*(0, 0, 10), *B*(10, 0, 10), *C*(10, 10, 10), *D*(0, 10, 10), *E*(0, 0, 0), *F*(10, 0, 0), *G*(10, 10, 0), *H*(0, 10, 0). The Y axis is vertical and Z axis is oriented towards the viewer. The cube is being viewed from the point (0,20,80). Work out the perspective view of the cube on the XY plane.

Plane of projection is the XY plane. Centre of projection is at (0,20,80). Let the coordinates of a projected point be (x_p, y_p, z_p), the coordinates of the point before projection being (x, y, z).

As the centre of projection , the point to be projected and the projected point lie on the same line of view.

$$\left.\begin{aligned} x_p &= x + (0 - x)t \\ y_p &= y + (20 - y)t \\ z_p &= z + (80 - z)t \end{aligned}\right\} t, \text{ being a scalar}$$

Here $z_p = 0$ since the plane of projection is the XY plane

$$\therefore \; z + (80 - z)\, t = 0$$

$$\therefore \; t = \frac{z}{z - 80} \qquad (14)$$

Substituting the value of t in equation (14) we have

$$x_p = x + (0 - x)\, t = x + (0 - x)\, \frac{z}{z - 80}$$

$$= \frac{xz - 80x - xz}{z - 80} = \frac{-80x}{z - 80} = \frac{80x}{80 - z}$$

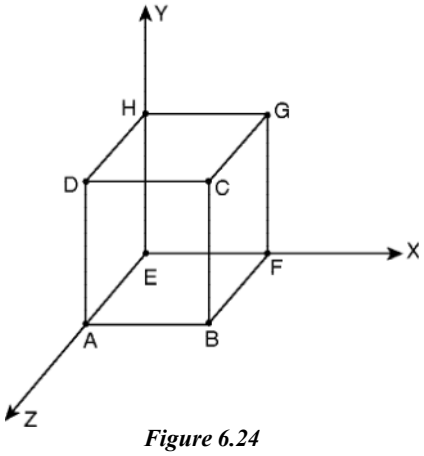


Figure 6.24

NOTES

$$x_p = \frac{80x}{80 - z}$$

$$y_p = y + (20 - y)\ t$$

$$= y + (20 - y)\ \frac{z}{z - 80} = \frac{yz - 80y + 20z - yz}{z - 80}$$

$$\therefore \quad y_p = \frac{20z - 80y}{z - 80}$$

$$x_p = x_h / h,\ y_p = y_h / h \text{ and } z_p = z_h / h \quad \text{where } h = z - 80$$

$$\therefore \begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} -80 & 0 & 0 & 0 \\ 0 & -80 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -80 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

On applying the above perspective projection matrix, the projected vertices of the cube are,

$$A' : \left(0, \frac{-20}{7}, 0\right), \quad B' : \left(\frac{80}{7}, \frac{-20}{7}, 0\right), \quad C' : \left(\frac{80}{7}, \frac{60}{7}, 0\right)$$

$$D' : \left(0, \frac{60}{7}, 0\right), \quad E' : (0, 0, 0), \quad F' : (10, 0, 0)$$

$$G' : (10, 10, 0) \quad H' : (0, 10, 0)$$

6.7 GRAPHICS COORDINATE SYSTEMS AND VIEWING PIPELINE

Computer generation of a view of an object on a display device requires stage-wise transformation operations of the object definitions in different coordinate systems. We present here a non-mathematical introduction to each of these coordinate systems to help you develop an intuitive notion of their use and relationship to one another.

Global Coordinate System, also called the *World Coordinate System* is the principal frame of reference in which all other coordinate systems are defined. This three-dimensional system is the basis for defining and locating in space all objects in a computer graphics scene, including the observer's position and line of sight. All geometric transformations like translation, rotation, scaling, reflection etc. are carried out with reference to this global coordinate system.

A *Local Coordinate System,* or a *Modeling Coordinate System* is used to define the geometry of an object independently of the global system. This is done for the ease of defining the object-details w.r.t. reference point and axes on the object itself. Once you define the object 'locally', you can place it in the global system simply by specifying the location and orientation of the origin and axes of the local system within the global system, then mathematically transforming the point coordinates defining the object from the local to the global system.

The *Viewing Coordinate System* locates objects in three-dimensional space relative to the location of the observer. We use it to simplify the mathematics for projecting an image of the object onto the projection plane. To establish the viewing coordinate reference frame first define an *eyepoint* P_E (eye of the observer) or *view reference point*. Next, specify the direction of the observer's line of sight in either of the two ways: as a set of direction angles (or direction cosines) in the global system, or by specifying the location of a *viewpoint* P_V or *look-at point* . This directed line segment from the eye point to the look-at point is also referred to as the *view-plane normal vector* **N** or viewing axis Z_V. A *view plane* or *projection plane* is then set up perpendicular to **N** or Z_V. The (+)ve direction of the Y_V

axis of the viewing coordinate system is called the *view-up vector* with the view reference point being the origin of the system.

Different views of an object is obtained on the view plane either by moving the object and keeping the eyepoint fixed or by moving the eyepoint keeping the object fixed. However the later technique is assumed by most of the computer graphics application to create a new view of the object.

The two-dimensional *Device Coordinate System* (or *Screen Coordinate System* for display monitor) locates points on the display/output of a particular output device such as graphics monitor or plotter. These coordinates are integers in terms of pixels, addressable points, inches, cms etc.

The *Normalized Device Coordinates* are used to map world coordinates in a device independent two-dimensional pseudospace within the range 0 to 1 for each of x and y before final conversion to specific device coordinates.

The modeling and world coordinate positions can be any floating-point values; normalized coordinates (x_{nc}, y_{nc}) satisfy the inequalities: $0 \le x_{nc} \le 1$, $0 \le y_{nc} \le 1$; and the device coordinates are integers within the range (0, 0) to $(x_{\max}, y_{\max})$, with $(x_{\max}, y_{\max})$ depending on the resolution of a particular output device.

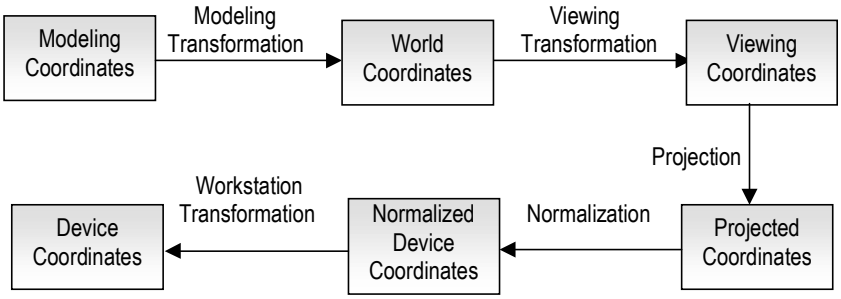


Figure 6.25: *Graphics Coordinate System*

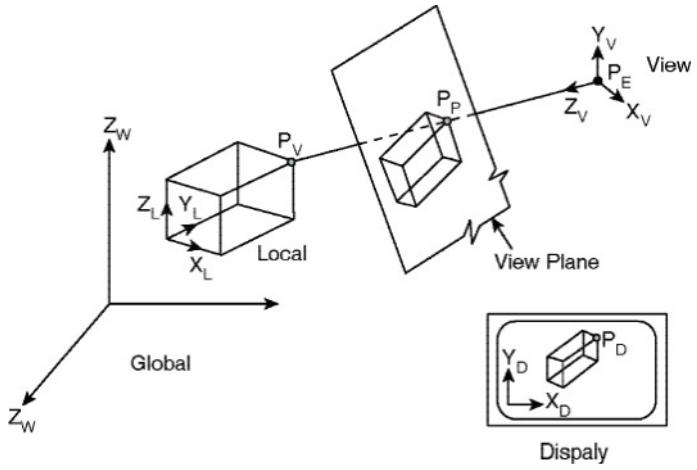


Figure 6.26: *Three-Dimensional Viewing Pipeline*

You will frequently see the terms *object space* and *image space*. Object space corresponds to the world coordinate system, image space to the display-screen coordinate system. Object space is an unbounded, infinite collection of continuous point. Image space is a finite 2D space. In 2D, we simply specify a window in the object space and a viewport in the display surface. Conceptually, 2D objects in the object space are clipped against the window and are then transformed to the normalized viewport and finally to the viewport for display using standard window-to-viewport mapping.

NOTES

Check Your Progress

4. The parametric equation of a line passing through point (p, q, r) and parallel to vector a **i** + b **j** + c **k** is
   (a) x = p + at, y = q + bt, z = r + ct
   (b) x = at, y = bt, z = ct
   (c) x – a = pt, y – b = qt, z – c = rt
   (d) (a/p)x + (b/q)y + (c/r)z = 0
5. 3D viewing transformation causes
   (a) mapping of a 3D object to a view volume
   (b) projection of a 3D object on to a projection plane
   (c) transformation of world coordinates to device coordinates

NOTES

Unlike 2D, 3D objects are conceptually clipped against a *view volume* and are then projected. The contents of the projection of the view volume onto the projection plane, i.e., the view window or projection window is then mapped to the viewport for display. Only those objects within the view volume will appear in the display viewport; all others are clipped from display. The shape of the view volume varies according to the type of projection though the size is limited by suitably choosing front plane (or near plane) and back plane (or far plane). For a orthographic parallel projection the view volume is a rectangular parallelopiped whereas for perspective projection the view volume is a truncated pyramid or frustum. Refer Figure 6.27.

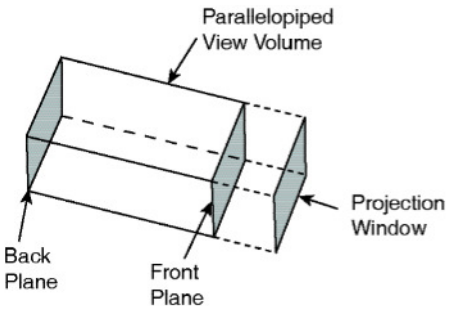


Figure 6.27: *Parallel Projection*

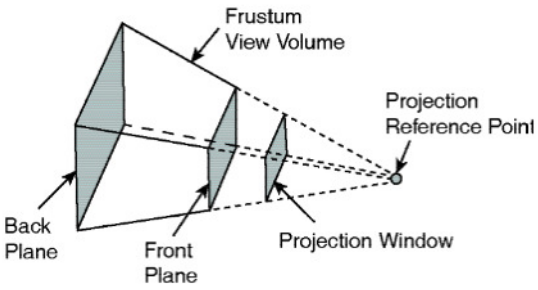


Figure 6.28: *Perspective Projection*

6.8 SUMMARY

In this unit we have learnt how computer graphics can be matured to implement models and effects closer to reality using 3D geometric and viewing transformations. Unlike two-dimensional viewing, 3D viewing requires clipping of the object against a 3D view volume followed by projection and finally mapping to a viewport for display. 3D geometric transformations, projection transformations and 3D clipping are interlinked as some cases of parallel or perspective transformation entail 3D rotations about coordinate axes. It is also illustrated in the projection mathematics how homogeneous coordinates can be effectively used to form composite transformation matrices arising from 3D geometric or viewing operations.

6.9 ANSWERS TO 'CHECK YOUR PROGRESS'

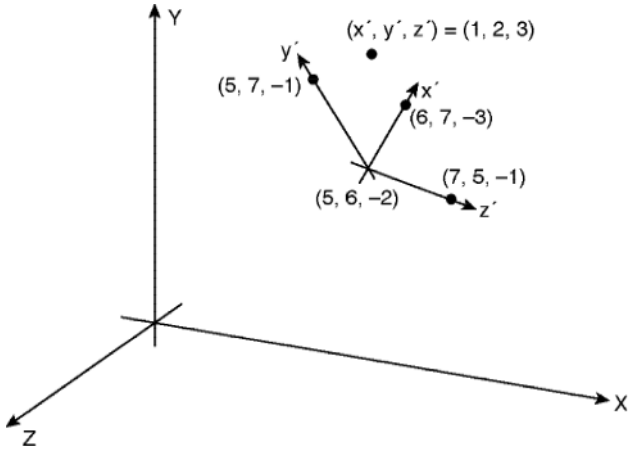
1. (c)
2. (b)
3. (c)
4. (a)
5. (a)

6.10 EXERCISES AND QUESTIONS

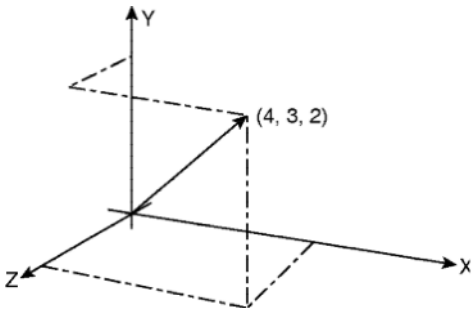
1. Specify suitable rotations (in terms of angle and axis of rotation) that you may apply so as to coincide the vector joining the origin and point (1,1,1) with Z axis.
2. Write necessary steps in order to describe a method of reflection of a three-dimensional figure about an arbitrary plane in terms of matrix operations.
3. Write 3D transformation matrix to find reflection of a point P(100,200,300) about plane z = 0.

4. Show how to use a three-dimensional matrix to rotate the unit cube [vertices (0,0,0), (1,0,0), (1,1,0), (0,1,0), (0,1,1), (1,1,1), (1,0,1), (0,0,1)] about the axis defined by vector [1 1 1].
5. Derive the transformation matrix (in homogeneous coordinate) to rotate a 3D object by an angle θ about an arbitrary line parallel to but not coincident with Z axis.
4. Determine 3D transformation matrices to scale the line AB in Z direction by 3.5 by keeping point A fixed. Then rotate this line by 45° anticlockwise about X axis. Given A (10, 15, 20) and B (45, 60, 30).
7. What is the sequence of transformations you would use to transform points in X'–Y′–Z′ into points in X-Y-Z? Write the matrices you would use, then compose them into one matrix and apply that matrix to transform the point $(x', y', z') = (1, 2, 3)$, into a point in (x, y, z).

All coordinates are in (x, y, z) except for the point at (1, 2, 3)



8. Show how you would construct the rotation matrix to rotate this vector down to the X axis. Describe each step. Show the final matrix, and test it by applying it to point (4, 3, 2).



9. Derive the necessary transformation matrix in 3D using homogeneous coordinate system to scale by a factor ***s*** w.r.t. the origin along a line making equal angles with all three axes.
10. What is oblique projection? Give some examples of oblique projection.
11. Compare parallel and perspective projections with reference to practical use only.
12. Given the perspective transformation matrix, show how the vanishing points may be located?

NOTES

NOTES

13. What is the physical significance of the vanishing point?
14. Why is vanishing point a direction?
15. For a 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).
16. An object is being viewed from the point (50,0,0). Obtain the transformation matrix to get projection of a point $P(x, y, z)$ on the YZ plane. Obtain the transformation matrix if the projection plane is now $x + 10 = 0$.
17. Consider a 3D coordinate system where the Y axis is vertical and Z axis pointing towards the viewer. A line A(10,-10,10) B(10,-10,0) is viewed from the point P(0,0,20). Find where the points A and B would be projected on the XY screen?
18. In a 3D coordinate system a box is placed at the origin such that its three edges are touching X,Y and Z axes. Describe the transformation matrix needed to show the side view of the box on the XY screen.
19. A rectangular field is described in 3D coordinate system a follows:
    A(-20,-20,0) B(20,-20,0)
    C(20,-20,-40) D(-20,-20,-40)
    Where the Y axis represents the vertical axis and Z axis is towards the viewer. A person is located at P(0,0,20) and is looking at the field. Obtain the perspective view generated on the XY plane.
20. A unit cube is projected onto the XY plane under perspective projection with centre of projection being (0, 0, –10). Draw the projected image of the cube. Determine the vanishing point for this transformation.
21. A unit cube is projected onto the XY plane under the parallel projection whose direction of projection is (1, 1, 1). Obtain the projection of the cube and draw it.
22. Obtain isometric projection matrix in order to obtain isometric projection of a point with coordinates (1, 1, 1).
23. Explain with the aid of a clearly labeled diagram what is meant by the term 'graphics pipeline'. or 'visualisation pipeline', indicating the coordinate systems and transformations at different stages to display two-dimensional shapes and patterns. Amend the diagram so as to describe what additional components would be required in order to display three-dimensional objects efficiently.
24. Define:
    (a) modeling coordinates
    (b) world coordinates
    (c) viewport
    (d) normalised device coordinates
    What is the use of Normalised Device Coordinates?
25. In Computer Graphics, why is a distinction made between modeling and viewing?
26. Describe how a 3D object is presented on the screen using perspective projection. Take a simple object for illustration.
27. Briefly explain all the viewing parameters while displaying a 3D object on a 2D screen.
28. Explain the terms: Projection plane, View plane, Coordinates and View volume with reference to 3D graphics. State and explain the anomalies of perspective projection.

29. A cube with sides of length 2 is placed so that a corner lies on the origin and three mutually perpendicular edges from this corner lie on the three positive coordinate axes. Now do the following:
    (i) Translate the cube along the XY plane so that the cube face is centred on the origin;
    (ii) Perform three-point perspective projection on the translated cube on the $z = 0$ plane with centres of projections $x = -10$, $y = -10$ and $z = 10$ on the respective coordinate axes. Draw the projected cube.
30. A cube has its vertices located at A(0,0,1), B(10,0,10), C(10,10,10), D(0,10,10), E(0,0,0), F(10,0,0), G(10,10,0), H(0,10,0). The Y axis is vertical and positive z axis is oriented towards the viewer. The cube is being viewed from the point (0,20,80). Work out the perspective view of the cube on the XY plane.

NOTES

6.11 FURTHER READING

1. Hearn, Donal and M. Pauline Baker, *Computer Graphics*.
2. Rogers, David F., *Procedural Elements For Computer Graphics*.
3. Foley, vanDam, Feiner, Hughes, *Computer Graphics Principles & Practice*.
4. Mukhopadhyay A. and A. Chattopadhyay, *Introduction to Computer Graphics and Multimedia*.

Appendix: C/C++ and VB Programs

NOTES

Midpoint Ellipse (VB)

```
Public Sub DrawEllipse(Rx As Single, Ry As Single, Xc As
Single, Yc As Single)
 Dim X As Single, Y As Single, p As Double
 Dim Ry2 As Double, Rx2 As Double
 Dim TwoRx2 As Double, TwoRy2 As Double
 Dim px As Double, py As Double

 Rx2 = (CDbl(Rx)) * (CDbl(Rx))

 Ry2 = (CDbl(Ry)) * (CDbl(Ry))
 TwoRx2 = 2 * Rx2
 TwoRy2 = 2 * Ry2
 X = 0
 Y = CSng(Ry)
 Call EllipsePlotPoints(Xc, Yc, X, Y)
 p = Ry2 - Rx2 * Ry + (0.25 * Rx2)
 px = 0
 py = TwoRx2 * Y
 While (px < py)
  X = X + 1
  px = px + TwoRy2
  If (p >= 0) Then
   Y = Y - 1
   py = py - TwoRx2

  End If
  If (p < 0) Then
   p = p + Ry2 + px
  Else
   p = p + Ry2 + px - py
  End If
  Call EllipsePlotPoints(Xc, Yc, X, Y)
 Wend
 p = Ry2 * (X + 0.5) * (X + 0.5) + Rx2 * (Y - 1) * (Y - 1) -
Rx2 * Ry2
 While (Y > 0)
  Y = Y - 1
  py = py - TwoRx2
  If (p < 0) Then
   X = X + 1
   px = px + TwoRy2
  End If
  If (p >= 0) Then
   p = p + Rx2 - py
  Else
   p = p + Rx2 - py + px
  End If
```

```
 Call EllipsePlotPoints(Xc, Yc, X, Y)
 Wend
End Sub
```

NOTES

```
Private Sub EllipsePlotPoints(Xc As Single, Yc As Single, X As
Single, Y As Single)
 Canvas.PSet ((Xc + X), (Yc + Y))
 Canvas.PSet ((Xc + X), (Yc - Y))
 Canvas.PSet ((Xc - X), (Yc - Y))
 Canvas.PSet ((Xc - X), (Yc + Y))
End Sub
```

Midpoint Ellipse (C++)

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
int Round(float n);
void StartGraphics(void);

class Ellipse
{
private:
        int xc,yc,a,b,aa,bb,2aa,2bb,color;
        void PlotPoints(float x,float y);
public:
Ellipse(int a,int b)
{
aa=a*a; bb=b*b; 2aa=2*aa;
2bb=2*bb;
}
void GetParameters(void);
void DrawEllipse(void);
}                       //End of class Ellipse


int Round(float n)
{
return(floor(0.5+n));
}

void Ellipse::GetParameters(void)
{
cout<<endl<<"\t\t Enter Center_x:";
cin>>xc;
cout<<endl<<"\t\t Enter Center_y:";
cin>>yc;
cout<<endl<<"\t\t Enter Semimajor axis:";
cin>>a;
cout<<endl<<"\t\t Enter Semiminor axis:";
cin>>b;
cout<<endl<<"\t\t Enter Color:";
cin>>color;
}
```

```
void StartGraphics(void)
{
int gd=DETECT,gm;
initgraph(&gd,&gm," ");
}

void Ellipse::PlotPoints(float x,float y)
{
putpixel(Round(xc+x),Round(yc+y),color);
putpixel(Round(xc+x),Round(yc-y),color);
putpixel(Round(xc-x),Round(yc-y),color);
putpixel(Round(xc-x),Round(yc+y),color);
}

void Ellipse::DrawEllipse(void)
{
float x=0;
float y=b;
float fx=0;
float fy=2aa*b;
float p=bb-aa*b+0.25*aa;

PlotPoints(x,y);

while (fx<fy)
{
        x++;
        fx+=2bb;
if(p>=0)
{
        y--;
        fy-=2aa;
}
If (p<0)
        p+=bb+fx;
else
        p+=bb+fx-fy;

PlotPoints(x,y);
}

p=bb*(x+0.5)*(x+0.5)+aa*(y-1)*(y-1)-aa*bb;

while(y>0)
{
        y--;
        fy-=2aa;
If(p<0)
{
        x++;
        fx+=2bb;
}
```

NOTES

NOTES

```
If(p>=0)
        p+=aa-fy;
else
        p+=aa-fy+fx;

PlotPoints(x,y);
}
}                       //End of DrawEllipse()

void main(void)
{
Ellipse E;              //Build object E of Ellipse
clrscr();               //Clear the screen
E.GetParameters();      //Get the parameters of E
clrscr();
StartGraphics();        //Switch to Graphics mode
E.DrawEllipse();        //Draw the ellipse E
getch();                //Wait
closegraph();           //Close Graphics mode
restorecrtmode();       //Get back to CRT mode
}                       //End of main
```

DDA Line (C)

```
void ddaline(int xs,int ys,int xe,int ye,int on2,int col)
{
  int i=1;
  float span,dx,dy,xi,yi;
  struct queue q;
  q.top=-1;
if(xs>=0 &&ys >=0&&xe>=0 && ye >=0)
{
  if ( abs(ye-ys) <= abs(xe-xs) )   //CHECKING FOR THE UNIT

                                    SAMPLING DIRECTION
                                    //BY FINDING THE LARGER OF
ΔX AND ΔY
 span= abs(xe-xs);
  else
     span=abs(ye-ys) ;
  dx=(xe-xs)/span;                  // SET THE SAMPLING INTER
                                      VAL FOR BOTH
  dy=(ye-ys)/span;                  // X,Y DIRECTION.
  xi=xs;
  yi=ys;

  while(i<=span)                    // TILL THE END POINT IS
                                      REACHED
  {
    if (on2==1)
   q.c[++q.top]=getpixel(xi,yi);
    if (on2==0)
     col=q.c[++q.top];                 // CALCULATE INTERMEDI
```

```
                                        ATE POINT
   Putpixel(round(xi),round(yi),col);   // COORDINATE BY
                                        INCREAMENTING
    xi=xi+dx;                           // METHOD. AFTER ROUND
                                            ING OF THE
    yi=yi+dy;                           // VALUE, SO OBTAINED
                                        SET THE
    i++ ;                               // CORRESPONDING PIXEL
                                            WITH DESIRED
  }
  }                                     // INTENSITY
 }

 int round(float  x)

{  int ix;
 ix=(int)x;
if (x-ix<=.5)
     return ix;
 else
   return (ix+1);
 }
```

General Bresenhams Line (C)

```
void brasline(int xs,int ys,int xe,int ye,int on1,int col)
{
      int dx,dy,x,y,s1,s2,temp,swap,p,n;
      struct queue q;
 q.top=-1;
  if (xs>=0 && ys>=0 && xe>=0&&ye >= 0)
  {
      x=xs;
      y=ys;
      dx=abs(xe-xs);
      dy=abs(ye-ys);
      s1=(xe-xs)/dx;
      s2=(ye-ys)/dy;

    if (dy>dx)
        {
      temp=dx;
      dx=dy;
      dy=temp;
      swap=1;
       }
    else
      swap=0;

   p=2*dy-dx;
   if(on1==1)

    q.c[++q.top]=getpixel(x,y);
```

NOTES

NOTES

```
if(on1==0)
  col=q.c[++q.top];
 putpixel(x,y,col);
  n=1;

 while(n<dx)
 {
   if (p>0)
    {
    x=x+s1;
    y=y+s2;
    p=p+2*dy-2*dx;
     }
   else
    {
     if (swap==0)
     x=x+s1;
     else
    y=y+s2;
    p=p+2*dy;
     }
      if(on1==1)
       q.c[++q.top]=getpixel(x,y);
     if(on1==0)
      col=q.c[++q.top];
     putpixel(x,y,col);
     n++;
   }
   }
}
```

Bresenham Circle (C)

```
Void brescircle(int x_centre,int y_centre,float radius,int
colour)
#define colour
{
      int D,delta,x,y;
      gotoxy(3,4);
      printf("enter the coordinates for the centre of the
circle :");
      scanf("%d %d",&x_centre,&y_centre);
      gotoxy(3,7);
      printf("enter the radius of the circle :");
      scanf("%d",&radius);
      clearviewport();
      D=2*(1-radius);
      x=0;
      y=radius;

      //    settextjustify(CENTER_TEXT,CENTER_TEXT);
      //    putpixel(x_centre,y_centre,colour);
      //    setcolor(GREEN);
      //    outtextxy(x_centre,y_centre+ADJUST,"centre");
```

```
//    setcolor(colour);

while(y>=x)
{
      putpixel((x_centre + x),(y_centre + y),colour);
      putpixel((x_centre - x),(y_centre + y),colour);
      putpixel((x_centre + x),(y_centre - y),colour);
      putpixel((x_centre - x),(y_centre - y),colour);
      putpixel((x_centre + y),(y_centre + x),colour);
      putpixel((x_centre - y),(y_centre + x),colour);
      putpixel((x_centre + y),(y_centre - x),colour);
      putpixel((x_centre - y),(y_centre - x),colour);

      if(D<0)
      {
            delta=2*D+2*y-1;
            if(delta<=0)
            {
                  x++;
                  D=D+2*x+1;
            }
            else
            {
                  x++;
                  y--;
                  D=D+2*x-2*y+2;
            }
      }
      else if(D>0)
      {
            delta=2*x-2*D+1;
            if(delta<0)
            {
                  y--;
                  D=D-2*y+1;
            }
            else
            {
                  x++;
                  y--;
                  D=D+2*x-2*y+2;
            }
      }
      else
      {
            x++;
            y--;
            D=D+2*x-2*y+2;
      }
}
}
```

NOTES

NOTES

DDA Line (VB)

```
Function lin(ByVal x11, ByVal y11, ByVal x12, ByVal y12)
Dim X, Y, X_ST, Y_ST, x_end, y_end As Double
X = x11
Y = y11
X_ST = x11
Y_ST = y11
x_end = x12
y_end = y12
If Abs(y_end - Y_ST) <= Abs(x_end - X_ST) Then
      span = Abs(x_end - X_ST)
  Else
      span = Abs(y_end - Y_ST)
    End If
 Del_x = (x_end - X_ST) / span
 Del_y = (y_end - Y_ST) / span
 I = 1
 While (i <= span)
  Form1.Picture1.PSet (X, Y), vbRed
  X = X + del_x
  Y = Y + del_y
  i = i + 1
  Wend
End Function
```

Bresenham Line (VB)

```
 Public Sub BresLine(ByVal Xs As Integer, ByVal Ys As
Integer, ByVal
 Xe As Integer, ByVal Ye As Integer)
 Dim s1 As Integer, s2 As Integer, Dx As Integer, Dy As Integer
 Dim tdx As Integer, tdy As Integer, Temp As Integer
 Dim n As Integer, X As Integer, Y As Integer, p As Integer
 Dim swap As Boolean
 Tdx = Xe - Xs
 Tdy = Ye - Ys
 If tdx >= 0 Then
  s1 = 1
 Else
  s1 = -1
 End If
 If tdy >= 0 Then
  s2 = 1
 Else
  s2 = -1
 End If
 Dx = Abs(tdx)
 Dy = Abs(tdy)
 Canvas.PSet (Xs, Ys)
 N = 1
 X = Xs
 Y = Ys
 P = 2 * Dy - Dx
 Swap = False
 If Dy > Dx Then
```

```
 Temp = Dx
 Dx = Dy
 Dy = Temp
 swap = True
End If
While (n <= Dx)
 If p >= 0 Then
  X = X + s1
  Y = Y + s2
  p = p + 2 * (Dy - Dx)
 Else
  If swap Then
   Y = Y + s2
  Else
   X = X + s1
  End If
  p = p + 2 * Dy
 End If
 n = n + 1
 Canvas.PSet (X, Y)
Wend
End Sub
```

Bresenham Circle (VB)

```
Private Sub DrawCircle(Xc As Single, Yc As Single, Radius As
Single)
 Dim X As Single, Y As Single
 Dim DelD As Single, DelHD As Single, DelVD As Single
 X = 0
 Y = Radius
 DelD = 2 * (1 - Radius)
 While (Y > X)
  Call CirclePlotPoints(Xc, Yc, X, Y)
  If (DelD < 0) Then
   DelHD = 2 * DelD + 2 * Y - 1
   If (DelHD < 0) Then
    X = X + 1
    Y = Y
    DelD = DelD + 2 * X + 1
   Else
     X = X + 1
     Y = Y - 1
     DelD = DelD + 2 * X - 2 * Y + 2
   End If
  Else
   If (DelD > 0) Then
    DelVD = 2 * X + 1 - 2 * DelD
    If (DelVD < 0) Then
     X = X
     Y = Y - 1
     DelD = DelD - 2 * Y + 1
    Else
     X = X + 1
```

NOTES

NOTES

```
      Y = Y - 1
      DelD = DelD + 2 * X - 2 * Y + 2
     End If
    Else
     X = X + 1
     Y = Y - 1
     DelD = DelD + 2 * X - 2 * Y + 2
    End If
   End If
 Wend
End Sub

Private Sub CirclePlotPoints(Xc As Single, Yc As Single, X As
Single, Y As Single)
 Canvas.PSet ((Xc + X), (Yc + Y))
 Canvas.PSet ((Xc + Y), (Yc + X))
 Canvas.PSet ((Xc + Y), (Yc - X))
 Canvas.PSet ((Xc + X), (Yc - Y))
 Canvas.PSet ((Xc - X), (Yc - Y))
 Canvas.PSet ((Xc - Y), (Yc - X))
 Canvas.PSet ((Xc - Y), (Yc + X))
 Canvas.PSet ((Xc - X), (Yc + Y))
End Sub
```

Trigonometric Circle (VB)

```
Function CIRC(ByVal X_C As Single, ByVal Y_C As Single, ByVal R
As Single)
Dim X, Y, THETA, X_TEMP As Single
X = 0
Y = R
THETA = 1 / R

While (Y > X)
Form1.Picture1.PSet ((X_C + X), (Y_C + Y)), vbRed
Form1.Picture1.PSet ((X_C + Y), (Y_C + X)), vbRed
Form1.Picture1.PSet ((X_C + Y), (Y_C - X)), vbRed
Form1.Picture1.PSet ((X_C + X), (Y_C - Y)), vbRed
Form1.Picture1.PSet ((X_C - X), (Y_C - Y)), vbRed
Form1.Picture1.PSet ((X_C - Y), (Y_C - X)), vbRed
Form1.Picture1.PSet ((X_C - Y), (Y_C + X)), vbRed
Form1.Picture1.PSet ((X_C - X), (Y_C + Y)), vbRed
X_TEMP = X
X = X * (Cos(THETA)) - Y * (Sin(THETA))
Y = Y * (Cos(THETA)) + X_TEMP * (Sin(THETA))
Wend
End Function
```

Trigonometric Circle (C)

```
void tricircle(int xc, int yc, int r,int col)
{
   float thetha,x=0,y,xtemp;
    thetha = (float) -1/r;
    y=r;
```

```
   while(y>x)

    {
    putpixel(round(x+xc),round(y+yc),col);
    putpixel(round(y+xc),round(x+yc),col);
    putpixel(round(y+xc),round(-x+yc),col);
    putpixel(round(x+xc),round(-y+yc),col);
    putpixel(round(-x+xc),round(-y+yc),col);
    putpixel(round(-y+xc),round(-x+yc),col);
    putpixel(round(-y+xc),round(x+yc),col);
    putpixel(round(-x+xc),round(y+yc),col);
      xtemp=x;
      x=x*cos(thetha)-y*sin(thetha);
      y=y*cos(thetha)+xtemp*sin(thetha);

      }

   }
```

Rectangle Using Bresenham Line (VB)

```
Private Sub Rectangle(ByVal topx As Integer,ByVal topy As
Integer,ByVal bottx As Integer,ByVal botty As Integer)
  Call BresLine(topx, topy, bottx, topy)
  Call BresLine(topx, botty, bottx, botty)
  Call BresLine(topx, topy, topx, botty)
  Call BresLine(bottx, topy, bottx, botty)
 End Sub
```

Polygon Using Mouse (C)

```
void Polygon(int color)
  {
       int button,x,y:
       float x1,y1,x2,y2,prevx2,prevy2;
       do
        {
             GetMousePosition(&button,&x,&y);
        }while(!(button==1 && x>=81 && x<=634 && y>=22 &&
y<=459)
);
       x1=prevx2=x-81;
       y1=prevy2=y-22;
       do
        {
             GetMousePosition (&button,&x,&y);
             if (button==1)
               {
                        x2=x-81;
                        y2=y-22;
                        HideMousePtr();
```

NOTES

NOTES

```
                        Line(prevx2,prevy2,x2,y2,color);

                        ShowMousePtr();   // As Restoration Is
Not Needed
                        prevx2=x2;
                        prevy2=y2;

                {
       }while(!(button==2));

      HideMousePtr();
      Line(x2,y2,x1,y1,color);
      ShowMousePtr();
```

Translation (C)

```
Void translation(float coordinate[3][3] ,int dx,int dy )
{  float point[3][3];          /*transforming the point*/
  unitmat(point);

  point[0][2]=1.0*dx;
  point[1][2]=1.0*dy;

  matmul(point,coordinate);
 }

 void initpoint(float a[3][3],int x,int y) /* initialisation
the pointmatrix */
 {
  a[0][0]=x;
 a[1][0]=y;
 a[1][1]=0;
 a[2][2]=0;
 a[2][0]=1;
  }
```

Rotation about (x_p, y_p) (C)

```
void rotate (float coordinate[3][3], double theta, int xp=0,
int yp=0 )
{
      float point [3][3] ;
      unitmat (point);
            point [0][0] = cos (theta);
      point [0][1] = -sin (theta);
      point [1][1] = cos (theta);
      point [1][0] = sin (theta);
      point [0][2] = ((1 - cos (theta)) * xp + yp *
sin(theta));
      point [1][2] = ((1 - cos (theta)) * yp - xp *
sin(theta));
      matmul (point, coordinate);

}
```

Scaling about (x_f, y_f) (C)

```
void scaling(float coordinate[3][3],float sx,float sy, int
xf=0,int yf=0)
{
      float point [3][3] ;
      unitmat (point);
      point[0][0]=sx;
      point[1][1]=sy;
      point[0][2]=(1-sx)*xf;
      point[1][2]=(1-sy)*yf;
      matmul(point,coordinate);
  }
```

Inverse Scaling (C)

```
 Void inv_scaling(float coordinate[3][3],float sx,float sy,int
xf=0,int yf=0)
 {    sx=1/sx;
 sy=1/sy;
 scaling(coordinate,sx,sy,xf,yf);
 }
```

Inverse Rotation (C)

```
Void inv_rotate(float coordinate[3][3],double theta,int
xp=0,int yp=0)
{
    theta=-theta;
  rotate(coordinate,theta,xp,yp);
}
```

Inverse Translation (C)

```
Void inv_translation(float coordinate[3][3],int dx ,int dy)
{
   dx=-dx;
   dy=-dy;
  translation(coordinate,dx,dy);
   }
```

Reflection about X axis (C)

```
Void xaxis_reflection(float coordinate[3][3])
{
float  point[3][3];
unitmat(point);
point[1][1]=-1;
//point[0][0]=-1;
matmul(point,coordinate);

}
```

Reflection about Y axis (C)

```
void yaxis_reflection(float coordinate[3][3])
{
float  point[3][3];
unitmat(point);
```

NOTES

NOTES

```
point[0][0]=-1;
//point[1][1]=-1;
matmul(point,coordinate);

}
```

Function Unitmat ( ) and Matmul ( ) (C)

```
void unitmat(float a[3][3])    /*creating unit matrix*/
{
  int i,j;
  for(i=0;i<3;i++)
   {
    for(j=0;j<3;j++)
     {
       if(i==j)
         a[i][j]=1;
       else
        a[i][j]=0;
     }
     }
     }

 void matmul(float  a[3][3],float b[3][3]) /*matrix multiplica-
tion and result is returning in matrix b*/
 {  float  temp[3][3];
   int i,j;

   for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        temp[i][j]=a[i][0]*b[0][j] +a[i][1]*b[1][j]
+a[i][2]*b[2][j];

        }

        }

       for(i=0;i<3;i++)
    {
      for(j=0;j<3;j++)
      {
        b[i][j]=temp[i][j];

         }

         }
     }
```

Translation (VB)

```
Public Function Translate(ObjectPoint() As Single, Tx As
Single, Ty As Single, Tz As Single) As Single()
 Dim NewPoint(4) As Single
```

```
 NewPoint(0) = ObjectPoint(0) + Tx
 NewPoint(1) = ObjectPoint(1) + Ty
 NewPoint(2) = ObjectPoint(2) + Tz
 NewPoint(3) = ObjectPoint(3)
 Translate = NewPoint()
End Function
```

NOTES

Inverse Translation (VB)

```
Public Function InverseTranslate(ObjectPoint() As Single, Tx
As Single, Ty As Single, Tz As Single) As Single()
 Dim NewPoint(4) As Single
 NewPoint(0) = ObjectPoint(0) - Tx
 NewPoint(1) = ObjectPoint(1) - Ty
 NewPoint(2) = ObjectPoint(2) - Tz
 NewPoint(3) = ObjectPoint(3)
 InverseTranslate = NewPoint()
End Function
```

Rotation About Origin (VB)

```
Public Function RotateAboutOrigin(ObjectPoint() As Single,
theta As Single)
As
Single()
 Dim NewPoint(4) As Single, ThetaDegree As Single
 ThetaDegree = (3.142 * theta) / 180
 NewPoint(0) = ObjectPoint(0) * Cos(ThetaDegree) -
ObjectPoint(1) *
Sin(ThetaDegree)
 NewPoint(1) = ObjectPoint(0) * Sin(ThetaDegree) +
ObjectPoint(1) *
Cos(ThetaDegree)
 NewPoint(2) = ObjectPoint(2)
 NewPoint(3) = ObjectPoint(3)
 RotateAboutOrigin = NewPoint()
End Function
```

Inverse Rotation about Origin (VB)

```
Public Function InverseRotateAboutOrigin(ObjectPoint() As
Single, theta As Single) As Single()
 Dim NewPoint(4) As Single, ThetaDegree As Single
 ThetaDegree = (3.142 * theta) / 180
 NewPoint(0) = ObjectPoint(0) * Cos(-ThetaDegree) -
ObjectPoint(1) *
Sin(-ThetaDegree)
 NewPoint(1) = ObjectPoint(0) * Sin(-ThetaDegree) +
ObjectPoint(1) *
Cos(-ThetaDegree)
 NewPoint(2) = ObjectPoint(2)
 NewPoint(3) = ObjectPoint(3)
 InverseRotateAboutOrigin = NewPoint()
End Function
```

NOTES

Rotation about Arbitrary Pivot Point (VB)

```
Public Function RotateAboutPivotPoint(ObjectPoint() As Single,
theta As Single, PivotPoint() As Single) As Single()
 Dim NewPoint(4) As Single, ThetaDegree As Single
 ThetaDegree = (3.142 * theta) / 180
 NewPoint(0) = ObjectPoint(0) * Cos(ThetaDegree) -
ObjectPoint(1) *
Sin(ThetaDegree) + ((1 - Cos(ThetaDegree)) * PivotPoint(0) +
PivotPoint(1) *
Sin(ThetaDegree))
 NewPoint(1) = ObjectPoint(0) * Sin(ThetaDegree) +
ObjectPoint(1) *
Cos(ThetaDegree) + (-PivotPoint(0) * Sin(ThetaDegree) + (1 -
Cos(ThetaDegree)) *
PivotPoint(1))
 NewPoint(2) = ObjectPoint(2)
 NewPoint(3) = ObjectPoint(3)
 RotateAboutPivotPoint = NewPoint()
End Function
```

Scaling about Origin (VB)

```
Public Function ScaleAboutOrigin(ObjectPoint() As Single, sx As
Single, sy As Single, Sz As Single) As Single()
 Dim NewPoint(4) As Single
 NewPoint(0) = ObjectPoint(0) * sx
 NewPoint(1) = ObjectPoint(1) * sy
 NewPoint(2) = ObjectPoint(2) * Sz
 NewPoint(3) = ObjectPoint(3)
 ScaleAboutOrigin = NewPoint()
End Function
```

Inverse Scaling about Origin (VB)

```
Public Function InverseScaleAboutOrigin(ObjectPoint() As
Single, sx As Single, Sy As Single, Sz As Single) As Single()
 Dim NewPoint(4) As Single
 NewPoint(0) = ObjectPoint(0) * (1 / sx)
 NewPoint(1) = ObjectPoint(1) * (1 / sy)
 NewPoint(2) = ObjectPoint(2) * (1 / Sz)
 NewPoint(3) = ObjectPoint(3)
 InverseScaleAboutOrigin = NewPoint()
End Function
```

Scaling about Arbitrary Point (VB)

```
Public Function ScaleAboutPivotPoint(ObjectPoint() As Single,
sx As Single, sy As Single, Sz As Single, PivotPoint() As
Single) As Single()
 Dim NewPoint(4) As Single
 NewPoint(0) = ObjectPoint(0) * sx + PivotPoint(0) * (1 - sx) *
ObjectPoint(3)
 NewPoint(1) = ObjectPoint(1) * sy + PivotPoint(1) * (1 - sy) *
ObjectPoint(3)
 NewPoint(2) = ObjectPoint(2) * Sz + PivotPoint(2) * (1 - Sz) *
ObjectPoint(2)
```

```
 NewPoint(3) = ObjectPoint(3)
 ScaleAboutPivotPoint = NewPoint()
End Function
```

Reflection about X Axis (VB)

```
Public Function ReflectionAboutXAxis(ObjectPoint() As Single)
As Single()
 Dim NewPoint(4) As Single
 NewPoint(0) = ObjectPoint(0)
 NewPoint(1) = -ObjectPoint(1)
 NewPoint(2) = ObjectPoint(2)
 NewPoint(3) = ObjectPoint(3)
 ReflectionAboutXAxis = NewPoint()
End Function
```

Reflection about Arbitrary Axis (VB)

```
Public Function ReflectionAboutAnyAxis(ObjectPoint() As
Single, StartAxis() As Single, EndAxis() As Single) As
Single()
 Dim NewPoint1() As Single, NewPoint2() As Single, NewPoint3()
As Single,
NewPoint4() As Single, NewPoint() As Single
 ReDim NewPoint1(4)
 ReDim NewPoint2(4)
 ReDim NewPoint3(4)
 ReDim NewPoint4(4)
 ReDim NewPoint(4)
 Dim C As Single
 Dim theta As Double
 Dim ThetaDegree As Single
 C = StartAxis(1) - ((EndAxis(1) - StartAxis(1)) / (EndAxis(0)
- StartAxis(0)))
* StartAxis(0)
 theta = Atn(CDbl(((EndAxis(1) - StartAxis(1)) / (EndAxis(0) -
StartAxis(0)))))
 ThetaDegree = CSng((theta * 180) / 3.142)
 NewPoint1() = Translate(ObjectPoint(), 0, -C, 0)
 NewPoint2() = RotateAboutOrigin(NewPoint1(), ThetaDegree)
 NewPoint3() = ReflectionAboutXAxis(NewPoint2())
 NewPoint4() = InverseRotateAboutOrigin(NewPoint3(),
ThetaDegree)
 NewPoint() = InverseTranslate(NewPoint4(), 0, -C, 0)

 ReflectionAboutAnyAxis = NewPoint()
End Function
```

NOTES

PUNJAB TECHNICAL UNIVERSITY

LADOWALI ROAD, JALANDHAR

INTERNAL ASSIGNMENT

TOTAL MARKS: 25

NOTE: Attempt any 5 questions
All questions carry 5 Marks.

Q. 1. What are the various means of feeding input to graphics programs?

Q. 2. What are the major categories of hardcopy devices? How is the printing quality quantitatively measured? Is it the same for all printers?

Q. 3. Write down the difference between the C functions cleardevice ( ) and clrscr ( ).

Q. 4. Explain the algorithm for scan-converting convex polygons. It is not necessary to write the code, a precise narrative description is sufficient.

Q. 5. Show how shear transformations may be expressed in terms of rotations and scales. Show how rotations can be expressed in terms of shears and scales. What scaling operations can be expressed as shears?

Q. 6. Consider a line P1P2 from (x_1, y_1, z_1) to (x_2, y_2, z_2) in aright handed coordinate system. Rotate this line such that P1 lies on the origin and P_2 on positive Z axis. Find the transformed line.

Q. 7. 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 center of projection is (10, 0, 0).

Q. 8. Design an interface where there is a provision for inputs from the keyboard, the scanner, the mouse and the webcam.

Q. 9. What are the major adverse side-effects of scan conversion? What methods are adopted to remove those effects?

Q.10. Indicate to what extent the principles underlying the Bresenham's line drawing algorithm can be utilized in drawing an arc of a curve defined by the function $f(x, y) = 0$.

PUNJAB TECHNICAL UNIVERSITY

LADOWALI ROAD, JALANDHAR

ASSIGNMENT SHEET

(To be attached with each Assignment)

Full Name of Student:_____

(First Name) (Last Name)

Registration Number:

Course:_____ Sem.:_____ Subject of Assignment:_____

Date of Submission of Assignment:

(Question Response Record-To be completed by student)

S.No.	Question Number Responded	Pages ____ - ____ of Assignment	Marks
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			

Total Marks:_____/25

Remarks by Evaluator:_____

_____

Note: Please ensure that your Correct Registration Number is mentioned on the Assignment Sheet.

Name of the Evaluator

Signature of the Student Signature of the Evaluator

Date:_____ Date:_____