

SUBJECT CODE : 210244

Strictly as per Revised Syllabus of
SAVITRIBAI PHULE PUNE UNIVERSITY
Choice Based Credit System (CBCS)
S.E. (Computer) Semester - I

COMPUTER GRAPHICS

(*For IN SEM Exam*)

Atul P. Godse

M.S. Software Systems (BITS Pilani)

B.E. Industrial Electronics

Formerly Lecturer in Department of Electronics Engg.

Vishwakarma Institute of Technology

Pune

Dr. Deepali A. Godse

M.E., Ph.D. (Computer Engg.)

Head of Information Technology Department,

Bharati Vidyapeeth's College of Engineering for Women,

Pune

Dr. Rajesh Prasad

Ph.D. (Computer Science & Engg.)

Professor (Department of Computer Engineering),

Sinhgad Institute of Technology & Science,

Narhe, Pune.



COMPUTER GRAPHICS

(For IN SEM Exam)

Subject Code : 210244

S.E. (Computer) Semester - I

First Edition : July 2020

© Copyright with A. P. Godse and Dr. D. A. Godse

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :



Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-81-946628-8-4



9 788194 662884

SPPU 19

PREFACE

The importance of **Computer Graphics** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Computer Graphics**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors

A.P. Godse

Dr. D. A. Godse

Dr. Rajesh Prasad

Dedicated to God

SYLLABUS

Computer Graphics - 210244

Credit Scheme	Examination Scheme and Marks :
03	Mid-Semester (TH) : 30 Marks

Unit I Graphics Primitives and Scan Conversion Algorithms

Introduction, graphics primitives - pixel, resolution, aspect ratio, frame buffer. Display devices, applications of computer graphics.

Introduction to OpenGL - OpenGL architecture, primitives and attributes, simple modelling and rendering of two- and three-dimensional geometric objects, GLUT, interaction, events and call-backs picking. (**Simple Interaction with the Mouse and Keyboard**)

Scan conversion : Line drawing algorithms : Digital Differential Analyzer (DDA), Bresenham. Circle drawing algorithms : DDA, Bresenham, and Midpoint. (**Chapters - 1,2,3**)

Unit II Polygon, Windowing and Clipping

Polygons : Introduction to polygon, types : convex, concave and complex. Inside test.

Polygon Filling : flood fill, seed fill, scan line fill.

Windowing and Clipping : viewing transformations, 2-D clipping: Cohen- Sutherland algorithm line Clipping algorithm, Sutherland Hodgeman Polygon clipping algorithm, Weiler Atherton Polygon Clipping algorithm. (**Chapters - 4,5**)

TABLE OF CONTENTS

Unit - I

Chapter - 1 Introduction to Computer Graphics	(1 - 1) to (1 - 18)
1.1 Basic Concepts	1 - 2
1.1.1 Pixel	1 - 2
1.1.2 Rasterization and Scan Conversion	1 - 3
1.1.3 Other Elements of Computer Graphics	1 - 3
1.2 Display Devices	1 - 3
1.2.1 Cathode-Ray-Tubes.....	1 - 3
1.2.2 Vector Scan/Random Scan Display	1 - 5
1.2.3 Raster Scan Display and Frame Buffer	1 - 6
1.2.4 Colour CRT Monitors	1 - 9
1.2.5 Direct-View Storage Tubes.....	1 - 10
1.2.6 Flat Panel Displays	1 - 11
1.2.7 Plasma Panel Display	1 - 11
1.2.8 Liquid Crystal Monitors	1 - 12
1.2.9 Three-Dimensional Viewing Devices.....	1 - 14
1.2.10 Persistence, Resolution and Aspect Ratio	1 - 15
1.2.11 Applications of Large Screen Displays.....	1 - 16
1.3 Applications of Computer Graphics	1 - 17

Chapter - 2 Introduction to OpenGL	(2 - 1) to (2 - 46)
------------------------------------------------	----------------------------

2.1 OpenGL Architecture	2 - 2
2.1.1 Basic OpenGL Syntax	2 - 2
2.1.2 Related Libraries	2 - 3
2.1.3 Header Files.....	2 - 4
2.1.4 Display-Window Management using GLUT	2 - 4

2.1.5 Format of OpenGL Command	2 - 6
2.1.6 Vertex Function.....	2 - 6
2.1.6.1 Multiple Forms of Vertex Functions.	2 - 6
2.1.7 A Simple OpenGL Program.....	2 - 7
2.1.8 A Complete OpenGL Program	2 - 10
2.1.9 OpenGL Data Types	2 - 13
2.1.10 OpenGL Function Types	2 - 13
2.2 Primitives and Attributes	2 - 14
2.2.1 Basic OpenGL Primitives.....	2 - 15
2.2.1.1 GL_POINTS	2 - 16
2.2.1.2 GL_LINES	2 - 16
2.2.1.3 GL_LINE_STRIP	2 - 17
2.2.1.4 GL_LINE_LOOP	2 - 17
2.2.1.5 GL_TRIANGLES	2 - 18
2.2.1.6 GL_TRIANGLE_STRIP	2 - 18
2.2.1.7 GL_TRIANGLE_FAN	2 - 19
2.2.1.8 GL_QUADS	2 - 19
2.2.1.9 GL_QUAD_STRIP	2 - 20
2.2.1.10 GL_POLYGON	2 - 20
2.2.2 Attributes.....	2 - 21
2.2.2.1 Color Attribute	2 - 21
2.2.2.2 OpenGL Point Attributes	2 - 24
2.2.2.3 OpenGL Line Attributes	2 - 24
2.2.2.4 OpenGL Fill Attributes	2 - 25
2.2.2.5 OpenGL Charater Attributes	2 - 25
2.3 Simple Modelling and Rendering of 2D and 3D Geometric Objects.....	2 - 26
2.3.1 Primitive Functions Examples	2 - 27
2.3.2 Modelling a Colored Cube	2 - 31
2.4 Interaction	2 - 32
2.4.1 Events	2 - 32
2.4.2 Keyboard Callback Functions.....	2 - 33

2.4.3 Mouse Event Callback Functions.....	2 - 36
2.4.4 Window Event.....	2 - 41
2.5 Picking.....	2 - 42
2.5.1 Picking and Selection Mode.....	2 - 43
Program 2.3.1 : Draw the basic primitives using OpenGL	2 - 27
Program 2.3.2 : Write a program in OpenGL to display the following Fig. 2.3.1 on a raster display system. Assume suitable coordinates for the vertices.....	2 - 30
Program 2.4.1 : This program uses keyboard callback function. The keyboard callback function checks the keypress and accordingly display graphics primitive.....	2 - 34
Program 2.4.2 : This program displays rectangle on left mouse click and polygon on right mouse click.....	2 - 37
Program 2.4.3 : Create a polyline using mouse interaction using OpenGL.....	2 - 39

Chapter - 3 Scan Conversion	(3 - 1) to (3 - 42)
3.1 Introduction.....	3 - 2
3.1.1 Lines	3 - 2
3.1.2 Lines Segments	3 - 4
3.1.3 Vectors.	3 - 6
3.2 Line Drawing Algorithms	3 - 7
3.2.1 Qualities of Good Line Drawing Algorithm	3 - 7
3.2.2 Vector Generation/Digital Differential Analyzer (DDA) Algorithm	3 - 9
3.2.3 Bresenham's Line Algorithm	3 - 16
3.2.4 Generalized Bresenham's Line Drawing Algorithm	3 - 19
3.3 Antialiasing and Antialiasing Techniques	3 - 26
3.3.1 Supersampling Considering Zero Line Width	3 - 27
3.3.2 Supersampling Considering Finite Line Width	3 - 28
3.3.3 Supersampling with Pixel-Weighting Mask	3 - 29

3.3.4 Unweighted Area Sampling	3 - 30
3.3.5 Weighted Area Sampling	3 - 30
3.3.6 Filtering Techniques	3 - 31
3.3.7 Pixel Phasing	3 - 31
3.4 Basic Concepts in Circle Drawing	3 - 32
3.4.1 Polynomial Method	3 - 32
3.4.2 Trigonometric Method	3 - 33
3.5 Circle Drawing Algorithms	3 - 33
3.5.1 Vector Generation/DDA Circle Drawing Algorithm	3 - 33
3.5.2 Bresenham's Circle Drawing Algorithm	3 - 35
3.5.3 Midpoint Circle Algorithm	3 - 40

Unit - II

Chapter - 4 Polygons and Polygon Filling (4 - 1) to (4 - 14)

4.1 Introduction to Polygon	4 - 2
4.2 Types : Convex, Concave and Complex	4 - 2
4.3 Representation of Polygon	4 - 3
4.4 Inside Test	4 - 4
4.5 Polygon Filling Algorithms	4 - 6
4.5.1 Seed Fill	4 - 6
4.5.1.1 Boundary Fill Algorithm / Edge Fill Algorithm	4 - 6
4.5.1.2 Flood Fill Algorithm	4 - 8
4.5.2 Scan Line Algorithm	4 - 9

Chapter - 5 Windowing and Clipping (5 - 1) to (5 - 42)

5.1 Introduction	5 - 2
5.2 Viewing Transformations	5 - 2
5.2.1 Viewing Co-ordinate Reference Frame	5 - 4
5.2.2 Transformation to Normalized Co-ordinates	5 - 5
5.2.3 Window to Viewport Co-ordinate Transformation	5 - 6

5.3 2 D Clipping	5 - 14
5.3.1 Point Clipping	5 - 14
5.3.2 Line Clipping	5 - 15
5.4 Cohen-Sutherland Line Clipping Algorithm	5 - 15
5.5 Polygon Clipping	5 - 23
5.5.1 Sutherland - Hodgeman Polygon Clipping	5 - 24
5.5.2 Weiler-Atherton Algorithm	5 - 29
5.5.3 Liang-Barsky Polygon Clipping	5 - 30
5.6 Generalized Clipping	5 - 40
5.7 Interior and Exterior Clipping	5 - 40

UNIT - I

1

Introduction to Computer Graphics

Syllabus

Introduction, graphics primitives - pixel, resolution, aspect ratio, frame buffer. Display devices, applications of computer graphics.

Contents

1.1 Basic Concepts	Dec.-09, 11, May-13,	Marks 2
1.2 Display Devices	Dec.-06, 10, 11, 12, 15, May-05, 09, 13, 14, 15,	Marks 5
1.3 Applications of Computer Graphics	Dec.-14, May-16	Marks 6

1.1 Basic Concepts

SPPU : Dec.-09, 11, May-13

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

1.1.1 Pixel

- In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called **pixels**.
- The pixel is the smallest addressable screen element.
- It is the smallest piece of the display screen which we can control.
- The control is achieved by **setting the intensity and colour** of the pixel which compose the screen. This is illustrated in Fig. 1.1.1.
- Each pixel on the graphics display does not represent mathematical point. Rather, it represents a region which theoretically can contain an infinite number of points.
- For example, if we want to display point P_1 whose co-ordinates are $(4.2, 3.8)$ and point P_2 whose co-ordinates are $(4.8, 3.1)$ then P_1 and P_2 are represented by only one pixel $(4, 3)$, as shown in the Fig. 1.1.2.
- In general, a point is represented by the integer part of x and integer part of y , i.e., pixel $(\text{int}(x), \text{int}(y))$.

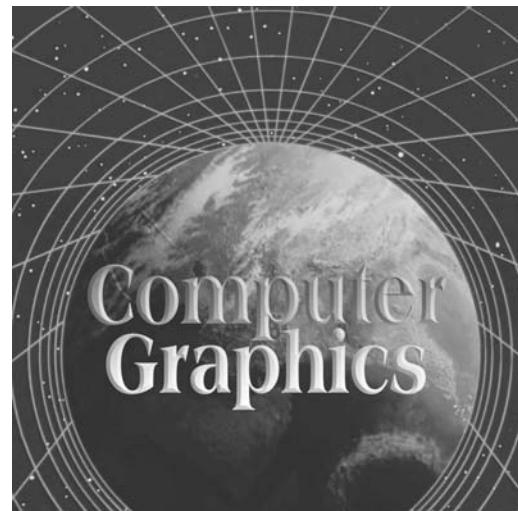


Fig. 1.1.1 Representation of picture

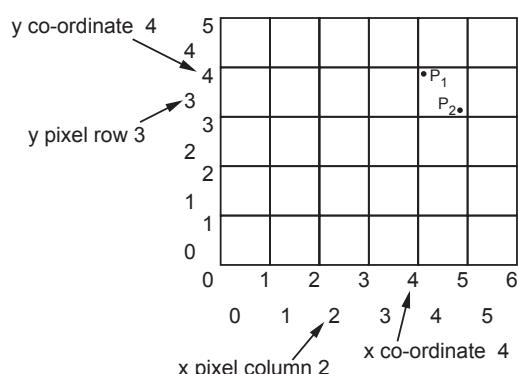


Fig. 1.1.2 Pixel display area of 6×5

1.1.2 Rasterization and Scan Conversion

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

1.1.3 Other Elements of Computer Graphics

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

Review Questions

1. What is pixel ?
2. Define rasterization.
3. Define scan conversion.

SPPU : Dec.-09,11, May-13, Marks 2

1.2 Display Devices

SPPU : Dec.-06, 10, 11, 12, 15, May-05, 09, 13, 14, 15

The display devices are also known as **output devices**. The most commonly used output device in a graphics system is a video monitor. The operation of most video monitors is based on the standard **Cathode-Ray-Tube** (CRT) design. Let us see the basics of the CRT.

1.2.1 Cathode-Ray-Tubes

A CRT is an evacuated glass tube. An **electron gun** at the rear of the tube produces a beam of electrons which is directed towards the front of the tube (screen) by a high voltage typically 15000 to 20,000 volts. The negatively charged electrons are then accelerated toward the phosphor coating at the inner side of the screen by a high positive voltage or by using **accelerating anode**. The phosphor substance gives off light when it is stroked by electrons.

One way to keep the phosphor glowing is to redraw the picture repeatedly by quickly directing the electron beam back over the same points. This type of display is called a **refresh CRT**.

The **control grid** voltage determines the velocity achieved by the electrons before they hit the phosphor. The control grid voltage determines how many electrons are actually in the electron beam.

A more negative voltage applied to the control grid will shut off the beam. A smaller negative voltage on the control grid simply decreases the number of electrons passing through. Since the amount of light emitted by the phosphor coating depends on the number of electrons striking the screen, we control the brightness of a display by varying the voltage on the control grid.

The **focusing system** concentrates the electron beam so that the beam converges to a small point when it hits the phosphor coating. It is possible to control the point at which the electron beam strikes the screen, and therefore the position of the dot upon the screen, by deflecting the electron beam. Fig. 1.2.1 shows the electrostatic deflection of the electron beam in a CRT.

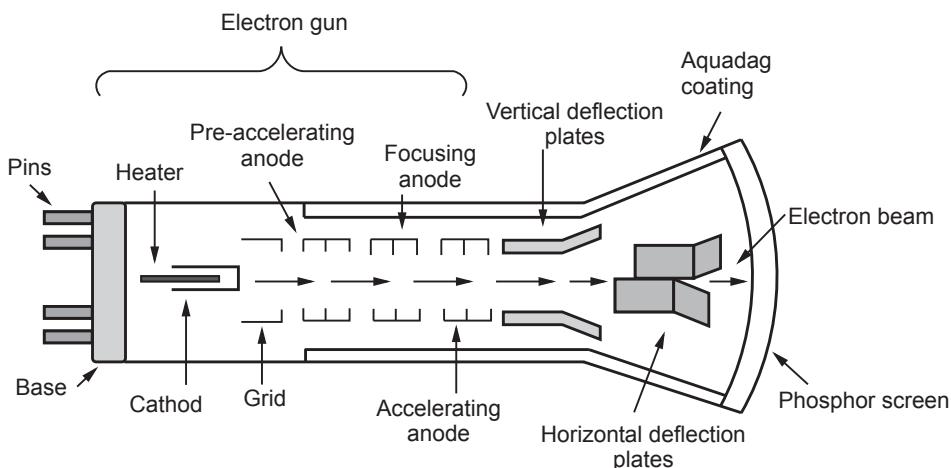


Fig. 1.2.1 Cathode ray tube

The **deflection system** of the cathode-ray-tube consists of two pairs of parallel plates, referred to as the **vertical** and **horizontal deflection** plates. The voltage applied to vertical plates controls the vertical deflection of the electron beam and voltage applied to the horizontal deflection plates controls the horizontal deflection of the electron beam.

The electrons that strike the screen, release secondary emission electrons. Aqueous graphite solution known as **Aquadag** collects these secondary electrons. To maintain the CRT in electrical equilibrium, it is necessary to collect the secondary electrons.

1.2.2 Vector Scan/Random Scan Display

As shown in Fig. 1.2.2, vector scan CRT display directly traces out only the desired lines on CRT i.e. If we want a line connecting point A with point B on the vector graphics display, we simply drive the beam deflection circuitry, which will cause beam to go directly from point A to B. If we want to move the beam from point A to point B without showing a line between points, we can blank the beam as we move it. To move the beam across the CRT, the information about both, magnitude and direction is required. This information is generated with the help of vector graphics generator.

The Fig. 1.2.3 shows the typical vector display architecture. It consists of display controller, Central Processing Unit (CPU), display buffer memory and a CRT. A display controller is connected as an I/O peripheral to the central processing unit (CPU). The display buffer memory stores the computer produced display list or display program. The program contains point and line plotting commands with (x, y) or (x, y, z) end point co-ordinates, as well as character plotting commands.

The display controller interprets commands for plotting points, lines and characters and sends digital and point co-ordinates to a vector generator. The vector generator then converts the digital co-ordinate values to analog voltages for beam-deflection circuits that displace an electron beam writing on the CRT's phosphor coating.

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

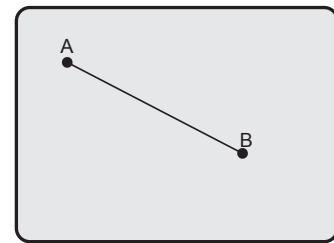


Fig. 1.2.2 Vector scan CRT

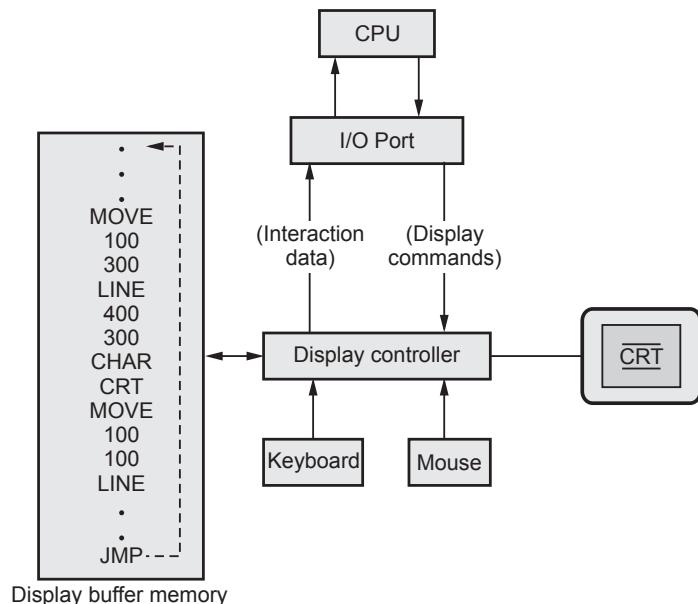


Fig. 1.2.3 Architecture of a vector display

1.2.3 Raster Scan Display and Frame Buffer

The Fig. 1.2.4 shows the architecture of a raster display. It consists of display controller, central processing unit (CPU), video controller, refresh buffer, keyboard, mouse and the CRT.

As shown in the Fig. 1.2.4, the display image is stored in the form of 1s and 0s in the refresh buffer. The video controller reads this refresh buffer and produces the actual image on the screen. It does this by scanning one scan line at a time, from top to bottom and then back to the top, as shown in the Fig. 1.2.4.

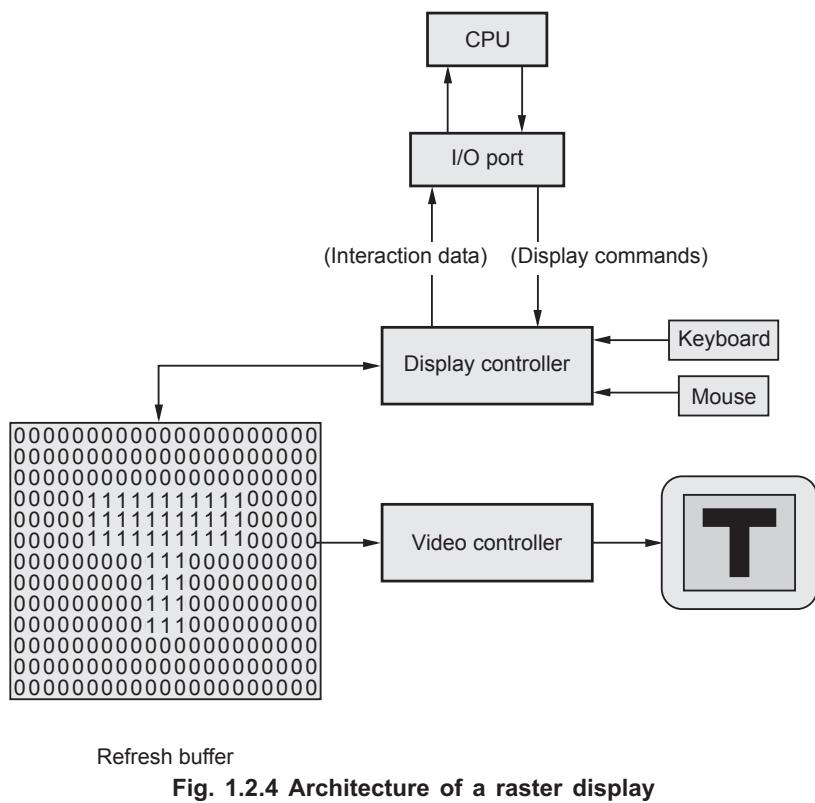


Fig. 1.2.4 Architecture of a raster display

Raster scan is the most common method of displaying images on the CRT screen. In this method, the horizontal and vertical deflection signals are generated to move the beam all over the screen in a pattern shown in the Fig. 1.2.5.

Here, the beam is swept back and forth from the left to the right across the screen. When the beam is moved from the left to the right, it is ON. The beam is OFF, when it is moved from the right to the left as shown by dotted line in Fig. 1.2.5.

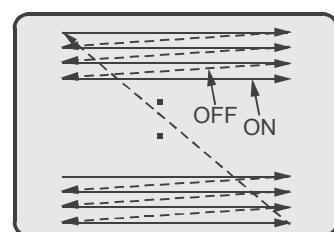


Fig. 1.2.5 Raster scan CRT

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

Frame Buffer

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

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

Sr. No.	Vector (Random) Scan Display	Raster Scan Display
1.	In vector scan display the beam is moved between the end points of the graphics primitives.	In raster scan display the beam is moved all over the screen one scan line at a time, from top to bottom and then back to top.
2.	Vector display flickers when the number of primitives in the buffer becomes too large.	In raster display, the refresh process is independent of the complexity of the image.
3.	Scan conversion is not required.	Graphics primitives are specified in terms of their endpoints and must be scan converted into their corresponding pixels in the frame buffer.
4.	Scan conversion hardware is not required.	Because each primitive must be scan-converted, real time dynamics is far more computational and requires separate scan conversion hardware.

5.	Vector display draws a continuous and smooth lines.	Raster display can display mathematically smooth lines, polygons, and boundaries of curved primitives only by approximating them with pixels on the raster grid.
6.	Cost is more.	Cost is low.
7.	Vector display only draws lines and characters.	Raster display has ability to display areas filled with solid colours or patterns.

Table 1.2.1

Example 1.2.1 How much memory is needed for the frame buffer to store a 640×400 display 16 gray levels ?

Solution : $2^4 = 16$. Therefore, 4-bits are required to store 16 gray levels.

$$\begin{aligned} \therefore \text{Frame buffer memory} &= 640 \times 400 \times 4 \\ &= 1024000 \text{ bits} = 128 \text{ kbytes} \end{aligned}$$

Example 1.2.2 Find the refresh rate of a 512×512 frame buffer, if the access time for each pixel is 200 nanoseconds.

SPPU : Dec.-10, Marks 4

Solution :

$$\begin{aligned} \text{Refresh rate} &= \frac{1}{\text{Access time for each pixel} \times \text{Number of pixels information in frame buffer}} \\ &= \frac{1}{200 \times 10^{-9} \times 512 \times 512} = 19.0734 \text{ frames / sec.} \end{aligned}$$

Example 1.2.3 Find the amount of memory required by an 8 plane frame buffer each of red, green, blue having resolution of 1024×768 .

SPPU : Dec.-12, Marks 4

Solution : For 8 plane frame buffer each of red, green and blue requires 24-bits to represent per pixel.

$$\begin{aligned} \therefore \text{Amount of memory required} &= 24 \times 1024 \times 768 \\ &= 18874368 \text{ bits} \\ &= 2359296 \text{ bytes} \end{aligned}$$

Example 1.2.4 How long would it take to load a 1280 by 1024 frame buffer with 12 bits per pixel if transfer rate is 1 Mbps ?

Solution :

$$\text{Time required to load frame buffer} = \frac{1280 \times 1024 \times 12}{10^6} = 15.72 \text{ sec}$$

1.2.4 Colour CRT Monitors

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

- Beam-penetration technique and
- Shadow-mask technique

Beam-penetration Technique

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

Merits and Demerits

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

Shadow Mask Technique

The shadow mask technique produces a much wider range of colours than the beam penetration technique. Hence this technique is commonly used in raster-scan displays including colour TV. In a shadow mask technique, CRT has three phosphor colour dots at each pixel position. One phosphor dot emits a red light, another emits a green light, and the third emits a blue light. The Fig. 1.2.6 shows the shadow mask CRT. It has three

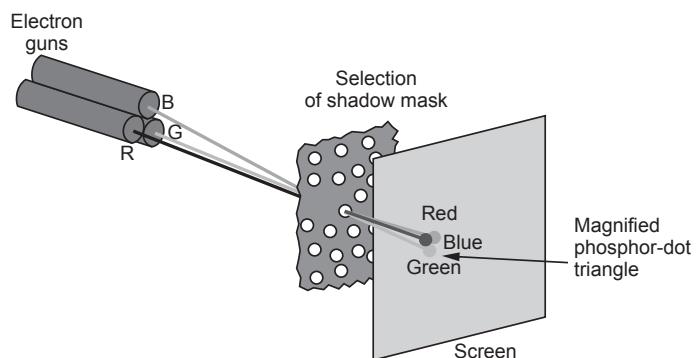


Fig. 1.2.6

electron guns, one for each colour dot, and a shadow mask grid just behind the phosphor coated screen.

The shadow mask grid consists of series of holes aligned with the phosphor dot pattern. As shown in the Fig. 1.2.6, three electron beams are deflected and focused as a group onto the shadow mask and when they pass through a hole in the shadow mask, they excite a dot triangle. A dot triangle consists of three small phosphor dots of red, green and blue colour. These phosphor dots are arranged so that each electron beam can activate only its corresponding colour dot when it passes through the shadow mask. A dot triangle when activated appears as a small dot on the screen which has colour of combination of three small dots in the dot triangle. By varying the intensity of the three electron beams we can obtain different colours in the shadow mask CRT.

1.2.5 Direct-View Storage Tubes

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

The Fig. 1.2.7 shows the general arrangement of the DVST. It consists of two electron guns : a primary gun and a flood gun. A primary gun stores the picture pattern and the flood gun maintains the picture display. A primary gun produces high speed electrons which strike on the storage grid to draw the picture pattern. As electron beam strikes on the storage grid with high speed, it knocks out electrons from the storage grid keeping the net positive charge.

The knocked out electrons are attracted towards the collector. The net positive charge on the storage grid is nothing but the picture pattern. The continuous low speed electrons from flood gun pass through the control grid and are attracted to the positive charged areas of the storage grid. The low speed electrons then penetrate the storage grid and strike the phosphor coating without affecting the positive charge pattern on the storage grid. During this process the collector just behind the storage grid smooths out the flow of flood electrons.

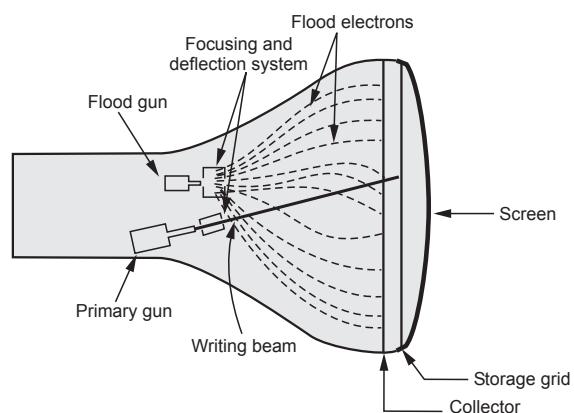


Fig. 1.2.7 Arrangement of the DVST

Advantages of DVST

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

Disadvantages of DVST

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

1.2.6 Flat Panel Displays

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

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

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

1.2.7 Plasma Panel Display

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

The Fig. 1.2.8 shows the construction of plasma panel display. It consists of two plates of glass with thin, closely spaced gold electrodes. The gold electrodes are attached to the inner faces and covered with a dielectric material. These are attached as a series of vertical conducting ribbons on one glass plate, and a set of horizontal ribbons to the other glass plate. The space between two glass plates is filled with neon-based gas and

sealed. By applying voltages between the electrodes the gas within the panel is made to behave as if it were divided into tiny cells, each one independent of its neighbours. These independent cells are made to glow by placing a firing voltage of about 120 volts across it by means of the electrodes. The glow can be sustained by maintaining a high frequency alternating voltage of about 90 volts across the cell. Due to this refreshing is not required.

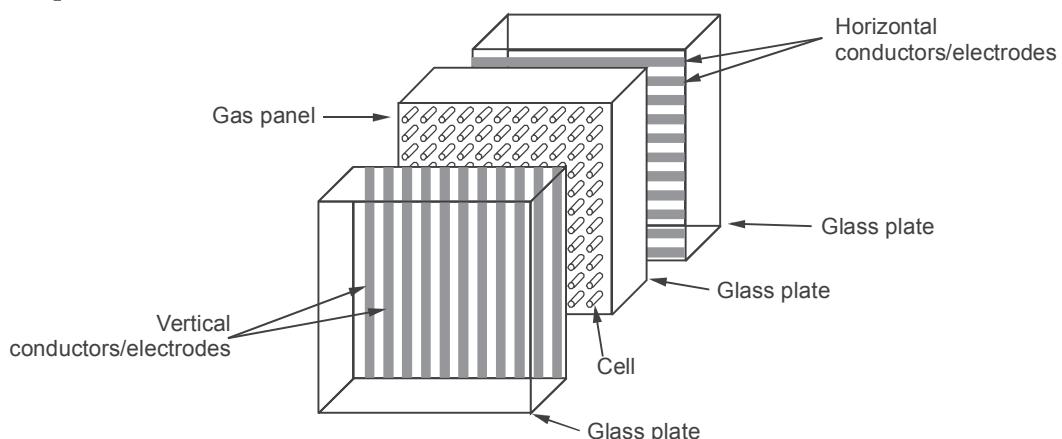


Fig. 1.2.8 Construction of plasma panel display

Advantages

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

Disadvantages

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

1.2.8 Liquid Crystal Monitors

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

Two glass plates, each containing a light polarizer at right angles to the other plate sandwich the liquid-crystal material. Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate. The intersection of two conductors defines a pixel position. In the ON state, polarized light passing through material is twisted so that it will pass through the opposite polarizer. It is then reflected back to the viewer. To turn OFF the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that the light is not twisted as shown in the Fig. 1.2.9. This type of flat panel device is referred to as a passive matrix LCD. In **active matrix LCD**, transistors are used at each (x, y) grid point. Use of transistors cause the crystal to change their state quickly and also to control the degree to which the state has been changed. These two properties allow LCDs to be used in miniature television sets with continuous-tone images.

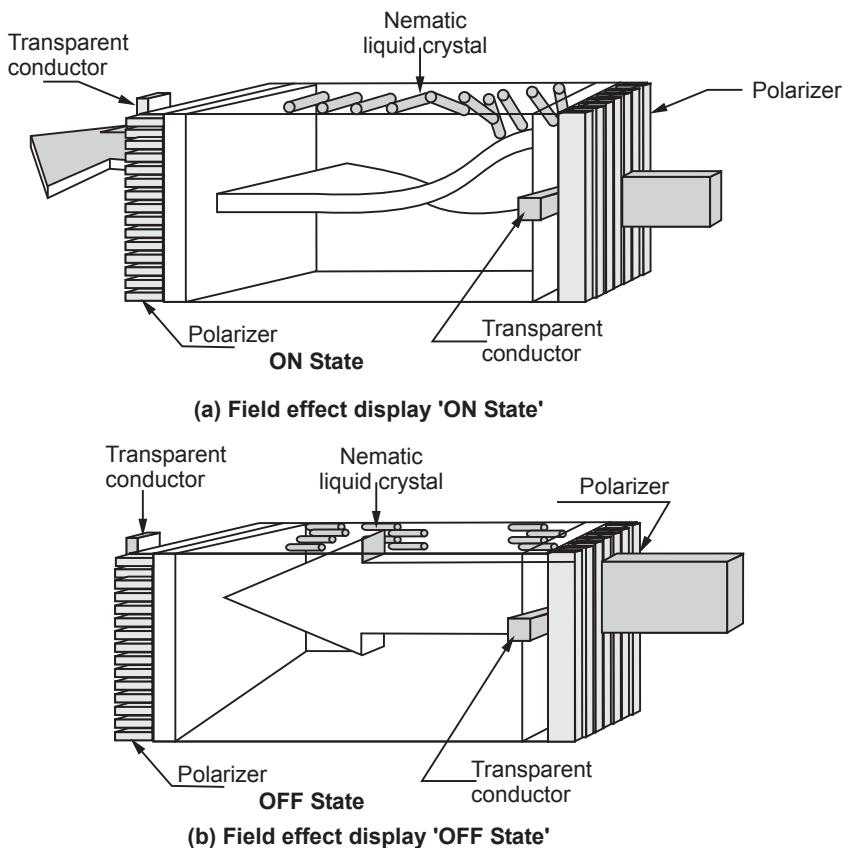


Fig. 1.2.9

The transistor can also serve as a memory for the state of a cell and can hold the cell in that state until it is changed. Thus use of transistor make cell on for all the time giving brighter display than it would be if it had to be refreshed periodically.

Advantages of LCD Displays

- Low cost
- Low weight
- Small size
- Low power consumption

1.2.9 Three-Dimensional Viewing Devices

Graphics monitors for the display of three-dimensional scenes employ the varying focal properties of a vibrating mirror to cause sequentially generated image components to appear 3-D in space. The Fig. 1.2.10 shows such three-dimensional display system. The mirror is constructed as a circular reflective plate having substantial stiffness and resilience. As the mirror vibrates, it changes focal length such that changes in the focal length are proportional to the depth of points in a scene. These vibrations are synchronized with the display of the object on a CRT. Due to this each point on the object is reflected from the mirror into a spatial position corresponding to the distance of that point from the specified viewing position. As a result we can view the object or scene from different sides.

Such a three-dimensional systems are very useful in medical applications such as ultrasonography and CAT scan to analyze data. They are also useful in three dimensional simulation systems and in geological applications to analyze topological and seismic data.

Stereoscopic and Virtual Reality Systems

Creating stereoscopic views is an another technique of viewing three-dimensional objects. The view obtained by such technique is not a true 3D view, but it does provide a 3D effect by presenting a different view to each eye of an observer so that scenes do appear to have depth.

To obtain a stereoscopic projection we have to obtain two views of scene generated from a viewing direction corresponding to each eye (left and right). Now looking

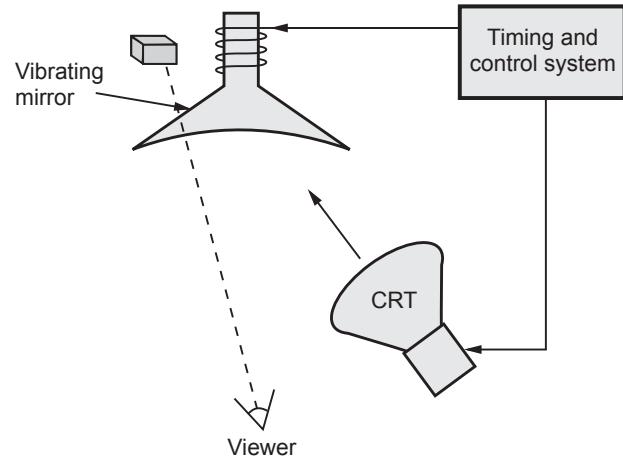


Fig. 1.2.10 Three-dimensional display system

simultaneously at the left view with the left eye and the right view with right eye, two views are merged into a single image providing necessary depth to appear as an 3D image. Two views of a scene can be obtained using computer or we can use a stereo camera pair to photograph scene.

We can produce stereoscopic effect in raster system by :

- Displaying each of the two views on alternate refresh cycles
- And then viewing screen through glasses with each lens designed to act as a rapidly alternating shutter that is synchronized to block out one of the views.

The stereoscopic viewing plays important role in the virtual reality systems. In such systems, the viewer can step into stereoscopic scenes and interact with the environment. Usually, a headset containing an optical system is used to generate the stereoscopic views.

1.2.10 Persistence, Resolution and Aspect Ratio

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

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

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

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

Example 1.2.1 If image of size 1024×800 needs to resize to one that has 640 pixels width with the same aspect ratio, what would be the height of the resized image ?

Solution : Height of the resized image in pixels = $\frac{800}{1024} \times 640 = 500$

1.2.11 Applications of Large Screen Displays

1. **Education** : Many institutes use large screen displays to give effective audio-visual training in the class-room.
2. **Entertainment** : Large screen displays are used to watch live matches in the hotels or in the community halls.
3. **Sports ground** : Large screen displays are used to show current scores in the stadiums.
4. **Advertisement** : Large screen displays are used for advertising purpose in many public places.
5. **Railway, airway and roadway transport** : Large screen displays are used at the railway stations, airports and bus stands to provide the schedule information to the travelers.
6. **Seminars, presentations and video-conferencing** : Large screen displays are used during presentations, seminars and video-conferencing.
7. **Space stations** : Large screen displays are used in space stations to display the satellite images of the planets. They are used as one of the visual communication medium for communicating with the astronauts.

Plasma-panel and flat panel displays support large screen displays.

Review Questions

1. Explain the working of cathode ray tube with a diagram.
2. Define random scan and raster scan displays. **SPPU : May-14, Marks 5**
3. Compare raster scan and vector scan displays.
4. Write a note on raster scan.
5. Define the following term : Frame buffer. **SPPU : Dec.-06,10,11,12,15, May-09,13, Marks 2**
6. Explain shadow mask technique and explain how does it differ from beam penetration technique ?
7. Write short notes on beam-penetration technique and shadow mask technique.
8. Explain Direct-View Storage Tubes (DVST). How persistence characteristics of phosphor affect on refresh rate of system ?
9. List merit and demerit of DVST.
10. Write a short note on a) Flat panel display b) Plasma panel display.
11. List merit and demerit of plasma panel display.
12. List the important characteristics of video display devices.
13. Write a note on liquid crystal displays.
14. Write a short note on three-dimensional viewing devices.
15. Explain important characteristics of video display devices.
16. List the applications of large screen displays.
17. Explain following terms
 1. Persistence 2. Resolution 3. Aspect ratio.**SPPU : May-05, 14, 15, Dec.-15, Marks 4**
18. Define the following terms : 1) Persistence 2) Frame buffer 3) Resolution 4) Aspect ratio.

1.3 Applications of Computer Graphics

SPPU : Dec.-14, May-16

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

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

symbols and identifications. This makes it easy for operator to monitor and control various processing parameters at a time.

- **Cartography :** Computer graphics is also used to represent geographic maps, weather maps, oceanographic charts, contour maps, population density maps and so on.
- **Education and Training :** Computer graphics can be used to generate models of physical, financial and economic systems. These models can be used as educational aids. Models of physical systems, physiological systems, population trends, or equipment, such as colour-coded diagram can help trainees to understand the operation of the system.
- **Image Processing :** In computer graphics, a computer is used to create pictures. Image processing, on the other hands, applies techniques to modify or interpret existing pictures such as photographs and scanned images. Image processing and computer graphics are typically combined in many applications such as to model and study physical functions, to design artificial limbs, and to plan and practice surgery. Image processing techniques are most commonly used for picture enhancements to analyze satellite photos, X-ray photography and so on.

Review Questions

1. *What is computer graphics ? State the applications of computer graphics.*

SPPU : Dec.-14, Marks 6

2. *Write and explain any four state of the applications of computer graphics.*

SPPU : May-16, Marks 4



UNIT - I

2

Introduction to OpenGL

Syllabus

*OpenGL architecture, primitives and attributes, simple modelling and rendering of two- and three-dimensional geometric objects, GLUT, interaction, events and call-backs picking. (**Simple Interaction with the Mouse and Keyboard**)*

Contents

- 2.1 *OpenGL Architecture*
- 2.2 *Primitives and Attributes*
- 2.3 *Simple Modelling and Rendering of 2D and 3D Geometric Objects*
- 2.4 *Interaction*
- 2.5 *Picking*

2.1 OpenGL Architecture

- OpenGL (Open Graphics Library) is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D computer graphics.
- The API is typically used to interact with a Graphics Processing Unit (GPU), to achieve hardware-accelerated rendering.
- A basic library of functions is provided in OpenGL for specifying graphics primitives, attributes, geometric transformations, viewing transformations and many other operations.
- OpenGL is designed to be hardware independent, therefore many operations, such as input and output routines, are not included in the basic library. However, input and output routines and many additional functions are available in auxiliary libraries that have been developed for OpenGL programs.

2.1.1 Basic OpenGL Syntax

- Function names in the **OpenGL basic library** (also called the **OpenGL core library**) are prefixed with **gl** and each component word within a function name has its first letter capitalized. The following examples illustrate this naming convention.
glBegin, **glClear**, **glCopyPixels**, **glPolygonMode**
- Some functions in OpenGL require that one (or more) of their arguments be assigned a symbolic constant specifying, for instance, a parameter name, a value for a parameter or a particular mode. All such constants begin with the uppercase letters **GL**. In addition, component words within a constant name are written in capital letters and the underscore () is used as a separator between all component words in the name. Following are a few examples of the several hundred symbolic constants available for use with OpenGL functions.
GL_2D, **GL_RGB**, **GL_CCW**, **GL_POLYGON**, **GL_AMBIENT_AND_DIFFUSE**
- The OpenGL functions **uses specific data types**. For example, an OpenGL function parameter might use a value that is specified as a 32-bit integer. But the size of an integer specification can be different on different machines.
- Special built-in, data-type supported by OpenGL are : **GLbyte**, **GLshort**, **GLint**, **GLfloat**, **GLdouble**, **GLboolean**. Each data-type name begins with the capital letters **GL** and the remainder of the name is a standard data-type designation, written in lower-case letters.
- It is also possible to assign values using an array that lists a set of data values for some arguments of OpenGL functions. These values are specified as a pointer to an array, rather than specifying each element of the list explicitly as a parameter argument.

2.1.2 Related Libraries

- In addition to the OpenGL basic (core) library, there are a number of associated libraries for handling special operations.

Libraries	Supported Operations
OpenGL Utility (GLU)	<p>Provides routines for setting up viewing and projection matrices, describing complex objects with line and polygon approximations, displaying quadrics and B-splines using linear approximations, processing the surface-rendering operations and other complex tasks.</p> <p>GLU function names start with the prefix <code>glu</code>.</p>
Object Oriented Toolkit (Open Inventor)	<p>Provides routines and predefined object shapes for interactive three-dimensional applications.</p> <p>This toolkit is written in C++.</p>

- To create a graphics display using OpenGL, we first need to set up a display window on our video screen. This is simply the rectangular area of the screen in which our picture will be displayed.
- We cannot create the display window directly with the basic OpenGL functions, because of following two reasons :
 - The basic OpenGL functions contains only device independent graphics functions and
 - Window-management operations are machine depend.
- However, there are several window-system libraries that support OpenGL functions for a variety of machines.

Extension/Interface	Supported Operations
(GLX)	<p>OpenGL Extension to the X Window System.</p> <p>Provides a set of routines that are prefixed with the letters <code>glx</code>.</p>
Apple GL (AGL)	<p>Apple systems can use the Apple GL (AGL) interface for window-management operations.</p> <p>Function names for this library are prefixed with <code>agl</code>.</p>
WGL	<p>Provide a Windows-to-OpenGL interface for Microsoft Windows systems.</p> <p>These routines are prefixed with the letters <code>wgl</code>.</p>
Presentation Manager to OpenGL (PGL)	Provides interface for the IBM OS/2, which uses the prefix <code>pgl</code> for the library routines.
OpenGL Utility Toolkit (GLUT)	Provides a library of functions for interacting with any screen-windowing system. The GLUT library functions are prefixed with <code>glut</code> , and this library also contains methods for describing and rendering quadric curves and surfaces.

2.1.3 Header Files

- For graphics programs we need to include the header file for the OpenGL core library.
- For most applications we will also need GLU. For instance, with Microsoft Windows, we need to include the header file (windows.h) for the window system.
- This header file must be listed before the OpenGL and GLU header files because it contains macros needed by the Microsoft Windows version of the OpenGL libraries. So the source file in this case would begin with
`#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>`

- However, if we use GLUT to handle the window-managing operations, we do not need to include gl.h and glu.h because GLUT ensures that these will be included correctly. Thus, we can replace the header files for OpenGL and GLU with
`#include <GL/glut.h>`
- We could include gl.h and glu.h as well, but doing so would be redundant and could affect program portability.
- In addition, we will often need to include header files that are required by the C++ code. For example,
`#include <stdio.h>
#include <stdlib.h>
#include <math.h>`

2.1.4 Display-Window Management using GLUT

- To get started, we can consider a simplified, minimal number of operations for displaying a picture. Since we are using the OpenGL Utility Toolkit, our first step is to initialize GLUT.
- We perform the GLUT initialization with the statement
`glutInit (&argc, argv);`
- Next, we can state that a display window is to be created on the screen with a given caption for the title bar. This is accomplished with the function
`glutCreateWindow ("An Example OpenGL Program");`
- Then we need to specify what the display window is to contain. For this, we create a picture using OpenGL functions and pass the picture definition to the GLUT routine **glutDisplayFunc**, which assigns our picture to the display window.
- As an example, suppose we have the OpenGL code for describing a line segment in a procedure called **lineSegment**. Then the following function call passes the line-segment description to the display window.
`glutDisplayFunc (lineSegment);`

- But the display window is not yet on the screen. We need one more GLUT function to complete the window-processing operations. After execution of the following statement, all display windows that we have created, including their graphic content, are now activated.

```
glutMainLoop();
```

- This function must be the last one in our program. It displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as a mouse or keyboard.
- In our first program, the display window that we created will be in some default location and size, we can set these parameters using additional GLUT functions.
- We use the **glutInitWindowPosition** function to give an initial location for the top left corner of the display window.
`glutInitWindowPosition (50, 80);`
- Similarly, the **glutInitWindowSize** function is used to set the initial pixel width and height of the display window.
`glutInitWindowSize (300, 200);`
- After the display window is on the screen, we can reposition and resize it.
- We can also set a number of other options for the display window, such as buffering and a choice of color modes, with the **glutInitDisplayMode** function. Arguments for this routine are assigned symbolic GLUT constants.
- For example,
`glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);`
- The above command specifies that a single refresh buffer is to be used for the display window and that the RGB (red, green, blue) color mode is to be used for selecting color values.
- The values of the constants passed to this function are combined using a logical **or** operation. Actually, single buffering and RGB color mode are the default options.

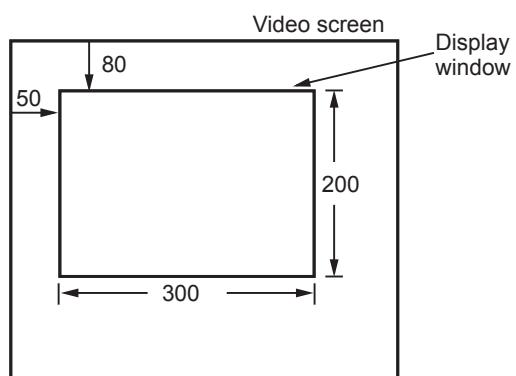


Fig. 2.1.1 A 300 by 200 display window at position (50, 80) relative to the top-left corner of the video display

2.1.5 Format of OpenGL Command

The Fig. 2.1.2 shows the format of OpenGL command.

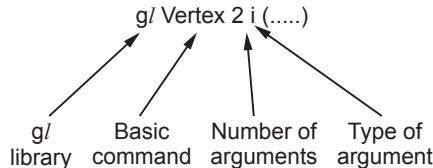


Fig. 2.1.2 Format of OpenGL command

2.1.6 Vertex Function

- We can represent a point in the plane $z = 0$ as $p = (x, y, 0)$ in the three dimensional world, or as $p = (x, y)$ in the two dimensional plane.
- OpenGL allows us to use both the representations.
- A vertex is a position in space. Single vertex defines a point, two vertices define a line segment, three vertices define a triangle or a circle, four vertices define a quadrilateral and so on.
- In OpenGL, the general syntax for vertex function is

`glvertex * ()`

where the * can be interpreted as either two or three characters of the form nt or ntv, where

- n : The number of dimensions (2, 3 or 4)
- t : Denotes data type
 - i : integer
 - f : float
 - d : double
 - v : indicates that the variables are specified through a pointer to an array, rather than through an argument list.

2.1.6.1 Multiple Forms of Vertex Functions

glvertex2i (GLint xi, GLint yi) : Function works in two dimension with integers.

glvertex3f (GLfloat x, GLfloat y, GLfloat z) : Function works in three dimension with floating point numbers.

glvertex3fv (vertex) : Function works in three dimension with floating point numbers.

Here variables are specified through a pointer to predefined array. For example, GLfloat vertex [3].

2.1.7 A Simple OpenGL Program

Fig. 2.1.3 shows a simple OpenGL program. Although this program does nothing useful, it defines a pattern that is used by most OpenGL programs. The line numbers are not part of the program, but are used in the explanation below.

```
1 #include <GL/glut.h>
2
3 void display (void)
4 {
5     glClear(GL_COLOR_BUFFER_BIT);
6 }
7
8 void init (void)
9 {
10 }
11
12 int main (int argc, char *argv[])
13 {
14     glutInit(&argc, argv);
15     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
16     glutInitWindowSize(800, 600);
17     glutInitWindowPosition(100, 50);
18     glutCreateWindow("My first openGL program");
19     init();
20     glutDisplayFunc(display);
21     glutMainLoop();
22     return 0;
23 }
```

Fig. 2.1.3 The basic OpenGL program

- Line 1.** Every OpenGL program should include GL/glut.h. The file glut.h includes glu.h, gl.h and other header files required by GLUT.
- Line 3.** You must write a function that displays the graphical objects in your model. The function shown here clears the screen but does not display anything. Your program does not call this function explicitly, but it will be called at appropriate times by OpenGL.
- Line 5.** It is usually a good idea to clear various buffers before starting to draw new objects. The colour buffer, cleared by this call, is where the image of your model is actually stored.

- Line 8.** It is a good idea to put "standard initialization" (the glut... calls) in the main program and application dependent initialization in a function with a name like init() or myInit(). We follow this convention in these text because it is convenient to give different bodies for init without having to explain where to put the initialization statements. In this program, the function init() is defined in lines 8 through 10, does nothing, and is invoked at line 19.
- Line 14.** The function **glutInit()** initializes the OpenGL library. It is conventional to pass the command line arguments to this function because it may use some of them. You will probably not need this feature.
- Line 15.** The function **glutInitDisplayMode()** initializes the display. Its argument is a bit string. Deleting this line would not affect the execution of the program, because the arguments shown are the default values.
- Line 16.** This call requests a graphics window 800 pixels wide and 600 pixels high. If this line was omitted, GLUT would use the window manager's default values for the window's size.
- Line 17.** This call says that the left side of the graphics window should be 100 pixels from the left of the screen and the top of the graphics window should be 50 pixels below the top of the screen. Note that the Y value is measured from the **top** of the screen. If this line was omitted, GLUT would use the window manager's default values for the window's position.
- Line 18.** This call creates the window using the settings given previously. The text in the argument becomes the title of the window. The window does not actually appear on the screen until **glutMainLoop()** has been called.
- Line 19.** This is a good place to perform any additional initialization that is needed (see above). Some initialization, such as calls to glEnable(), must come **after** the call to **glutCreateWindow()**.
- Line 20.** This call **registers** the callback function **display()**. After executing the call, OpenGL knows what function to call to display your model in the window. Section 2.4 describes callback functions.
- Line 21.** The last statement of an OpenGL program calls **glutMainLoop()**. This function processes events until the program is terminated. Well-written programs should provide the user with an easy way of stopping the program, for example by selecting Quit from a menu or by pressing the esc key.

Drawing Objects

All that remains is to describe additions to Fig. 2.1.3. The first one that we will consider is an improved display() function. Fig. 2.1.4 shows a function that displays a line. The call `glColor3f(1.0, 0.0, 0.0)` specifies the colour of the line as "maximum red, no green, no blue".

```
void display (void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
        glVertex2f(-1.0, 0.0);
        glVertex2f(1.0, 0.0);
    glEnd();
    glFlush();
}
```

Fig. 2.1.4 Displaying a bright red line

The construction `glBegin(mode); ...; glEnd();` is used to display groups of primitive objects. The value of mode determines the kind of object. In this case, `GL_LINES` tells OpenGL to expect one or more lines given as pairs of vertices.

In this case, the line goes from $(-1, 0)$ to $(1, 0)$, as specified by the two calls to `glVertex2f()`. If we use the default viewing region, this line runs horizontally across the centre of the window. The call `glFlush()` forces previously executed OpenGL commands to begin execution.

The suffix "3f" indicates that `glColor3f()` requires three arguments of type `GLfloat`. Similarly, `glVertex2f()` requires two arguments of type `GLfloat`. The code between `glBegin()` and `glEnd()` is not restricted to gl calls. Fig. 2.1.5 shows a display function that uses a loop to draw 51 vertical yellow (red + green) lines across the window. Section 2.3 describes some of the other primitive objects available in OpenGL.

```
void display (void)
{
    int i;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_LINES);
        for (i = -25; i <= 25; i++)
            glVertex2f(i, -1.0);
            glVertex2f(i, 1.0);
    glEnd();
    glFlush();
}
```

```
{
float x = i / 25.0;
glVertex2f(x, -1.0);
glVertex2f(x, 1.0);
}
glEnd();
}
```

Fig. 2.1.5 Drawing vertical yellow lines

The call `glClear(GL_COLOR_BUFFER_BIT)` clears the colour buffer to the background colour. You can set the background colour by executing

```
glClearColor(r, g, b, a);
```

with appropriate values of r, g, b and a during initialization. Each argument is a floating point number in the range [0, 1] specifying the amount of a colour (red, green, blue) or blending (alpha). The default values are all zero, giving a black background. To make the background blue, you could call

```
glClearColor(0.0, 0.0, 1.0, 0.0);
```

Blending is an advanced feature; unless you know how to use it, set the fourth argument of `glClearColor` to zero.

2.1.8 A Complete OpenGL Program

- For complete program we have to perform following additional task.
 - For the display window, we can choose a background color.
 - We need to construct a procedure that contains the appropriate OpenGL functions for the picture that we want to display.
- Using RGB color values, we set the background color for the display window to be white with the OpenGL function
`glClearColor (1.0, 1.0, 1.0, 0.0);`
- The first three arguments in this function set each of the red, green and blue component colors to the value 1.0. Thus we get a white color for the display window. We can set each of the component colors to 0.0, to get a black background.
- The fourth parameter in the `glClearColor` function is called the **alpha value** for the specified color. One use for the alpha value is as a “**blending**” parameter. When we activate the OpenGL blending operations, alpha values can be used to determine the resulting color for two overlapping objects. An alpha value of 0.0 indicates a totally transparent object and an alpha value of 1.0 indicates an opaque object.

- At this point we are not using Blending operations, so the value of alpha is irrelevant. For now, we simply set alpha to 0.0.
- Although the `glClearColor` command assigns a color to the display window, it does not put the display window on the screen. To get the assigned window color displayed, we need to invoke the following OpenGL function.
`glClear (GL_COLOR_BUFFER_BIT);`
- The argument `GL_COLOR_BUFFER_BIT` is an OpenGL symbolic constant specifying that it is the bit values in the color buffer (refresh buffer) that are to be set to the values indicated in the **glClearColor function**.
- In addition to setting the background color for the display window, we can choose a variety of color schemes for the objects we want to display in a scene.
- For example, the following function sets object color to be blue.

```
glColor3f (0.0, 0.0, 1.0);
```

- The suffix `3f` on the `glColor` function indicates that we are specifying the three RGB color components using floating-point (`f`) values. These values must be in the range from 0.0 to 1.0 and we have set red and green = 0.0 and blue = 1.0.
- In our first program, we simply display a two-dimensional line segment. To do this, we have to set the projection type (mode) and other viewing parameters to tell OpenGL how we want to “project” our picture onto the display window.
- We can set the projection type (mode) and other viewing parameters with following two functions.

```
glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0, 150.0, 0.0, 120.0);
```

- The above functions specify that an orthogonal projection is to be used to map the contents of a two-dimensional (2D) rectangular area of world coordinates to the screen and that the x-coordinate values within this rectangle range from 0.0 to 150.0 with y-coordinate values ranging from 0.0 to 120.0.
- Whatever objects we define within this world-coordinate rectangle will be shown within the display window. Anything outside this coordinate range will not be displayed. Therefore, the GLU function `gluOrtho2D` defines the coordinate reference frame within the display window to be (0.0, 0.0) at the lower-left corner of the display window and (150.0, 120.0) at the upper-right window corner.
- Finally, we need to call the appropriate OpenGL routines to create the line segment. The following code defines a two-dimensional, straight-line segment with integer, Cartesian endpoint coordinates (110, 25) and (20, 100).

```
glBegin (GL_LINES);
 glVertex2i (110, 25);
```

```
glVertex2i (20, 100);
glEnd ( );
```

- Let us see the entire program organized into three procedures. We place all initializations and related one-time parameter settings in procedure in it.
- Our geometric description of the “picture” we want to display is in procedure lineSegment, which is the procedure that will be referenced by the GLUT function glutDisplayFunc.
- The main procedure contains the GLUT functions for setting up the display window and getting the line segment onto the screen.

```
#include <GL/glut.h> // Includes the header file
                      // depending on the system in use, this may change)

void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 0.0);           // Set display-window color to white
    glMatrixMode (GL_PROJECTION);             // Set projection parameters
    gluOrtho2D (0.0, 150.0, 0.0, 120.0);
}

void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);           // Clear display window
    	glColor3f (0.0, 0.0, 1.0);              // Set line segment color to blue
     glBegin (GL_LINES);
     glVertex2i (110, 25);                  // Specify line-segment geometry.
     glVertex2i (20, 100);
     glEnd ( );
     glFlush ( );                         // Process all OpenGL routines as quickly as
                                             // possible.
}
void main (int argc, char** argv)
{
    	glutInit (&argc, argv);                   //Initialize GLUT
    	glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB); // Set display mode
    	glutInitWindowPosition (50, 80);          // Set top-left position of display-window
    	glutInitWindowSize (300, 200);            // Set width and height of display-window
    	glutCreateWindow ("Draw Line");          // Create display window
    	init ();                            // Execute initialization procedure.
    	glutDisplayFunc (lineSegment);          // Send graphics to display window.
    	glutMainLoop ( );                     // Display everything and wait.
}
```

2.1.9 OpenGL Data Types

Table 2.1.1 shows the data types supported by OpenGL.

Suffix	Data type	C Type	OpenGL Type
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int	GLuint, GLenum, GLbitfield
	Nothing	void	GLvoid

Table 2.1.1 OpenGL data types

2.1.10 OpenGL Function Types

- An API for interfacing with graphics system can contain hundreds of individual functions. These functions are divided into seven major groups :
 1. Primitive functions
 2. Attribute functions
 3. Viewing functions
 4. Transformation functions
 5. Input functions
 6. Control functions
 7. Query functions
- **Primitive functions** : These functions define the low-level objects or atomic entities that our system can display. Depending on the API, the primitives can include points, line segments, polygons, pixels, text, and various types of curves and surfaces.
- **Attribute functions** : These functions allow us to perform operations ranging from choosing the color with which we display a line segment, to picking a pattern with which to fill the inside of a polygon, to selecting a typeface for the titles on a graph.

- **Viewing functions** : These functions allow us to specify various views, although APIs differ in the degree of flexibility they provide in choosing a view.
- **Transformation functions** : These functions allow us to carry out transformations of objects, such as rotation, translation, and scaling. Providing the user with a set of transformation functions is one of the characteristics of a good API.
- **Input functions** : These functions allow us to deal with the diverse forms of input that characterize modern graphics systems. These functions include the functions to deal with devices such as keyboards, mice, and data tablets.
- **Control functions** : These functions enable us to communicate with the window system, to initialize our programs, and to deal with any errors that take place during the execution of our programs.
- **Query functions** : Within our applications we can often use other information within the API, including camera parameters or values in the frame buffer. A good API provides this information through a set of query functions.

Review Questions

1. *What is OpenGL ?*
2. *List the datatypes supported by OpenGL.*
3. *List the libraries supported by OpenGL and their supported operations.*
4. *Explain the display window management using GLUT.*
5. *Draw the format of OpenGL command.*
6. *Explain the vertex function with examples.*
7. *Explain the various types of function supported by OpenGL.*

2.2 Primitives and Attributes

- There are two types of primitives :
 - **Geometric primitives** and
 - **Image or raster primitives**

Geometric primitives

- Geometric primitives are specified in the problem domain and include points, line segments, polygons, curves, and surfaces.
- These primitives pass through a geometric pipeline, as shown in Figure 2.2.1, where they are subject to a series of geometric operations.

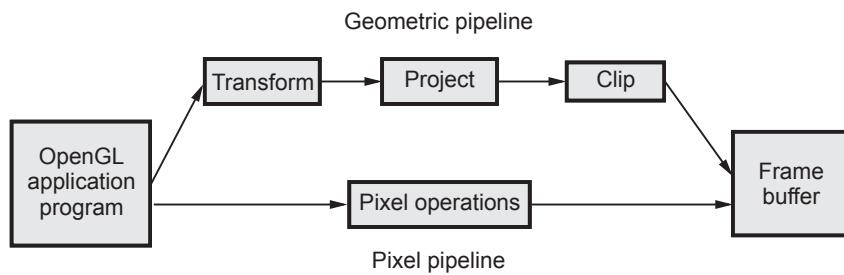


Fig. 2.2.1 Simplified OpenGL pipeline

- Series of geometric operations determine :
 - whether a primitive is visible,
 - where on the display it appears if it is visible, and
 - the rasterization of the primitive into pixels in the frame buffer.
- Because geometric primitives exist in a two- or three-dimensional space, they can be manipulated by operations such as rotation and translation.
- Geometric operations can be used as building blocks for other geometric objects.

Raster primitives

- Raster primitives, such as arrays of pixels, lack geometric properties and cannot be manipulated in space in the same way as geometric primitives.
- They pass through a separate parallel pipeline on their way to the frame buffer.

2.2.1 Basic OpenGL Primitives

- OpenGL supports several basic primitive types: points, lines, quadrilaterals, and general polygons. All of these primitives are specified using a sequence of vertices.
- Fig. 2.2.2 shows the basic primitive types, where the numbers indicate the order in which the vertices have been specified. Note that for the GL_LINES primitive only every second vertex causes a line segment to be drawn. Similarly, for the GL_TRIANGLES primitive, every third vertex causes a triangle to be drawn. Note that for the GL_TRIANGLE_STRIP and GL_TRIANGLE_FAN primitives, a new triangle is produced for every additional vertex. All of the closed primitives shown below are solid-filled, with the exception of GL_LINE_LOOP, which only draws lines connecting the vertices.
- OpenGL provides ten different primitive types for drawing points, lines, and polygons, as shown in Fig. 2.2.2.

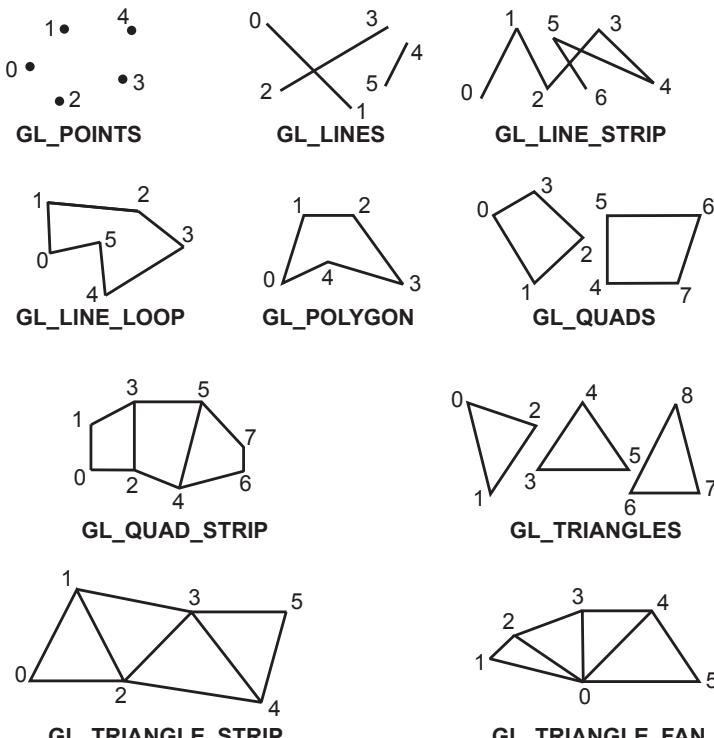


Fig. 2.2.2 OpenGL primitive types

2.2.1.1 GL_POINTS

- We use this primitive type to render mathematical points. OpenGL renders a point for each vertex specified.
 - Treats each vertex as a single point.
 - Vertex n defines a point n.

• Example :

```
glBegin(GL_POINTS);
glVertex2f(x1, y1);
glEnd();
```

2.2.1.2 GL_LINES

- We use this primitive to draw unconnected line segments. OpenGL draws a line segment for each group of two vertices. If the application specifies n vertices, OpenGL renders n/2 line segments. If n is odd, OpenGL ignores the final vertex.
 - Treats each pair of vertices as an independent line segment.
 - Vertices 2n-1 and 2n define a line n.
 - n/2 lines are drawn.

- Example :

```
glBegin(GL_LINES);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glEnd();
```

2.2.1.3 GL_LINE_STRIP

- We use this primitive to draw a sequence of connected line segments. OpenGL renders a line segment between the first and second vertices, between the second and third, between the third and fourth, and so on. If the application specifies n vertices, OpenGL renders n-1 line segments.

- Draws a connected group of line segments from the first vertex to the last.
- Vertices n and n+1 define line n.
- N-1 lines are drawn.

- Example :

```
glBegin(GL_LINE_STRIP);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glEnd();
```

2.2.1.4 GL_LINE_LOOP

- We use this primitive to close a line strip. OpenGL renders this primitive like a GL_LINE_STRIP with the addition of a closing line segment between the final and first vertices.

- Draws a connected group of line segments from the first vertex to the last, then back to the first.
- Vertices n and n+1 define line n.
- n lines are drawn.

- Example :

```
glBegin(GL_LINE_LOOP);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3,y3);
glEnd();
```

2.2.1.5 GL_TRIANGLES

- We use this primitive to draw individual triangles. OpenGL renders a triangle for each group of three vertices. If your application specifies n vertices, OpenGL renders $n/3$ triangles. If n isn't a multiple of 3, OpenGL ignores the excess vertices.
 - Treats each triplet of vertices as an independent triangle.
 - Vertices $3n-2$, $3n-1$, and $3n$ define triangle n .
 - $n/3$ triangles are drawn.

- Example :

```
glBegin(GL_TRIANGLES);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glEnd();
```

2.2.1.6 GL_TRIANGLE_STRIP

- We use this primitive to draw a sequence of triangles that share edges. OpenGL renders a triangle using the first, second, and third vertices, and then another using the second, third, and fourth vertices, and so on. If the application specifies n vertices, OpenGL renders $n-2$ connected triangles. If n is less than 3, OpenGL renders nothing.
 - Draws a connected group of triangles.
 - One triangle is defined for each vertex presented after the first two vertices.
 - For odd n , vertices n , $n+1$, and $n+2$ define triangle n .
 - For even n , vertices $n+1$, n , and $n+2$ define triangle n .
 - $n-2$ triangles are drawn.

- Example :

```
glBegin(GL_TRIANGLE_STRIP);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glVertex2f(x4, y4);
glEnd();
```

2.2.1.7 GL_TRIANGLE_FAN

- We use this primitive to draw a fan of triangles that share edges and also share a vertex. Each triangle shares the first vertex specified. If the application specifies a sequence of vertices v , OpenGL renders a triangle using v_0, v_1 and v_2 ; another triangle using v_0, v_2 and v_3 ; another triangle using v_0, v_3 and v_4 and so on. If the application specifies n vertices, OpenGL renders $n-2$ connected triangles. If n is less than 3, OpenGL renders nothing.
 - Draws a connected group of triangles that fan around a central point.
 - One triangle is defined for each vertex presented after the first two vertices.
 - Vertices 1, $n+1$, and $n+2$ define triangle n .
 - $n-2$ triangles are drawn.
- Example :

```
glBegin(GL_TRIANGLE_FAN);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glVertex2f(x4, y4);
glEnd();
```

2.2.1.8 GL_QUADS

- We use this primitive to draw individual convex quadrilaterals. OpenGL renders a quadrilateral for each group of four vertices. If the application specifies n vertices, OpenGL renders $n/4$ quadrilaterals. If n isn't a multiple of 4, OpenGL ignores the excess vertices.
 - Treats each group of four vertices as an independent quadrilateral.
 - Vertices $4n-3, 4n-2, 4n-1$, and $4n$ define quadrilateral n .
 - $n/4$ quadrilaterals are drawn.
- Example :

```
glBegin(GL_QUADS);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glVertex2f(x4, y4);
glEnd();
```

2.2.1.9 GL_QUAD_STRIP

- We use this primitive to draw a sequence of quadrilaterals that share edges. If the application specifies a sequence of vertices v, OpenGL renders a quadrilateral using v0, v1, v3 and v2 ; another quadrilateral using v2 ,v3 ,v5 and v4 ; and so on. If the application specifies n vertices, OpenGL renders $(n-2)/2$ quadrilaterals. If n is less than 4, OpenGL renders nothing.
 - Draws a connected group of quadrilaterals.
 - Treats each group of four vertices as an independent quadrilateral.
 - Vertices $4n-3$, $4n-2$, $4n-1$, and $4n$ define quadrilateral n.
 - $n/4$ quadrilaterals are drawn.
- Example :

```
glBegin(GL_QUADS);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
```

2.2.1.10 GL_POLYGON

- We use GL_POLYGON to draw a single filled convex n-gon primitive. OpenGL renders an n-sided polygon, where n is the number of vertices specified by the application. If n is less than 3, OpenGL renders nothing.
 - Draws a single, convex polygon.
 - Vertices 1 through N define this polygon.
 - A polygon is convex if all points on the line segment between any two points in the polygon or at the boundary of the polygon lie inside the polygon.
- Example :

```
glBegin(GL_POLYGON);
glVertex2f(x1, y1);
glVertex2f(x2, y2);
glVertex2f(x3, y3);
glVertex2f(x4, y4);
glVertex2f(x5, y5);
glEnd();
```

2.2.2 Attributes

- Properties that describe how an object should be rendered are called **attributes**.
- Available attributes depend on the type of object. For example, a line could be black or green. It could be solid or dashed. A polygon could be filled with a single color or with a pattern. We could display it as filled or only by its edges.
- Fig. 2.2.3 shows various attributes for lines and polygons.
- Some of the text attributes include the direction of the text string, the path followed by successive characters in the string, the height and width of the characters, the font, and the style (bold, italic, underlined).

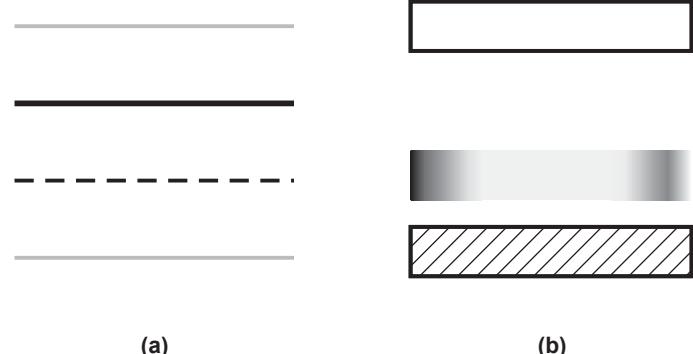


Fig. 2.2.3 Attributes for (a) Lines and (b) Polygons

2.2.2.1 Color Attribute

Colour can be viewed as a point in a colour solid, as shown in Fig. 2.2.4. As shown in Fig. 2.2.4, the solid is drawn using a coordinate system corresponding to the three primaries. The distance along a coordinate axis represents the amount of the corresponding primary in the colour.

The maximum value of each primary is normalized to 1. Any colour can be represented with this set of primaries as a point in a unit cube. The vertices of the cube correspond to black (no primaries on); red, green, and blue (one primary fully on); the pairs of primaries, cyan (green and blue fully on), magenta (red and blue fully on), and yellow (red and green fully on); and white (all primaries fully on). The principal diagonal of the cube connects the origin (black) with white. All colours along this line have equal tristimulus values and appear as shades of gray.

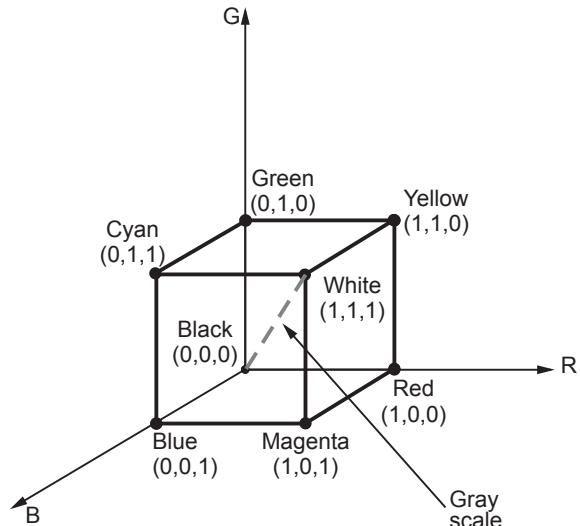


Fig. 2.2.4 Colour solid

RGB Colour

- The red, green and blue (RGB) colour model used in colour CRT monitors and colour raster graphics employs a Cartesian co-ordinate system. In this model, the individual contribution of red, green and blue are added together to get the resultant colour.
- We can represent this colour model with the unit cube defined on R, G and B axes, as shown in the Fig. 2.2.4.
- The vertex of the cube on the axes represent the primary colours and the remaining vertices represent the complementary colour for each of the primary colours. The main diagonal of the cube, with equal amounts of each primary represents the gray levels. The end at the origin of the diagonal represents black (0, 0, 0) and other end represents white (1, 1, 1).
- Each colour point within the bounds of the cube is represented as the triple (R, G, B), where value for R, G, B are assigned in the range from 0 to 1. As mentioned earlier, it is an additive model. Intensities of the primary colours are added to get the resultant colour. Thus, the resultant colour C_λ is expressed in RGB component as,

$$C_\lambda = RR + GG + BB$$

- The RGB colour model is used in image capture devices, televisions and colour monitors. This model is additive in which different intensities of red, green and blue are added to generate various colours. However, this model is not suitable for image processing.
- In OpenGL, we use the colour cube as follows. To set a colour for drawing in blue, we have to issue following function call.

```
glColor3f(0.0, 0.0, 1.0);
```

- In four-dimensional colour scheme to clear an area of the screen (a drawing window) we use following function call.

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

- Since RGB options are 1.0, 1.0, 1.0 clear colour is white.
- The fourth component in the function call is alpha. It specifies the opacity value. When alpha = 1.0, it is opaque and alpha = 0.0 it is transparent.

Indexed Colour

- It is a fact that any picture has finite number of colours. Thus by storing limited number colour information in the **colour look-up table** it is possible to manage digital images. This techniques is known as **indexed colouring**.

- This technique is used in order to save computer memory and file storage, while speeding up display refresh and file transfers. It is a form of **vector quantization compression**.
- When an image is encoded in this way, colour information is not directly carried by the image pixel data, but is stored in a separate piece of data called a **palette**: an array of colour elements, in which every element, a colour, is indexed by its position within the array. The image pixels do not contain the full specification of its colour, but only its index in the palette. Thus this technique is also referred as **pseudocolour** or **indirect colour**, as colours are addressed indirectly.
- Suppose that our frame buffer has k bits per pixel. Each pixel value or index is an integer between 0 and $2^k - 1$. Suppose that we can display each color component with a precision of n bits; that is, we can choose from 2^n reds, 2^n greens and 2^n blues. Hence, we can produce any of 2^{3n} colours on the display, but the frame buffer can specify only 2^k of them.
- In OpenGL, GLUT allows us to set the entries in a colour table for each window through the following function call.

```
glutSetColor (int color, GLfloat red, GLfloat green, GLfloat blue);
```

- If we are in colour-index mode, the present colour is selected by following function call.

```
glIndexi (element) ;
```

Index	Red	Green	Blue
0
1
2
.
.
.
.
$2^K - 1$
-----n-bits----- -----n-bits----- -----n-bits-----			

Table 2.2.1 Colour look-up table

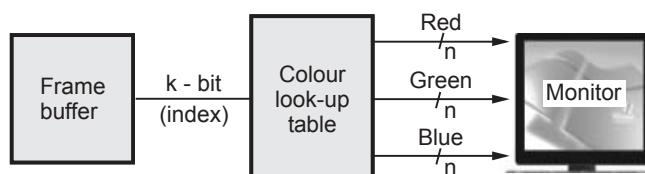


Fig. 2.2.5 Indexed colour

2.2.2.2 OpenGL Point Attributes

- Besides changing the color of a point, we can also specify the point size. We set the size for an OpenGL point with:

```
glPointSize (size);
```

- Parameter size is assigned a positive floating point value, which is rounded to an integer (unless the point is to be anti-aliased). A point size of 1.0 (the default value) displays a single pixel, and a point size of 2.0 displays a 2 by 2 pixel array.

2.2.2.3 OpenGL Line Attributes

- Similar to the point size, the line width can be changed using the following OpenGL function:

```
glLineWidth (width);
```

- We assign a floating-point value to parameter width, and this value is rounded to the nearest nonnegative integer. If the input value rounds to 0.0, the line is displayed with the standard (default) width of 1.0. When using anti-aliasing, fractional widths are possible as well.
- By default, a straight-line segment is displayed as a solid line. But we can also display dashed lines, dotted lines, or a line with a combination of dashed and dots.
- We set a current display style for lines with this OpenGL function:

```
glLineStipple (repeatFactor, pattern);
```

- Parameter pattern is used to reference a 16-bit integer that describes how the line should be displayed. A 1 in the bit-pattern denotes an "on" pixel position. The pattern is applied to the pixels along the line path starting with the low-order bits in the pattern. The default pattern is 0xFFFF which produces a solid line.
- As an example of specifying a line style, the following function call results in dashed lines :

```
glLineStipple (1, 0x00FF);
```

- The first half of this pattern (eight pixels) switches those pixels off, while the second half results in visible pixels. Also, since low-order bits are applied first, a line begins with an eight-pixel dash starting at the first endpoint. This dash is followed by an eight-pixel space, then another eight-pixel dash, and so forth, until the second endpoint position is reached.
- Integer parameter **repeatFactor** specifies how many times each bit in the pattern is to be repeated before the next bit in the pattern is applied. The default repeat value is 1.
- Before a line can be displayed in the current line-style pattern, we must activate the line-style feature of OpenGL. We accomplish this with the following function:

```
glEnable(GL_LINE_STIPPLE);
```

- Without enabling this feature, lines would still appear as solid lines, even though a pattern was provided.
- To disable the use of a pattern we can issue the following function call:

```
glDisable(GL_LINE_STIPPLE);
```

2.2.4 OpenGL Fill Attributes

- By default, a polygon is displayed as a solid-color region, using the current color setting.
- To fill the polygon with a pattern in OpenGL, a 32-bit by 32-bit bit mask has to be specified similar to defining a line-style:

```
GLubyte pattern [] = { 0xff, 0x00, 0xff, 0x00, ...};
```

- Once we have set the mask, we can establish it as the current fill pattern:

```
glPolygonStipple(pattern);
```

- We need to enable the fill routines before we specify the vertices for the polygons that are to be filled with the current pattern. We do this with the statement :

```
glEnable(GL_POLYGON_STIPPLE);
```

- Similarly, we turn off the pattern filling with :

```
glDisable(GL_POLYGON_STIPPLE);
```

2.2.5 OpenGL Charater Attributes

- We have two methods for displaying characters with the OpenGL. Either we can design a font set using the bitmap functions in the core library, or we can invoke the GLUT character-generation routines.
- The GLUT library contains functions for displaying predefined bitmap and stroke character sets. Therefore, the character attributes we can set are those that apply to either bitmaps or line segments.
- For either bitmap or outline fonts, the display color is determined by the current color state.
- In general, the spacing and size of characters is determined by the font designation, such as GLUT BITMAP 9 BY 15 and GLUT STROKE MONO ROMAN. However, we can also set the line width and line type for the outline fonts.
- We specify the width for a line with the glLineWidth function, and we select a line type with the glLineStipple function. The GLUT stroke fonts will then be displayed using the current values we specified for the OpenGL line-width and line-type attributes.

Review Questions

1. Explain the different polygons in OpenGL.
2. List out different OpenGL primitives, giving examples for each.
3. What is an attribute with respect to graphics system ? List attributes for lines and polygons.
4. With a neat diagram, discuss the colour formation. Explain the additive and subtractive colours, indexed colour and colour solid concept.
5. Write explanatory notes on : i) RGB colour model ii) Indexed colour model.
6. List the character attributes.

2.3 Simple Modelling and Rendering of 2D and 3D Geometric Objects

- Two principal tasks are required to create an image of a two or three-dimensional scene: **modeling** and **rendering**.
- The modeling task generates a model, which is the description of an object that is going to be used by the graphics system. Models must be created for every object in a scene; they should accurately capture the geometric shape and appearance of the object.
- The second task, rendering, takes models as input and generates pixel values for the final image.
- We have already seen that OpenGL supports a handful of primitive types for modeling two-dimensional (2D) and three-dimensional (3D) objects: points, lines, triangles, quadrilaterals, and (convex) polygons.
- In addition, OpenGL includes support for rendering higher-order surface patches using evaluators. A simple object, such as a box, can be represented using a polygon for each face in the object.
- Part of the modeling task consists of determining the 3D coordinates of each vertex in each polygon that makes up a model.
- To provide accurate rendering of a model's appearance or surface shading, the modeler may also have to determine color values, shading normals, and texture coordinates for the model's vertices and faces.
- Complex objects with curved surfaces can also be modeled using polygons. A curved surface is represented by a gridwork or mesh of polygons in which each polygon vertex is placed on a location on the surface.

2.3.1 Primitive Functions Examples

Program 2.3.1 : Draw the basic primitives using OpenGL.

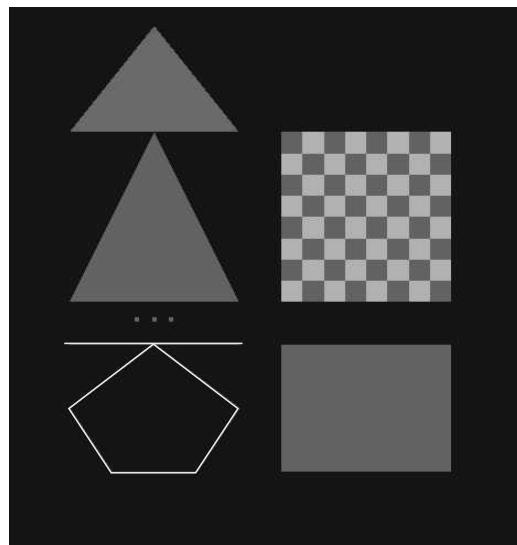
```
#include <GL/glut.h>
void drawChessBoard(int left, int top, int size)
{
    int length;
    GLfloat col = 0.0;
    length = size/8.0;
    for(int i=0; i < 8; i++)
    {
        for(int j=0; j < 8; j++)
        {
            glColor3f(1.0, col, 0.0);
            glRecti(left + length * j, top - length *i, left + length * j+length,
                     top - length *i-length);
            if(col == 0.0)
            {
                col = 1.0;
            }
            else
            {
                col = 0.0;
            }
        }
    }
    if(col == 0.0)
    {
        col = 1.0;
    }
    else
    {
        col = 0.0;
    }
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
```

```
glBegin(GL_POINTS);           // Draw Points
    glVertex2i(180,280);
    glVertex2i(200,280);
    glVertex2i(220,280);
glEnd();
glBegin(GL_LINES);           // Draw Line
    glVertex2i(100, 250);
    glVertex2i(300, 250);
glEnd();
glRecti(350, 250, 550, 100); // Draw Rectangle
glBegin(GL_TRIANGLES);        // Draw Triangle
    glVertex2i(100, 300);
    glVertex2i(300, 300);
    glVertex2i(200, 500);
glEnd();
glBegin(GL_LINE_LOOP);        // Draw Polygon
    glVertex2i(100, 175);
    glVertex2i(150, 100);
    glVertex2i(250, 100);
    glVertex2i(300, 175);
    glVertex2i(200, 250);
glEnd();
drawChessBoard(350,500,200);
glBegin(GL_POLYGON);
    glColor3f(1, 0, 0); glVertex3i(100, 500, 0.5);
    glColor3f(0, 1, 0); glVertex3i(300, 500, 0);
    glColor3f(0, 0, 1); glVertex3i(200,625, 0);
glEnd();
glFlush(); // Send all output to display
}
void myinit()
{
    // Set a deep black background and draw in a red.
    glClearColor(0.0, 0.0, 0.0, 1.0);      // Set background as black
    glColor3f(1.0, 0.0, 0.0);              // Draw in Red
    glPointSize(4.0);                    // Set point size = 4 pixels
    glMatrixMode(GL_PROJECTION);         // Establish the coordinate system
    glLoadIdentity();
```

```
    gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}

void myKeyboard(unsigned char key, int mouseX, int mouseY )
{
    switch (key) {
        case 27:
            exit(0);
    }
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);                                // Initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // Set display mode
    glutInitWindowSize(500, 500);                         // Set window size
    glutInitWindowPosition(100,100); // Set window position on the screen
    // Open the screen window
    glutCreateWindow("Draw Graphics Primitives on Keypress");
    glutDisplayFunc(display);                            // Register redraw function
    glutKeyboardFunc(myKeyboard);                      // Register keyboard function
    myinit();
    glutMainLoop();                                     // go into a perpetual loop
    return 0;
}
```

Output

Program 2.3.2 : Write a program in OpenGL

to display the following

Fig. 2.3.1 on a raster display

system. Assume suitable

coordinates for the vertices.

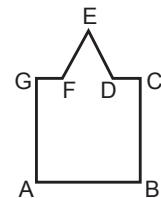


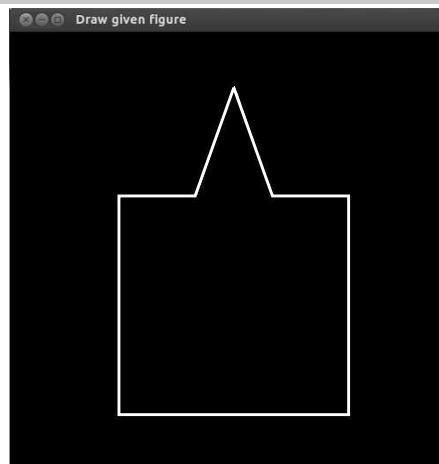
Fig. 2.3.1

```
#include <GL/glut.h>
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_LINE_LOOP); // Draw Polygon
        glVertex2i(100, 50); // Point A
        glVertex2i(310, 50); // Point B
        glVertex2i(310, 250); // Point C
        glVertex2i(240, 250); // Point D
        glVertex2i(205, 350); // Point E
        glVertex2i(170, 250); // Point F
        glVertex2i(100, 250); // Point G
    glEnd();
    glFlush(); // send all output to display
}
void myinit()
{
    // Set a deep black background and draw in a White.
    glClearColor(0.0, 0.0, 0.0, 1.0); // Set background as black
    glColor3f(1.0, 1.0, 1.0); // Draw in White
    glMatrixMode(GL_PROJECTION); // Establish the coordinate system
    glLoadIdentity();
    gluOrtho2D(0.0, 400.0, 0.0, 400.0);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv); // Initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB); // Set display mode
    glutInitWindowSize(500, 500); // Set window size
```

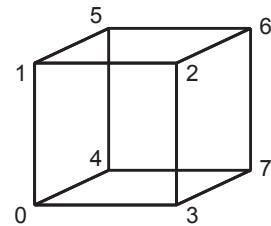
```

glutInitWindowPosition(100,100);           // Set window position on the screen
// Open the screen window
glutCreateWindow("Draw given figure");
glutDisplayFunc(display);                  // Register redraw function
myinit();
glutMainLoop();                          // go into a perpetual loop
return 0;
}

```

Output**2.3.2 Modelling a Colored Cube**

- A cube is a simple 3D object.
- Here, we use surface based model. It represents cube either as the intersection of six planes or as the six polygons, called **facets**. Facets define the faces of cube.
- The following code statement initializes an array with the vertices of a cube.

**Fig. 2.3.2**

```

GLfloat vertices [8] [3] = { { - 1.0, - 1.0, - 1.0},
                             { 1.0, - 1.0, - 1.0},
                             {1.0, 1.0, - 1.0},
                             { - 1.0, 1.0, - 1.0},
                             { - 1.0, - 1.0, 1.0},
                             { 1.0, - 1.0, 1.0},
                             { 1.0, 1.0, 1.0},
                             { - 1.0, 1.0, 1.0 } }
};

```

- Now we can specify the faces of the cube. An example, code to specify one face is as given below :

```
glBegin (GL_POLYGON) ;
    glVertex3fv (vertices [0]) ;
    glVertex3fv (vertices [3]) ;
    glVertex3fv (vertices [2]) ;
    glVertex3fv (vertices [1]) ;
glEnd () ;
```

- Similarly the other five faces can be defined. The other five faces are - (2, 3, 7, 6), (0, 4, 7, 3), (1, 2, 6, 5), (4, 5, 6, 7) and (0, 1, 5, 4). Here, the order of traversal of the edges of cube faces is determined by **right hand rule**. This is illustrated in Fig. 2.3.3.

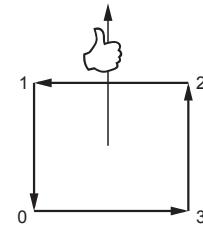


Fig. 2.3.3 Traversal of the edges of a polygon

Review Questions

- What is modeling ?
- What is rendering ?
- Write a short note on modeling and rendering.

2.4 Interaction

- Definition :** Interaction in the field of computer graphics refers to the process of enabling the users to interact with computer displays. The user can observe the image change in response to the input from the user and act accordingly.
- Support by OpenGL for Interaction OpenGL does not support interaction directly. The main reason for this is to make OpenGL portable (i.e., work on all types of systems irrespective of the hardware).
- However, OpenGL provides the GLUT tool kit which supports minimum functionality such as opening of windows, use of keyboard, mouse and creation of pop-up menus, etc.

2.4.1 Events

- The OpenGL supports basic window events : the window has been resized, the window needs to draw its contents, a key on the keyboard has been pressed or released, a mouse has moved or mouse button was pressed or released. All such event processing is based on callbacks. It is necessary to write functions to process

specific events and set them as callbacks for a specific window. When event occurs, a corresponding callback function is called.

- GLUT handles events with **callback functions**. If you want to handle an event, such as a keystroke or mouse movement, you write a function that performs the desired action. Then you **register** your function by passing its name as an argument to a function with a name of the form `glut...Func()`, in which "..." indicates which callback you are registering.

2.4.2 Keyboard Callback Functions

- If the program has registered a keyboard callback function, the keyboard callback function is called whenever the user presses a key.
1. Register a keyboard callback function.

```
glutKeyboardFunc(keyboard);
```

2. A keyboard callback function. It receives three arguments : The key that was pressed, and the current X and Y coordinates of the mouse.
- The callback function registered by `glutKeyboardFunc()` recognizes only keys corresponding to ASCII graphic characters and esc, backspace and delete. The keyboard callback function in Fig. 2.4.1 is a useful default keyboard function : It allows the user to quit the program by pressing the escape key.

```
#define ESCAPE 27
void keyboard (unsigned char key, int x, int y)
{
    if (key == ESCAPE)
        exit(0);
}
```

Fig. 2.4.1 Quitting with the escape key

- To make your program respond to other keys, such as the arrow keys and the function ("F") keys :
1. Register a special key callback function.

```
glutSpecialFunc(special);
```

2. Declare the special key function as follows :

GLUT_KEY_F1	GLUT_KEY_F8	GLUT_KEY_UP
GLUT_KEY_F2	GLUT_KEY_F9	GLUT_KEY_DOWN
GLUT_KEY_F3	GLUT_KEY_F10	GLUT_KEY_PAGE_UP

GLUT_KEY_F4	GLUT_KEY_F11	GLUT_KEY_PAGE_DOWN
GLUT_KEY_F5	GLUT_KEY_F12	GLUT_KEY_HOME
GLUT_KEY_F6	GLUT_KEY_LEFT	GLUT_KEY_END
GLUT_KEY_F7	GLUT_KEY_RIGHT	GLUT_KEY_INSERT

Table 2.4.1 Constants for special keys

```
void special (int key, int x, int y)
{
switch (key)
{
case GLUT_KEY_F1:
    // code to handle F1 key
    break;
.....
}
}
```

- The arguments that OpenGL passes to special() are : the key code, as defined in Table 2.4.1, the X co-ordinate of the mouse and the Y co-ordinate of the mouse.

Program 2.4.1 : This program uses keyboard callback function. The keyboard callback function checks the keypress and accordingly display graphics primitive.

```
#include <GL/glut.h>
int Height=650, Width= 650;
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush(); // Send all output to display
}
void myinit()
{
    // Set a deep black background and draw in a red.
    glColor3f(0.0, 0.0, 0.0);           // Set background as black
    glColor3f(1.0, 0.0, 0.0);           // Draw in Red
    glMatrixMode(GL_PROJECTION);      // Establish the coordinate system
    glLoadIdentity();
```

```
    gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}

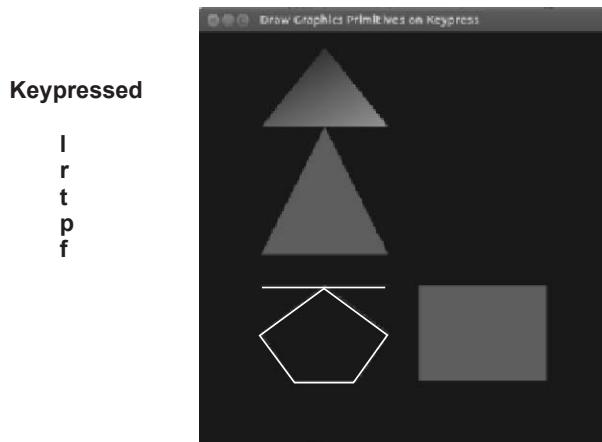
void myKeyboard(unsigned char key, int mouseX, int mouseY )
{
    switch(key) {
    case 'l':
        glBegin(GL_LINES);           // Draw Line
        glVertex2i(100, 250);
        glVertex2i(300, 250);
        glEnd();
        break;
    case 'r':
        glRecti(350, 250, 550, 100); // Draw rectangle
        break;
    case 't':
        glBegin(GL_TRIANGLES);      // Draw Triangle
        glVertex2i(100, 300);
        glVertex2i(300, 300);
        glVertex2i(200, 500);
        glEnd();
        break;
    case 'p':
        glBegin(GL_LINE_LOOP);     // Draw Polygon without fill
        glVertex2i(100, 175);
        glVertex2i(150, 100);
        glVertex2i(250, 100);
        glVertex2i(300, 175);
        glVertex2i(200, 250);
        glEnd();
        break;
    case 'f':
        glBegin(GL_POLYGON);       // Draw Polygon with fill
        glColor3f(1, 0, 0); glVertex3i(100, 500, 0.5);
        glColor3f(0, 1, 0); glVertex3i(300, 500, 0);
        glColor3f(0, 0, 1); glVertex3i(200, 625, 0);
        glEnd();
        break;
    }
```

```

case 27:
    exit(0);           // Terminate the program
}
glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);           // Initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);      // Set display mode
    glutInitWindowSize(500, 500);          // Set window size
    glutInitWindowPosition(100,100);        // Set window position on the screen
    // Open the screen window
    glutCreateWindow("Draw Graphics Primitives on Keypress");
    glutDisplayFunc(display);            // Register redraw function
    glutKeyboardFunc(myKeyboard);        // Register keyboard function
    myinit();
    glutMainLoop();                  // go into a perpetual loop
    return 0;
}

```

Output



2.4.3 Mouse Event Callback Functions

- There are several ways of using the mouse and two ways of responding to mouse activity. The first callback function responds to pressing or releasing one of the mouse buttons.
1. Register a mouse callback function.

```
glutMouseFunc(mouse_button);
```

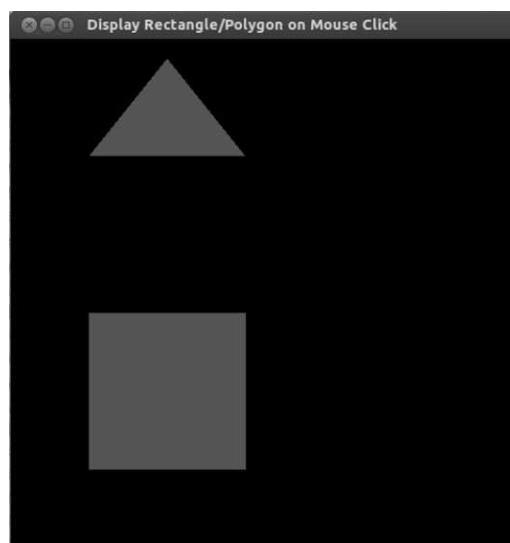
2. The mouse callback function is passed four arguments.
 - The first argument specifies the button. Its value is one of
 - GLUT_LEFT_BUTTON,
 - GLUT_MIDDLE_BUTTON, or
 - GLUT_RIGHT_BUTTON
 - The second argument specifies the button state. Its value is one of
 - GLUT_DOWN (the button has been pressed) or
 - GLUT_UP (the button has been released).
 - The remaining two arguments are the X and Y co-ordinates of the mouse.

Program 2.4.2 : This program displays rectangle on left mouse click and polygon on right mouse click.

```
#include <GL/glut.h>
int Height=650, Width= 650;
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush(); // Send all output to display
}
void myinit()
{
    // Set a deep black background and draw in a red.
    glClearColor(0.0, 0.0, 0.0, 1.0);      // Set background as black
    glColor3f(1.0, 0.0, 0.0);            // Draw in Red
    glMatrixMode(GL_PROJECTION); // Establish the coordinate system
    glLoadIdentity();
    gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}
void myMouse(int button, int state, int x, int y )
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        glRecti(100, 100, 300, 300); // Draw rectangle
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
```

```
{  
    glBegin(GL_POLYGON);  
        glVertex3i(100, 500, 0.5);  
        glVertex3i(300, 500, 0);  
        glVertex3i(200, 625, 0);  
    glEnd();  
}  
glFlush(); // Send output to display  
}  
  
int main(int argc, char **argv)  
{  
    glutInit(&argc, argv); // Initialize the toolkit  
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // Set display mode  
    glutInitWindowSize(500, 500); // Set window size  
    glutInitWindowPosition(100,100); // Set window position on the screen  
    // Open the screen window  
    glutCreateWindow("Display Rectangle/Polygon on Mouse Click");  
    glutDisplayFunc(display); // Register redraw function  
    glutMouseFunc(myMouse); // Register mouse function  
    myinit();  
    glutMainLoop(); // go into a perpetual loop  
    return 0;  
}
```

Output



Program 2.4.3 : Create a polyline using mouse interaction using OpenGL

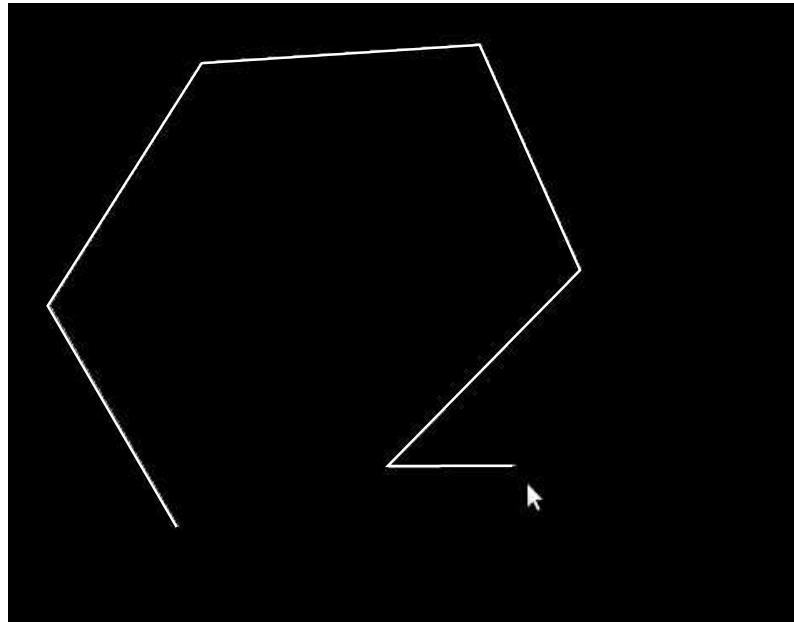
```
#include <GL/glut.h>
struct GLintPoint
{ GLint x,y;
};
int Height=650, Width= 650;
void myMouse(int button, int state, int x, int y );

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush(); // Send all output to display
}

void myinit() {
    glClearColor(0.0, 0.0, 0.0, 1.0);      // Set background as black
    glColor3f(1.0, 1.0, 0.0);            // Draw in Yellow
    glMatrixMode(GL_PROJECTION);        // Establish the coordinate system
    glLoadIdentity();
    gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}
void myKeyboard(unsigned char key, int mouseX, int mouseY )
{
    switch (key) {
    case 27:
        exit(0);
    }
}
void myMouse(int button, int state, int x, int y )
{
    static GLintPoint vertex [1];
    static int pt = 0;
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        if (pt == 0)
        {
            vertex [pt].x = x;
```

```
        vertex [pt].y = Height- y;
        pt++;
    }
else
{
    glBegin(GL_LINE_STRIP);
    glVertex2i(vertex [0].x, vertex[0].y);
    glVertex2i(x, Height- y);
    glEnd();
    vertex [0].x = x;
    vertex [0].y = Height- y;
}
glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);                                // Initialize the toolkit
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);           // Set display mode
    glutInitWindowSize(500, 500);                          // Set window size
    glutInitWindowPosition(100,100); // Set window position on the screen
    // Open the screen window
    glutCreateWindow("Draw Polyline using Interaction using OpenGL");
    glutDisplayFunc(display);                            // Register redraw function
    glutKeyboardFunc(myKeyboard);                      // Register keyboard function
    glutMouseFunc(myMouse);                           // Register mouse function
    myinit();
    glutMainLoop();                                    // go into a perpetual loop
    return 0;
}
```

Output**2.4.4 Window Event**

- Most window systems allow a user to resize the window interactively, usually by using the mouse to drag a corner of the window to a new location.
- Whenever the user moves or resizes the graphics window, OpenGL informs your program, provided that you have registered the appropriate **callback** function. The main problem with reshaping is that the user is likely to change the shape of the window as well as its size; you do not want the change to distort your image. Register a callback function that will respond to window reshaping :

```
glutReshapeFunc(reshape);
```

- The above function sets the reshape callback for the current window.
- The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created.

Reshape Callback Function

```
void reshape (GLsize w, GLsize h)
{
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity( );
```

```
gluOrtho2D (0.0, (GLdouble)w, 0.0, (GLdouble)h);
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ( );
glViewport (0, 0, w, h);
}
```

- The width and height parameters of the callback specify the new window size in pixels.
- We use these values to create a new OpenGL clipping window using gluOrtho2D, as well as a new viewport with the same aspect ratio.
- Another event such as a window movement without resizing are handled using following function call.

```
glutMotionFunc( );
```

This function sets the motion callback for the current window.

Review Questions

1. *What is interaction?*
2. *What is the necessity of programming event-driven input ? Describe window events and keyboard events.*
3. *Explain how an event driven input can be programmed for a keyboard device.*
4. *Explain how an event driven input can be performed for window events.*
5. *Explain how an event driven input can be programmed for a mouse device.*

2.5 Picking

- In science and engineering 3D visualization computer applications, it is useful for the user to point to something and have the program figure out what is being pointed to. This is called **picking**.
- A pick device is considerably more difficult to implement than the locator device. The reason for this is that the pick device has to identify the object on the display instead to identifying only position on the display.
- We know that, primitives defined in the application program have to go through sequence of geometric operation, rasterization and fragment operation on their way to the frame buffer. Although much of this process is reversible in mathematical sense; however, the hardware is not reversible. Hence, converting from a location on the display to the corresponding primitive is not a simple task.

- There are three ways to deal with this task -
 - Selection
 - Bounding boxes or extends.
 - Back buffer and an extra rendering.
- **Selection** process involves adjusting the clipping region and viewport such that we can keep track of which primitives in a small clipping region are rendered into a region near the cursor. These primitives go into a **hit list** that can be examined by the user program. The OpenGL uses this approach.
- The **bounding box** or extent of an object is the smallest rectangle, aligned with the coordinate axes that contains the object.
- Using bounding boxes it is possible determine the rectangle in screen co-ordinates that corresponds to a rectangle point in object or world co-ordinate.
- This is easy in 2D applications, but difficult in 3D applications.
- In case double buffering, two colour buffers are used : Front buffer and a back buffer. Since **back buffer** is not displayed, we can use it to store rendered objects each in a distinct colour. Thus it is possible to determine an object's contents by simply changing colours wherever a new object appears in the program.
- Steps involved in such picking process are as follows :
 - Draw the objects into the back buffer with different colours.
 - Get the position of mouse using callback function.
 - Find the colour at the position in the frame buffer corresponding to the mouse position using `glReadPixel()` function.
 - Identify the object using pixel colour.

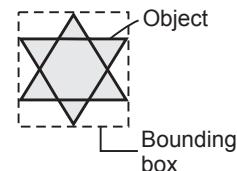


Fig. 2.5.1

2.5.1 Picking and Selection Mode

- OpenGL uses selection mode approach to implement picking.
- When we plan to use OpenGL's selection mode, we have to first draw our scene into the frame buffer, and then we enter selection mode and redraw the scene. However, once we are in selection mode, the contents of the frame buffer don't change until we exit selection mode.
- When we exit selection mode, OpenGL returns a list of the primitives that intersect the viewing volume (remember that the viewing volume is defined by the current model view and projection matrices and any additional clipping planes).

- Each primitive that intersects the viewing volume causes a selection **hit**. The list of primitives is actually returned as an array of integer-valued **names** and related data - the **hit records** - that correspond to the current contents of the **name stack**.
- We construct the name stack by loading names onto it as we issue primitive drawing commands while in selection mode. Thus, when the list of names is returned, we can use it to determine which primitives might have been selected on the screen by the user.
- In addition to this selection mode, OpenGL provides a utility routine designed to simplify selection in some cases by restricting drawing to a small region of the viewport. Typically, we use this routine to determine which objects are drawn near the cursor, so that we can identify which object the user is picking.

The Basic Steps

- To use the selection mode, we need to perform the following steps :
 1. Specify the array to be used for the returned hit records with **glSelectBuffer()**.
 2. Enter selection mode by specifying **GL_SELECT** with **glRenderMode()**.
 3. Initialize the name stack using **glInitNames()** and **glPushName()**.
 4. Define the viewing volume we want to use for selection. Usually this is different from the viewing volume we originally used to draw the scene, so we probably want to save and then restore the current transformation state with **glPushMatrix()** and **glPopMatrix()**.
 5. Alternately issue primitive drawing commands and commands to manipulate the name stack so that each primitive of interest has an appropriate name assigned.
 6. Exit selection mode and process the returned selection data (the hit records).

Using Selection Mode for Picking

- We can use selection mode to determine if objects are picked. To do this, we use a special picking matrix in conjunction with the projection matrix to restrict drawing to a small region of the viewport, typically near the cursor.
- Then we allow some form of input, such as clicking a mouse button, to initiate selection mode.
- With selection mode established and with the special picking matrix used, objects that are drawn near the cursor cause selection hits. Thus, during picking we are typically determining which objects are drawn near the cursor.
- Picking is set up almost exactly like regular selection mode is, with the following major differences.
 - Picking is usually triggered by an input device.

- We use the utility routine **gluPickMatrix()** to multiply a special picking matrix onto the current projection matrix. This routine should be called prior to multiplying a standard projection matrix (such as **gluPerspective()** or **glOrtho()**). We will probably want to save the contents of the projection matrix first, so the sequence of operations may look like this :

```
glMatrixMode (GL_PROJECTION);
glPushMatrix ();
glLoadIdentity ();
gluPickMatrix (...);
gluPerspective, glOrtho, gluOrtho2D, or glFrustum
/* ... draw scene for picking ; perform picking ... */
glPopMatrix();
```

Review Questions

1. *What is picking ?*
2. *Explain the steps to be performed in the selection mode.*
3. *Describe logical input operation of picking in selection mode.*
4. *List down the steps involved in the picking process.*



Notes

UNIT - I

3

Scan Conversion

Syllabus

Line drawing algorithms : Digital Differential Analyzer (DDA), Bresenham.

Circle drawing algorithms : DDA, Bresenham, and Midpoint.

Contents

3.1 Introduction	Dec.-07,11,	Marks 2
3.2 Line Drawing Algorithms	June-12,May-05,10,11,13, 14, 15, 16,17,18,19, Dec.-06,07,08,10,11,12,14, 15,18,19,	Marks 10
3.3 Antialiasing and Antialiasing Techniques	Dec.-10,14, May-13,	Marks 4
3.4 Basic Concepts in Circle Drawing		
3.5 Circle Drawing Algorithms.....	May-05,06,07,13,14,16,18, Dec.-07,12,15,17	Marks 12

3.1 Introduction

SPPU : Dec.-07,11

- In a raster graphics display system, a picture is completely specified by the set of intensities for the pixel positions in the display.
- We can also describe a picture as a set of complex objects such as trees and furniture and walls, positioned at specified co-ordinate locations within the scene.
- Shapes and colours of the objects can be described internally with pixel arrays or with set of basic geometric structures, such as straight line segments and polygon colour areas.
- Typically, graphics programming packages provide functions to describe a scene in-terms of these basic geometric structures referred to as **output primitives**.
- Point and line segments are the basic output primitives, whereas circles, conic sections, quadric surfaces, polygons, spline curves are the other output primitives.

3.1.1 Lines

- Point is the fundamental element of the picture representation. It is nothing but the position in a plane defined as either pairs or triplets of numbers depending on whether the data are two or three dimensional. Thus, (x_1, y_1) or (x_1, y_1, z_1) would represent a point in either two or three dimensional space.
- Two points would represent a line or edge, and a collection of three or more points a polygon.
- The representation of curved lines is usually accomplished by approximating them by short straight line segments.
- If the two points used to specify a line are (x_1, y_1) and (x_2, y_2) , then an equation for the line is given as

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1} \quad \dots (3.1.1)$$

$$\therefore y = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) + y_1 \quad \dots (3.1.2)$$

$$\text{or } y = mx + b \quad \dots (3.1.3)$$

$$\text{where } m = \frac{y_2 - y_1}{x_2 - x_1} \quad \dots (3.1.4)$$

$$\text{and } b = y_1 - mx_1 \quad \dots (3.1.5)$$

The above equation is called the **slope intercept form** of the line.

- The **slope** m is the change in height ($y_2 - y_1$) divided by the change in width ($x_2 - x_1$) for two points on the line.
- The **intercept** b is the height at which the line crosses the y -axis.
- Another form of the line equation, called the general form, may be found by multiplying out the factors and rearranging the equation (3.1.1).

$$(y_2 - y_1)x - (x_2 - x_1)y + x_2y_1 - x_1y_2 = 0 \quad \dots (3.1.6)$$

$$\text{or } rx + sy + t = 0 \quad \dots (3.1.7)$$

where $r = (y_2 - y_1)$, $s = -(x_2 - x_1)$ and $t = (x_2y_1 - x_1y_2)$

- The values given for r , s and t are called possible values of them because we can see that multiplying r , s and t by any common factor will produce a new set of r' , s' and t' values which will also satisfy equation (3.1.7) and, therefore, also describe the same line. Usually values for r , s and t are taken so that

$$r^2 + s^2 = 1 \quad \dots (3.1.8)$$

Writing equation (3.1.7) in the form of equation (3.1.2) we have,

$$y = \frac{-rx}{s} - \frac{t}{s}$$

$$\therefore \text{We have, } m = \frac{-r}{s} \text{ and } b = -\frac{t}{s}$$

- We can determine whether two lines are crossing or parallel.
- When two lines cross, they share some point in common. Of course, that point satisfies equations for the two lines. Let us see how to obtain this point.
- Consider two lines having slopes m_1 and m_2 and y -intercepts b_1 and b_2 respectively. Then we can write the equations for these two lines as,

$$\text{Line 1 : } y = m_1x + b_1 \quad \dots (3.1.9)$$

$$\text{Line 2 : } y = m_2x + b_2 \quad \dots (3.1.10)$$

- If point (x_p, y_p) is shared by both lines, then it should satisfy equations (3.1.9) and (3.1.10).

$$\therefore y_p = m_1 x_p + b_1 \quad \dots (3.1.11)$$

$$\text{and } y_p = m_2 x_p + b_2 \quad \dots (3.1.12)$$

Equating the above equations gives,

$$m_1 x_p + b_1 = m_2 x_p + b_2$$

$$\therefore x_p (m_1 - m_2) = b_2 - b_1$$

$$\therefore x_p = \frac{b_2 - b_1}{m_1 - m_2} \quad \dots (3.1.13)$$

Substituting this into equation for line1, (3.1.11) (or for line2, (3.1.12)) gives,

$$\begin{aligned} y_p &= m_1 \left(\frac{b_2 - b_1}{m_1 - m_2} \right) + b_1 \\ \therefore y_p &= \frac{m_1(b_2 - b_1) + b_1(m_1 - m_2)}{m_1 - m_2} \\ \therefore y_p &= \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \end{aligned} \quad \dots (3.1.14)$$

Therefore, the point $\left(\frac{b_2 - b_1}{m_1 - m_2}, \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \right)$... (3.1.15)

is the point shared by both the lines, i.e. intersection point.

- If the two lines are parallel, they have the same slope. In the expression (3.1.15) for the point of intersection, $m_1 = m_2$. So the denominator becomes zero. So the expression results in a divide by zero. This means, there is no point of intersection for the parallel lines.
- If we consider the expression for line given by equation (3.1.7), then by the similar mathematical calculations, we will get the intersecting point,

$$\left(\frac{s_1 t_2 - s_2 t_1}{s_2 r_1 - s_1 r_2}, \frac{t_1 r_2 - t_2 r_1}{s_2 r_1 - s_1 r_2} \right) \quad \dots (3.1.16)$$

3.1.2 Lines Segments

- When we say the line, it extends forever both forward and backward.
- Any point in a given direction satisfies equation of the line.
- In graphics, we need to display only pieces of lines. When we consider the piece of line, i.e. only those points which lie between two endpoints, then this is called a line segment. Fig. 3.1.1 shows an example of a line segment, where P_1 and P_2 are the endpoints.
- From the endpoints of a line segment the equation of the line can be obtained. Let the endpoints be $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, and equation of the line, $y = mx + b$. Whether any point lies on the line segment or not, can be determined from the equation of the line and the end-points.
- If any another point $P_i(x_i, y_i)$, is considered, then it lies on the segment if the following conditions are satisfied.
 1. $y_i = mx_i + b$
 2. $\min(x_1, x_2) \leq x_i \leq \max(x_1, x_2)$
 3. $\min(y_1, y_2) \leq y_i \leq \max(y_1, y_2)$



Fig. 3.1.1 A line segment

Note Here $\min(x_1, x_2)$ means the smallest of x_1 and x_2 , and $\max(x_1, x_2)$ means the largest of x_1 and x_2 .

Parametric form

- The equations which give the x and y values on the line in terms of a parameter, u are called 'parametric form' of the line equation.
- This form is useful for constructing line segments.
- For example, we want line segment between (x_1, y_1) and (x_2, y_2) . If the parameter is u and if we want the x co-ordinate to go uniformly from x_1 to x_2 , then the expression for x can be written as,

$$x = x_1 + (x_2 - x_1)u \quad \dots (3.1.17)$$

- The line segment points correspond to value of the parameter between 0 and 1. So when u is 0, x is x_1 . As u approaches to 1, x changes uniformly to x_2 . For a line segment, when x changes from x_1 to x_2 , at the same time, y must change from y_1 to y_2 uniformly.

$$\text{i.e. } y = y_1 + (y_2 - y_1)u \quad \dots (3.1.18)$$

- The two equations (3.1.17) and (3.1.18) together describe a straight line. This can be shown by equating over u. If the parameter value for a point on the line is given, we can easily test if the point lies on the line segment.

Length of a line segment

- If the two endpoints of a line segment are known, we can obtain its length.
- Consider the two endpoints of a line segment as $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ and L is the length of a line segment. Construct a right triangle $P_1 P_2 A$ as shown in Fig. 3.1.2.
- The Pythagorean theorem states that the square of the length of the hypotenuse is equal to the sum of the squares of the lengths of the two adjacent sides. Here, hypotenuse is $P_1 P_2$ and two adjacent sides are $P_1 A$ and $P_2 A$.
- The co-ordinates of P_1 , P_2 and A are (x_1, y_1) , (x_2, y_2) and (x_2, y_1) respectively. Then the length of the segment $P_1 P_2$, L is given by,

$$L^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2 \quad \dots (3.1.9 \text{ (a)})$$

$$\therefore L = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2} \quad \dots (3.1.19 \text{ (b)})$$

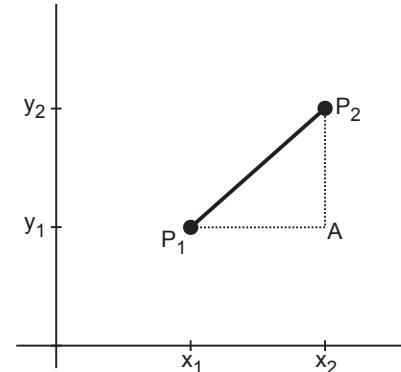


Fig. 3.1.2 Right triangle $P_1 P_2 A$

Midpoint of a line segment

To obtain the mid-point of a line segment, take the x co-ordinate halfway between the x co-ordinates of the endpoints and y co-ordinate halfway between the y co-ordinates of the endpoints. Thus, the mid-point is,

$$(x_m, y_m) = \left[\frac{(x_1 + x_2)}{2}, \frac{(y_1 + y_2)}{2} \right] \quad \dots (3.1.20)$$

The mid-point of a line segment can be calculated easily. It is oftenly useful.

3.1.3 Vectors

- A vector has a single direction and a length.
- A vector may be indicated by $[D_x, D_y]$ where D_x denotes the distance on x-axis direction and D_y denote the distance on y-axis direction, as shown in Fig. 3.1.3.
- When we consider a line segment, it has a fixed position in space. But vector does not have a fixed position in space.
- The vector does not tell us the starting point. It gives how far to move and in which direction.

Vector notation

- The vectors may be denoted by a shorthand way. The operations are performed on all co-ordinates.
- For example, the vector form of parametric equations for a line can be written as,

$$V = V_1 + u(V_2 - V_1) \quad \dots (3.1.21)$$

Addition of vectors

- The addition of vectors is nothing but the addition of their respective components. If the two vectors are $V_1 [D_{x1}, D_{y1}]$ and $V_2 [D_{x2}, D_{y2}]$, then the addition of V_1 and V_2 is given by,

$$\begin{aligned} V_1 + V_2 &= [D_{x1}, D_{y1}] + [D_{x2}, D_{y2}] \\ &= [D_{x1} + D_{x2}, D_{y1} + D_{y2}] \end{aligned} \quad \dots (3.1.22)$$

- We can visualize the vector addition by considering a point A. The first vector V_1 moves from point A to point B, the second vector V_2 moves from point B to point C. Then the addition of V_1 and V_2 is nothing but the vector from point A to point C.

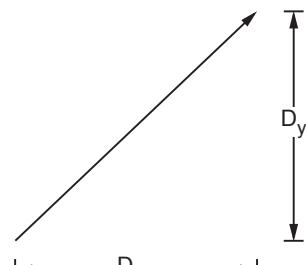


Fig. 3.1.3 A vector

Multiplication of vectors

- The vector can be multiplied by a number. The multiplication is performed by multiplying each component of a vector by a number.

Let $V(D_x, D_y)$ be a vector which is to be multiplied by a number n . Then,

$$nV = n [D_x, D_y] = [n D_x, n D_y] \quad \dots (3.1.23)$$

- The magnitude of the resultant vector is given by,

$$|V| = (D_x^2 + D_y^2)^{1/2} \quad \dots (3.1.24)$$

- The result of the multiplication changes the magnitude of a vector but preserves its direction.

Unit vectors

- From equation (3.1.22) and (3.1.23) we can say that the multiplication of a vector and the reciprocal of its length is equal to 1. Such vectors are called **unit vectors**. They conveniently capture the direction information.

Review Question

1. What do you mean by output primitives ?

2. Define line.

SPPU : Dec.-07,11, Marks 2

3. What is a line segment.

4. Define vectors.

SPPU : Dec.-07,11, Marks 2

3.2 Line Drawing Algorithms SPPU : June-12, May-05,10,11,13,14,15,16,17,18,19 Dec.-06,07,08,10,11,12,14, 15,18,19

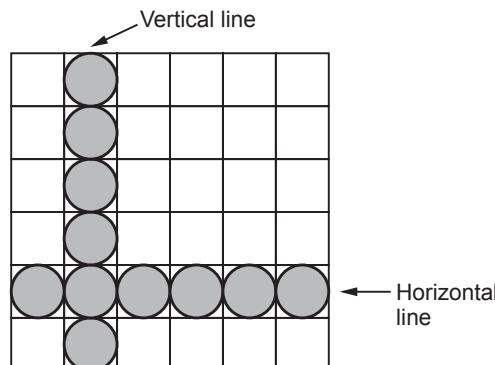
- The process of 'turning on' the pixels for a line segment is called **vector generation** or **line generation** and the algorithms for them are known as **vector generation algorithms** or **line drawing algorithms**.

3.2.1 Qualities of Good Line Drawing Algorithm

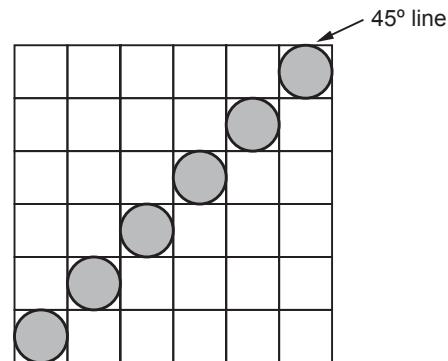
- Before discussing specific line drawing algorithms it is useful to note the general requirements for such algorithms. These requirements specify the desired characteristics of line.
 - The line should appear as a straight line and it should start and end accurately.
 - The line should be displayed with constant brightness along its length independent of its length and orientation.
 - The line should be drawn rapidly.

Let us see the different lines drawn in Fig. 3.2.1.

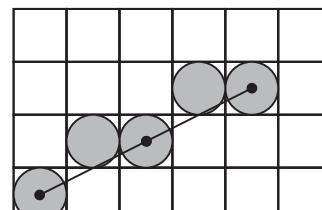
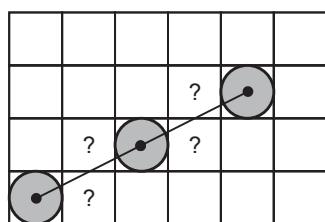
- As shown in Fig. 3.2.1 (a), horizontal and vertical lines are straight and have same width.
- The 45° line is straight but its width is not constant.
- The line with any other orientation is neither straight nor has same width. Such cases are due to the finite resolution of display and we have to accept approximate pixels in such situations, shown in Fig. 3.2.1 (c).



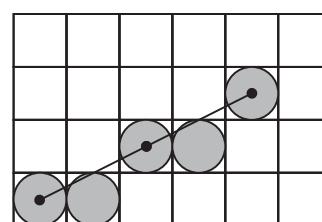
(a) Vertical and Horizontal lines



(b) 45° line



OR



(c) Line with other orientation

Fig. 3.2.1

- The brightness of the line is dependent on the orientation of the line.
- We can observe that the effective spacing between pixels for the 45° line is greater than for the vertical and horizontal lines. This will make the vertical and horizontal lines appear brighter than the 45° line.

- Complex calculations are required to provide equal brightness along lines of varying length and orientation. Therefore, to draw line rapidly some compromises are made such as
 - Calculate only an approximate line length
 - Reduce the calculations using simple integer arithmetic
 - Implement the result in hardware or firmware

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

Incremental Algorithm

1. CurrPosition = Start
Step = Increment
2. if ($| \text{CurrPosition} - \text{End} | < \text{Accuracy}$) then go to step 5
[This checks whether the current position is reached upto approximate end point. If yes, line drawing is completed.]
if ($\text{CurrPosition} < \text{End}$) then go to step 3
[Here start < End]
if ($\text{CurrPosition} > \text{End}$) then go to step 4
[Here start > End]
3. CurrPosition = CurrPosition + Step
go to step 2
4. CurrPosition = CurrPosition - Step
go to step 2
5. Stop.

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

3.2.2 Vector Generation/Digital Differential Analyzer (DDA) Algorithm

- The vector generation algorithms which step along the line to determine the pixel which should be turned on are sometimes called **digital differential analyzer (DDA)**.
- The slope of a straight line is given as

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \quad \dots (3.2.1)$$

The above differential equation can be used to obtain a rasterized straight line. For any given x interval Δx along a line, we can compute the corresponding y interval Δy from equation (3.2.1) as

$$\Delta y = \frac{y_2 - y_1}{x_2 - x_1} \Delta x \quad \dots (3.2.2)$$

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

$$\Delta x = \frac{x_2 - x_1}{y_2 - y_1} \Delta y \quad \dots (3.2.3)$$

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

$$\begin{aligned} x_{i+1} &= x_i + \Delta x \\ &= x_i + \frac{x_2 - x_1}{y_2 - y_1} \Delta y \end{aligned} \quad \dots (3.2.4)$$

and $y_{i+1} = y_i + \Delta y$

$$= y_i + \frac{y_2 - y_1}{x_2 - x_1} \Delta x \quad \dots (3.2.5)$$

- The equations (3.2.4) and (3.2.5) represent a recursion relation for successive values of x and y along the required line. Such a way of rasterizing a line is called a **digital differential analyzer** (DDA). For simple DDA either Δx or Δy , whichever is larger, is chosen as one raster unit, i.e.

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

else $\Delta x = 1$
 else $\Delta y = 1$

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

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

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

If $\Delta y = 1$ then

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

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

Let us see the vector generation/digital differential analyzer (DDA) routine for rasterizing a line.

Vector Generation/DDA Line Algorithm

1. Read the line end points (x_1, y_1) and (x_2, y_2) such that they are not equal.
[if equal then plot that point and exit]
2. $\Delta x = |x_2 - x_1|$ and $\Delta y = |y_2 - y_1|$
3. if $(\Delta x \geq \Delta y)$ then
 length = Δx
else
 length = Δy
end if
4. $\Delta x = (x_2 - x_1) / \text{length}$
 $\Delta y = (y_2 - y_1) / \text{length}$
[This makes either Δx or Δy equal to 1 because length is either $|x_2 - x_1|$ or $|y_2 - y_1|$. Therefore, the incremental value for either x or y is one.]
5. $x = x_1 + 0.5 * \text{Sign}(\Delta x)$
 $y = y_1 + 0.5 * \text{Sign}(\Delta y)$

Here, Sign function makes the algorithm work in all quadrant. It returns $-1, 0, 1$ depending on whether its argument is $< 0, = 0, > 0$ respectively. The factor 0.5 makes it possible to round the values in the integer function rather than truncating them.

- ```

 plot (Integer (x), Integer (y))
6. i = 1
 [Begins the loop, in this loop points are plotted]
 While (i ≤ length)
 {
 x = x + Δx
 y = y + Δy
 plot (Integer (x), Integer (y))
 i = i + 1
 }
7. Stop

```

Let us see few examples to illustrate this algorithm.

**Example 3.2.1** Consider the line from  $(0, 0)$  to  $(4, 6)$ . Use the simple DDA algorithm to rasterize this line.

**Solution :** Evaluating steps 1 to 5 in the DDA algorithm we have

$$\begin{aligned} x_1 &= 0 & y_1 &= 0 & x_2 &= 4 & y_2 &= 6 \\ \therefore \text{Length} &= |y_2 - y_1| = 6 \\ \therefore \Delta x &= |x_2 - x_1| / \text{length} = \frac{4}{6} \end{aligned}$$

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

Initial value for

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

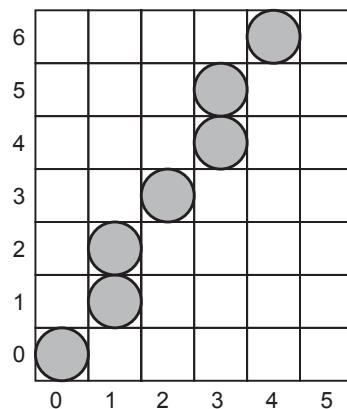
$$y = 0 + 0.5 * \text{Sign}(1) = 0.5$$

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

| i | x    | y    | Plot   |
|---|------|------|--------|
| 1 | 0.50 | 0.50 | (0, 0) |
| 2 | 1.17 | 1.50 | (1, 1) |
| 3 | 1.83 | 2.50 | (1, 2) |
| 4 | 2.50 | 3.50 | (2, 3) |
| 5 | 3.17 | 4.50 | (3, 4) |
| 6 | 3.83 | 5.50 | (3, 5) |
| 7 | 4.50 | 6.50 | (4, 6) |

**Table 3.2.1**

The results are plotted as shown in the Fig. 3.2.2. It shows that the rasterized line lies to both sides of the actual line, i.e. the algorithm is **orientation dependent**.



**Fig. 3.2.2 Result for a simple DDA**

**Example 3.2.2** Consider the line from  $(0, 0)$  to  $(-6, -6)$ . Use the simple DDA algorithm to rasterize this line.

**SPPU : June-12, Marks 10**

**Solution :** Evaluating steps 1 to 5 in the DDA algorithm we have

$$x_1 = 0$$

$$y_1 = 0$$

$$x_2 = -6$$

$$y_2 = -6$$

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

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

Initial values for

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

$$= -0.5$$

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

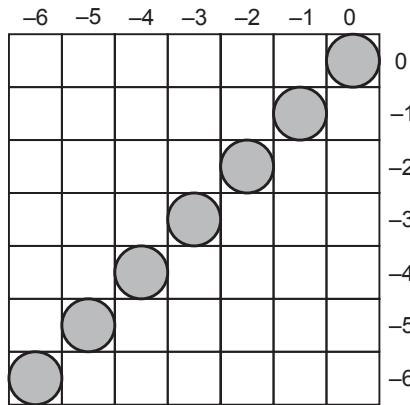
$$= -0.5$$

Tabulating the results of each iteration in the step 7 we get,

| i | x      | y      | Plot       |
|---|--------|--------|------------|
| 1 | - 0.50 | - 0.50 | $(0, 0)$   |
| 2 | - 1.50 | - 1.50 | $(-1, -1)$ |
| 3 | - 2.50 | - 2.50 | $(-2, -2)$ |
| 4 | - 3.50 | - 3.50 | $(-3, -3)$ |
| 5 | - 4.50 | - 4.50 | $(-4, -4)$ |
| 6 | - 5.50 | - 5.50 | $(-5, -5)$ |
| 7 | - 6.50 | - 6.50 | $(-6, -6)$ |

**Table 3.2.2**

The results are plotted as shown in the Fig. 3.2.3. It shows that the rasterized line lies on the actual line and it is  $45^\circ$  line.



\* -ve pixel values are  
with reference to pixel  
at the center of screen

**Fig. 3.2.3 Result for a simple DDA**

**Example 3.2.3** Using DDA algorithm find out which pixels would be turned on for the line with end points (1, 1) to (5, 3).

**SPPU : May-15, Marks 4**

**Solution.** :  $x_1 = 1, y_1 = 1, x_2 = 5, y_2 = 3$

$$\therefore \text{Length} = |x_2 - x_1| = 5 - 1 = 4$$

$$\therefore \Delta x = |x_2 - x_1| / \text{length} = \frac{4}{4} = 1$$

$$\therefore \Delta y = |y_2 - y_1| / \text{length} = \frac{2}{4} = 0.5$$

Initial value for

$$x = 1 + 0.5 * \text{sign}(1) = 1.5$$

$$y = 1 + 0.5 * \text{sign}(0.5) = 1.5$$

| i | x   | y   | Plot   |
|---|-----|-----|--------|
| 1 | 1.5 | 1.5 | (1, 1) |
| 2 | 2.5 | 2.0 | (2, 2) |
| 3 | 3.5 | 2.5 | (3, 2) |
| 4 | 4.5 | 3.0 | (4, 3) |
| 5 | 5.5 | 3.5 | (5, 3) |

**Example 3.2.4** Scan convert a line with end points (10, 5) and (16, 10) using DDA line drawing algorithm.

**SPPU : May-18, Marks 4**

**Solution :**  $x_1 = 10, x_2 = 16, y_1 = 5$  and  $y_2 = 10$

$$\therefore \text{Length} = |x_2 - x_1| = 16 - 10 = 6$$

$$\Delta y = |y_2 - y_1| / \text{length} = \frac{10 - 5}{6} = \frac{5}{6}$$

$$\Delta x = |x_2 - x_1| / \text{length} = \frac{6}{6} = 1$$

Initial values for,

$$x = 10 + 0.5 * \text{sign}(1) = 10.5$$

$$y = 5 + 0.5 * \text{sign}(5/6) = 5.5$$

| i | x     | y     | Plot     |
|---|-------|-------|----------|
| 1 | 10.50 | 5.50  | (10, 5)  |
| 2 | 11.50 | 6.33  | (11, 6)  |
| 3 | 12.50 | 7.16  | (12, 7)  |
| 4 | 13.50 | 8.99  | (13, 8)  |
| 5 | 14.50 | 8.83  | (14, 8)  |
| 6 | 15.50 | 9.66  | (15, 9)  |
| 7 | 16.50 | 10.49 | (16, 10) |

**Example 3.2.5** Consider line segment from  $A(-2, -1)$  to  $B(6, 3)$  user DDA line drawing algorithm to rasterize this line.

**SPPU : May-19, Marks 4**

**Solution :** Evaluating steps 1 to 5 in the DDA algorithm we have

$$x_1 = -2, y_1 = -1, x_2 = 6 \text{ and } y_2 = 3$$

$$\Delta x = |x_2 - x_1| = |6 - (-2)| = 8$$

$$\Delta y = |y_2 - y_1| = |3 - (-1)| = 4$$

$$\therefore \text{Length} = 8$$

$$\Delta x = (x_2 - x_1)/\text{Length} = 8/8 = 1$$

$$\Delta y = (y_2 - y_1)/\text{Length} = 4/8 = 0.5$$

| i | x  | y    | Plot     |
|---|----|------|----------|
| 1 | -2 | -1   | (-2, -1) |
| 2 | -1 | -0.5 | (-1, -1) |
| 3 | 0  | 0    | (0, 0)   |
| 4 | 1  | 0.5  | (1, 0)   |

|   |   |     |        |
|---|---|-----|--------|
| 5 | 2 | 1   | (2, 1) |
| 6 | 3 | 1.5 | (3, 1) |
| 7 | 4 | 2   | (4, 2) |
| 8 | 5 | 2.5 | (5, 2) |
| 9 | 6 | 3   | (6, 3) |

### Advantages of DDA Algorithm

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

### Disadvantages of DDA Algorithm

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

#### Examples for Practice

**Example 3.2.6 :** Digitize the line with end-points (10, 12) and (20, 18) using DDA algorithm.

**Example 3.2.7 :** Find out points for line segment having end points (0, 0) (- 8, - 4) using DDA line drawing algorithm.

SPPU : Dec.-15, Marks 6

#### 3.2.3 Bresenham's Line Algorithm

- Bresenham's line algorithm uses only integer addition and subtraction and multiplication by 2, and we know that the computer can perform the operations of integer addition and subtraction very rapidly. The computer is also time-efficient when performing integer multiplication by powers of 2. Therefore, it is an efficient method for scan-converting straight lines.
- The basic principle of Bresenham's line algorithm is to select the optimum raster locations to represent a straight line. To accomplish this the algorithm always increments either x or y by one unit depending on the slope of line. The increment in the other variable is determined by examining the distance between the actual line location and the nearest pixel. This distance is called decision variable or the error. This is illustrated in the Fig. 3.2.4.

- As shown in the Fig. 3.2.4, the line does not pass through all raster points (pixels). It passes through raster point  $(0, 0)$  and subsequently crosses three pixels. It is seen that the intercept of line with the line  $x = 1$  is closer to the line  $y = 0$ , i.e. pixel  $(1, 0)$  than to the line  $y = 1$  i.e. pixel  $(1, 1)$ . Hence, in this case, the raster point at  $(1, 0)$  better represents the path of the line than that at  $(1, 1)$ . The intercept of the line with the line  $x = 2$  is close to the line  $y = 1$ , i.e. pixel  $(2, 1)$  than to the line  $y = 0$ , i.e. pixel  $(2, 0)$ . Hence, the raster point at  $(2, 1)$  better represents the path of the line, as shown in the Fig. 3.2.4.

- In mathematical terms error or decision variable is defined as

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

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

- The error term is initially set as

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

where  $\Delta y = y_2 - y_1$ , and  $\Delta x = x_2 - x_1$

Then according to value of  $e$  following actions are taken.

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

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

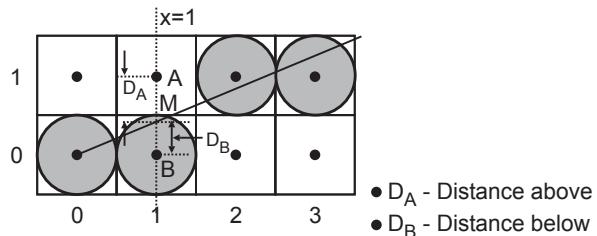


Fig. 3.2.4

**Bresenham's Line Algorithm for  $|m = \Delta y / \Delta x| < 1$** 

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

**Example 3.2.8** Consider the line from  $(5, 5)$  to  $(13, 9)$ . Use the Bresenham's algorithm to rasterize the line.

**Solution :** Evaluating steps 1 through 4 in the Bresenham's algorithm we have,

$$\begin{aligned}\Delta x &= |13 - 5| = 8 \\ \Delta y &= |9 - 5| = 4 \\ x &= 5 & y &= 5 \\ e &= 2 * \Delta y - \Delta x = 2 * 4 - 8 = 0\end{aligned}$$

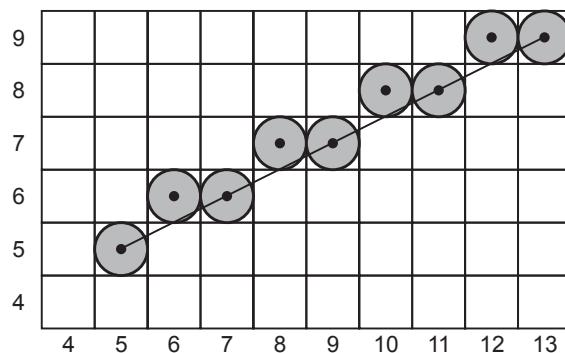
Tabulating the results of each iteration in the step 5 through 10.

| i | e  | x | y | Plot     |
|---|----|---|---|----------|
| 0 | 0  | 5 | 5 | $(5, 5)$ |
| 1 | -8 | 6 | 6 | $(6, 6)$ |
| 2 | 0  | 7 | 6 | $(7, 6)$ |
| 3 | -8 | 8 | 7 | $(8, 7)$ |
| 4 | 0  | 9 | 7 | $(9, 7)$ |

|   |     |    |    |          |
|---|-----|----|----|----------|
| 5 | - 8 | 10 | 8  | (10, 8)  |
| 6 | 0   | 11 | 8  | (11, 8)  |
| 7 | - 8 | 12 | 9  | (12, 9)  |
| 8 | 0   | 13 | 9  | (13, 9)  |
| 9 | - 8 | 14 | 10 | (14, 10) |

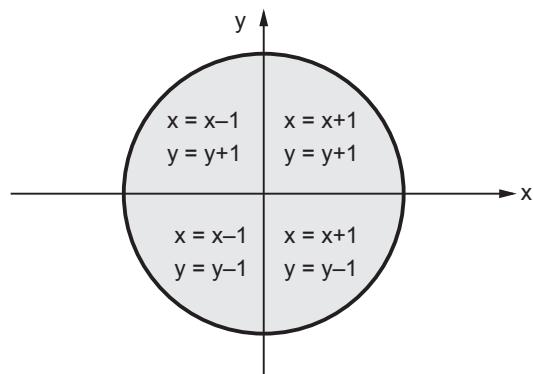
**Table 3.2.3**

The results are plotted as shown in Fig. 3.2.5.

**Fig. 3.2.5**

### 3.2.4 Generalized Bresenham's Line Drawing Algorithm

- The Bresenham's algorithm only works for the first octant.
- The generalized Bresenham's algorithm requires modification for lines lying in the other octants. Such algorithm can be easily developed by considering the quadrant in which the line lies and its slope.
- When the absolute magnitude of the slope of the line is greater than 1,  $y$  is incremented by one and Bresenham's decision variable

**Fig. 3.2.6 Conditions for generalized Bresenham's algorithm**

or error is used to determine when to increment x. The x and y incremental values depend on the quadrant in which the line exists. This is illustrated in Fig. 3.2.6.

### Generalized Bresenham's Algorithm

1. Read the line end point  $(x_1, y_1)$  and  $(x_2, y_2)$  such that they are not equal.
2.  $\Delta x = |x_2 - x_1|$  and  $\Delta y = |y_2 - y_1|$
3. Initialize starting point  
 $x = x_1$   
 $y = y_1$   
Plot  $(x, y)$
4.  $s_1 = \text{Sign}(x_2 - x_1)$   
 $s_2 = \text{Sign}(y_2 - y_1)$   
[Sign function returns -1, 0, 1 depending on whether its argument is  $< 0$ ,  $= 0$ ,  $> 0$  respectively]
5. if  $\Delta y > \Delta x$  then  
Exchange  $\Delta x$  and  $\Delta y$   
Ex\_change = 1
- else  
Ex\_change = 0
- end if  
[Interchange  $\Delta x$  and  $\Delta y$  depending on the slope of the line and set Ex\_change flag accordingly]
6.  $e = 2 * \Delta y - \Delta x$   
[Initialize value of decision variable or error to compensate for nonzero intercept ].
7.  $i = 1$   
[ Initialize counter ]
8. while ( $e \geq 0$ )  
{
  - if ( Ex\_change = 1 ) then  
 $x = x + s_1$   
else  
 $y = y + s_2$   
end if  
 $e = e - 2 * \Delta x$
}
9. if Ex\_change = 1 then  
 $y = y + s_2$   
else  
 $x = x + s_1$   
end if  
 $e = e + 2 * \Delta y$
10. plot  $(x, y)$
11.  $i = i + 1$
12. if (  $i \leq \Delta x$  ) then go to step 8
13. Stop

**Example 3.2.9** Generate all raster points on the line segment, if the two end points are given as (10, 20) and (18, 30) using the Bresenham's algorithm.

$$\text{Solution : } \Delta x = |18 - 10| = 8 \quad \Delta y = |30 - 20| = 10$$

$$x = 10 \quad y = 20$$

$$S_1 = 1 \quad S_2 = 1$$

Since  $\Delta y > \Delta x$

Exchange  $\Delta x$  and  $\Delta y$

$$\therefore \Delta x = 10 \text{ and } \Delta y = 8 \text{ and } \text{Ex\_change} = 1$$

Tabulating the result of each iteration we have,

| i  | e  | x  | y  | Plot     |
|----|----|----|----|----------|
| 1  | 6  | 10 | 20 | (10, 20) |
| 2  | 2  | 11 | 21 | (11, 21) |
| 3  | -2 | 12 | 22 | (12, 22) |
| 4  | 14 | 12 | 23 | (12, 23) |
| 5  | 10 | 13 | 24 | (13, 24) |
| 6  | 6  | 14 | 25 | (14, 25) |
| 7  | 2  | 15 | 26 | (15, 26) |
| 8  | -2 | 16 | 27 | (16, 27) |
| 9  | 14 | 16 | 28 | (16, 28) |
| 10 | 10 | 17 | 29 | (17, 29) |
| 11 | 6  | 18 | 30 | (18, 30) |

**Example 3.2.10** Using Bresenham's line algorithm, find out which pixel would be turned on for the line with end points (4, 4) to (12, 9)

**SPPU : Dec.-10, Marks 8**

**Solution :** Evaluating step 1 through 4 in the Bresenham's algorithm we have,

$$\Delta x = |12 - 4| = 8, \quad \Delta y = |9 - 4| = 5$$

$$x = 4, \quad y = 4$$

$$e = 2 * \Delta y - \Delta x = 2 * 5 - 8 = 2$$

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

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

**Example 3.2.11** Consider the line from (1, 1) to (6, 4). Use Bresenham's line drawing algorithm to rasterize this line and give output pixels.

**SPPU : Dec.-11, Marks 10**

**Solution :** Evaluating step 1 through 4 in Bresenham's algorithm we have,

$$\Delta x = |6 - 1| = 5, \quad \Delta y = |4 - 1| = 3$$

$$x = 1, \quad y = 1$$

$$e = 2 * \Delta y - \Delta x = 2 * 3 - 5 = 1$$

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

| i | e   | x | y | Plot   |
|---|-----|---|---|--------|
| 1 | 1   | 1 | 1 | (1, 1) |
| 2 | - 3 | 2 | 2 | (2, 2) |
| 3 | 3   | 3 | 2 | (3, 2) |
| 4 | - 1 | 4 | 3 | (4, 3) |
| 5 | 5   | 5 | 3 | (5, 3) |
| 6 | 1   | 6 | 4 | (6, 4) |

**Example 3.2.12** Consider a line from (4, 9) to (7, 7). Use Bresenham's line drawing algorithm to rasterize this line.

**SPPU : June-12, Marks 10**

**Solution :** Evaluating step 1 through 4 in Bresenham's algorithm we have,

$$\Delta x = |7 - 4| = 3, \quad \Delta y = |7 - 9| = 2$$

$$x = 4, \quad y = 9$$

$$e = 2 * \Delta y - \Delta x = 2 * 2 - 3 = 1$$

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

| i | e  | x | y | Plot   |
|---|----|---|---|--------|
| 1 | 1  | 4 | 9 | (4, 9) |
| 2 | -1 | 5 | 8 | (5, 8) |
| 3 | 3  | 6 | 8 | (6, 8) |
| 4 | 1  | 7 | 7 | (7, 7) |

**Example 3.2.13** Consider the line from (0, 0) to (6, 6). Use Bresenham's algorithm to rasterize this line.

SPPU : May-13, 16, Marks 6

**Solution :** Evaluating step 1 through 4 in the Bresenham's algorithm we have,

$$\Delta x = |6 - 0| = 6, \quad \Delta y = |6 - 0| = 6$$

$$x = 0, \quad y = 0$$

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

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

| i | e | x | y | Plot   |
|---|---|---|---|--------|
| 1 | 6 | 0 | 0 | (0, 0) |
| 2 | 6 | 1 | 1 | (1, 1) |
| 3 | 6 | 2 | 2 | (2, 2) |
| 4 | 6 | 3 | 3 | (3, 3) |
| 5 | 6 | 4 | 4 | (4, 4) |
| 6 | 6 | 5 | 5 | (5, 5) |
| 7 | 6 | 6 | 6 | (6, 6) |

**Example 3.2.14** Find out which pixel would be turned on for the line with end points (2, 2) to (6, 5) using the Bresenham's algorithm.

SPPU : May-14, Marks 3

**Solution :** Evaluating step 1 through 4 in the Bresenham's algorithm.

$$\Delta x = |6 - 2| = 4, \quad \Delta y = |5 - 2| = 3$$

$$x = 2, \quad y = 2$$

$$e = (2 * \Delta y) - \Delta x = (2 * 3) - 4 = 2$$

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

| i | e  | x | y | Plot   |
|---|----|---|---|--------|
| 1 | 2  | 2 | 2 | (2, 2) |
| 2 | 0  | 3 | 3 | (3, 3) |
| 3 | -2 | 4 | 4 | (4, 4) |
| 4 | 4  | 5 | 4 | (5, 4) |
| 5 | 2  | 6 | 5 | (6, 5) |

**Example 3.2.15** Find out which pixels would be turned on for the line with end points (5, 2) to (8, 4) using the Bresenham's algorithm.

SPPU : Dec.-14, Marks 6

**Solution :**  $\Delta x = |8 - 5| = 3$ ,  $\Delta y = |4 - 2| = 2$ ,  $x = 5$ ,  $y = 2$

$$e = 2 * \Delta y - \Delta x = 2 * 2 - 3 = 1$$

| i | e  | x | y | Plot   |
|---|----|---|---|--------|
| 1 | 1  | 5 | 2 | (5, 2) |
| 2 | -1 | 6 | 3 | (6, 3) |
| 3 | 3  | 7 | 3 | (7, 3) |
| 4 | 1  | 8 | 4 | (8, 4) |

Pixels would be turned on for the line : (5, 2), (6, 3), (7, 3), (8, 4).

**Example 3.2.16** Find out which pixel would be turned on for the line with end points (3, 2) to (7, 4) using the Bresenham's algorithm.

SPPU : May-17, Marks 4

**Solution :**

$$\Delta x = |7 - 3| = 4, \Delta y = |4 - 2| = 2, x = 3, y = 2$$

$$e = 2 * \Delta y - \Delta x = 2 * 2 - 4 = 0$$

| i | e  | x | y | Plot   |
|---|----|---|---|--------|
| 1 | 0  | 3 | 2 | (3, 2) |
| 2 | -4 | 4 | 3 | (4, 3) |
| 3 | 0  | 5 | 3 | (5, 3) |
| 4 | -4 | 6 | 4 | (6, 4) |
| 5 | 0  | 7 | 4 | (7, 4) |

Pixels would be turned on for the line : (3, 2), (4, 3), (5, 3), (6, 4), (7, 4)

**Example 3.2.17** Consider a line from (2, 5) to (8, 8). Use Bresenham's line drawing algorithm rasterize this line.

SPPU : Dec.-19, Marks 6

**Solution :** Evaluating step 1 through step 4 in the Bresenham's algorithm we have,

$$\Delta x = |8 - 2| = 6, \Delta y = |8 - 5| = 3$$

$$x = 2, y = 5$$

$$e = 2 * \Delta y - \Delta x = 2 * 3 - 6 = 0$$

Tabulating the result of each iteration in the step 5 through 10

| i | e   | x | y | Plot   |
|---|-----|---|---|--------|
| 1 | 0   | 2 | 5 | (2, 5) |
| 2 | - 6 | 3 | 6 | (3, 6) |
| 3 | 0   | 4 | 6 | (4, 6) |
| 4 | - 6 | 5 | 7 | (5, 7) |
| 5 | 0   | 6 | 7 | (6, 7) |
| 6 | - 6 | 7 | 8 | (7, 8) |
| 7 | 0   | 8 | 8 | (8, 8) |

**Example 3.2.18** Differentiate Bresenham and Vector generation algorithm for line.

Solution :

| Sr. No. | Vector generation algorithm                                                  | Bresenham's line algorithm                                          |
|---------|------------------------------------------------------------------------------|---------------------------------------------------------------------|
| 1.      | It uses floating point arithmetic.                                           | It uses only integer addition, subtraction and multiplication by 2. |
| 2.      | Due to floating point arithmetic it takes more time.                         | It is quicker than vector generation algorithm.                     |
| 3.      | Less efficient.                                                              | More efficient.                                                     |
| 4.      | Where speed is important this algorithm needs to be implemented in hardware. | Hardware implementation is not required.                            |

**Examples for Practice**

**Example 3.2.19 :** Digitize the line with end-points (- 2, 5) to (5, 12) using Bresenham's line algorithm.

**Example 3.2.20 :** Digitize the line with end-points (3, 15) to (- 4, 18) using Bresenham's line algorithm.

**Review Questions**

- What is vector generation ?
- What is vector generation ? Explain the problem of vector generation.

SPPU : May-05, Marks 2

SPPU : Dec.-12, Marks 6

3. Explain DDA algorithm for line. Discuss its advantages and disadvantages.

**SPPU : May-05, 10, 11, 17, 19 Dec.-06,07,08, Marks 10**

4. Describe Bresenham's line drawing algorithm.

**SPPU : May-14, 15, 16, 17, 19 Dec 07, 12, 18 Marks 4**

5. Explain any two advantages of Bresenham's line drawing algorithm over other line drawing algorithm.

**SPPU : Dec 18 Marks 2**

6. Give differences between Bresenham's and DDA line drawing algorithm.

**SPPU : May-11, Marks 8**

### 3.3 Antialiasing and Antialiasing Techniques

**SPPU : Dec.-10,14, May-13**

- In the line drawing algorithms, we have seen that all rasterized locations do not match with the true line and we have to select the optimum raster locations to represent a straight line. This problem is severe in low resolution screens. In such screens line appears like a stair-step, as shown in the Fig. 3.3.1. This effect is known as **aliasing**.
- It is dominant for lines having gentle and sharp slopes.
- The aliasing effect can be reduced by adjusting intensities of the pixels along the line. The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called **antialiasing**.
- The aliasing effect can be minimized by increasing resolution of the raster display. By increasing resolution and making it twice the original one, the line passes through twice as many column of pixels and therefore has twice as many jags, but each jag is half as large in x and in y direction.

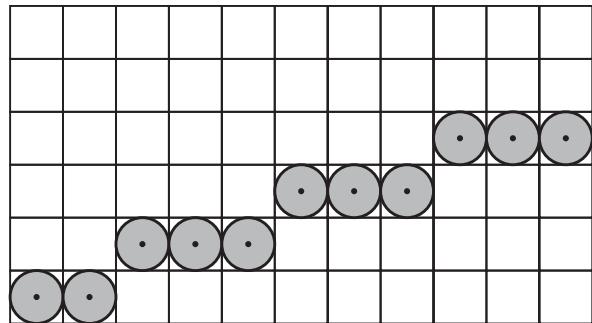


Fig. 3.3.1 Aliasing effect

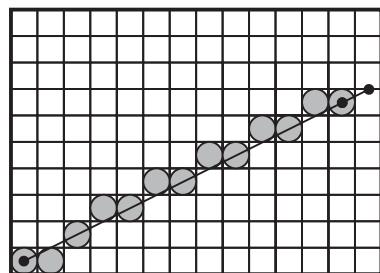
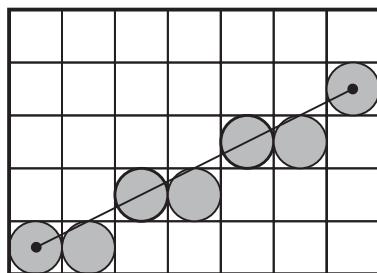


Fig. 3.3.2 Effect on aliasing with increase in resolution

- As shown in the Fig. 3.3.2, line looks better in twice resolution, but this improvement comes at the price of quadrupling the cost of memory, bandwidth of memory and scan-conversion time. Thus increasing resolution is an expensive method for reducing aliasing effect.
- With raster systems that are capable of displaying more than two intensity levels (colour or gray scale), we can apply antialiasing methods to modify pixel intensities. By appropriately varying the intensities of pixels along the line or object boundaries, we can smooth the edges to lessen the stair-step or the jagged appearance. Anti-aliasing methods are basically classified as
  - Supersampling or Postfiltering :** In this antialiasing method the sampling rate is increased by treating the screen as if it were covered with a finer grid than is actually available. We can then use multiple sample points across this finer grid to determine an appropriate intensity level for each screen pixel. This technique of sampling object characteristics at a high resolution and displaying the results at a lower resolution is called **supersampling** or **postfiltering**.
  - Area sampling or Prefiltering :** In this antialiasing method pixel intensity is determined by calculating the areas of overlap of each pixel with the objects to be displayed. Antialiasing by computing overlap area is referred to as **area sampling** or **prefiltering**.

Antialiasing methods are further classified as shown in the Fig. 3.3.3. Let us discuss them one by one.

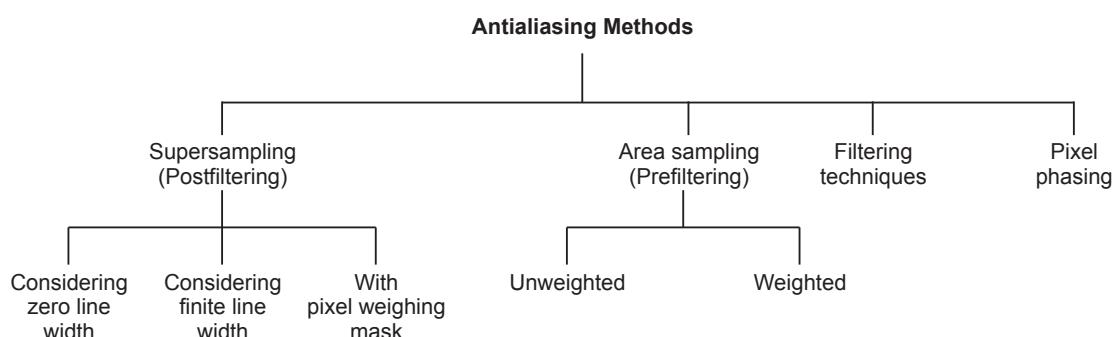


Fig. 3.3.3 Classification of antialiasing methods

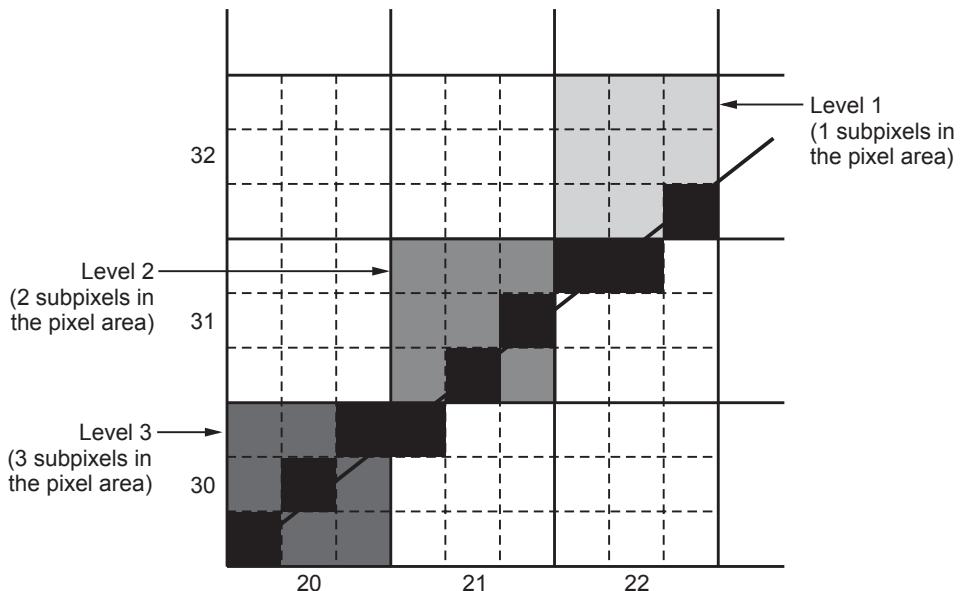
### 3.3.1 Supersampling Considering Zero Line Width

Supersampling of straight lines with zero line width can be performed in several ways.

- For the gray scale display of a straight-line segment, we can divide each pixel into a number of subpixels and count the number of subpixels that are along the line

path. The intensity level for each pixel is then set to a value that is proportional to this subpixel count.

- This is illustrated in Fig. 3.3.4. Here, each pixel area is divided into nine equal-sized square subpixels. The black regions show the subpixels that would be selected by Bresenham's algorithm. In this example, the pixel at position (20, 30) has 3 subpixels along the line path, the pixel at position (21, 31) has 2 subpixels along the line path and the pixel at position (22, 32) has 1 subpixel along the line path. Since pixel at position (20, 30) has 3 (maximum) subpixels its intensity level is kept highest (level 3). The intensity level at pixel position (21, 31) is kept to the medium level (level 2) because pixel has 2 subpixels along the line path, and the intensity at pixel position (22, 32) is kept to the lowest level above zero intensity because it has only 1 subpixel along the line path. Thus the line intensity is spread out over a greater number of pixels, and the stair-step effect is smoothed by displaying a somewhat blurred line path in the vicinity of the stair-steps.
- We can use more intensity levels to antialias the line. For this we have to increase the number of sampling positions across each pixel.
- For example, sixteen subpixels gives us four intensity levels above zero and twenty five subpixels gives us five levels; and so on.



**Fig. 3.3.4 Supersampling of subpixel positions**

### 3.3.2 Supersampling Considering Finite Line Width

- Actually most of the times line width is equal to the size of pixel instead of zero width.

- If we take this finite width of the line into account, we can perform supersampling by setting each pixel intensity proportional to the number of subpixels inside the polygon representing the line area as shown in the Fig. 3.3.5.
- A subpixel can be considered to be inside the line if its lower left corner is inside the polygon boundaries. This techniques has following advantages :
  - The number of possible intensity levels for each pixel is equal to the total number of subpixels within the pixel area.
  - The total line intensity is distributed over more pixels.
  - In colour displays, we can extend the method to take background colours into account.

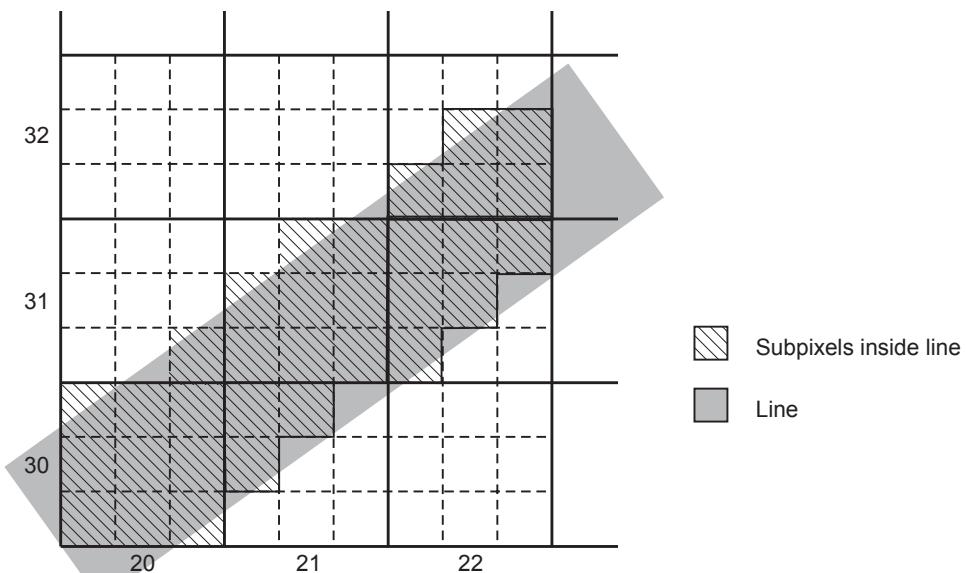


Fig. 3.3.5 Super sampling with finite line width

### 3.3.3 Supersampling with Pixel-Weighting Mask

- Supersampling method often implemented by giving more weight to subpixels near the center of a pixel area, since we consider these subpixels to be more important in determining the overall intensity of a pixel.
- Fig. 3.3.6 shows the weighting scheme for 3 by 3 pixel subdivisions. Here, center subpixel has four time weight that of the corner subpixels and twice that of the remaining pixels.
- An array of values specifying the relative weights of subpixels is sometimes referred to as a **mask** of subpixel weights.

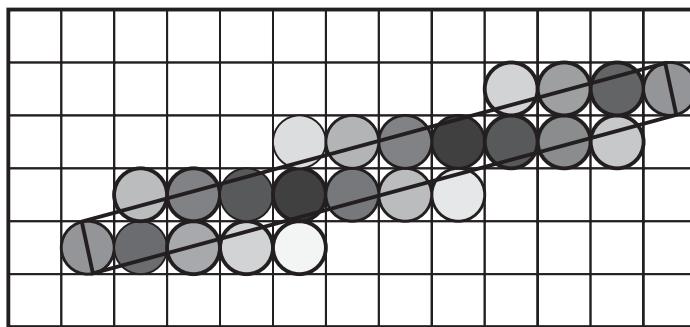
|   |   |   |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

Fig. 3.3.6 Weighing mask for a grid of 3 by 3 subpixels

- Such pixel-weighting mask can be set up for larger subpixel grids. These masks are often extended to include contributions from subpixels belonging to neighbouring pixels, so that intensities can be averaged over adjacent pixels.

### 3.3.4 Unweighted Area Sampling

- We have seen that for sloped lines, many a times the line passes between two pixels. In these cases, line drawing algorithm selects the pixel which is closer to the true line. This step in line drawing algorithms can be modified to perform antialiasing.
- In antialiasing, instead of picking closest pixel, both pixels are highlighted. However, their intensity values may differ.

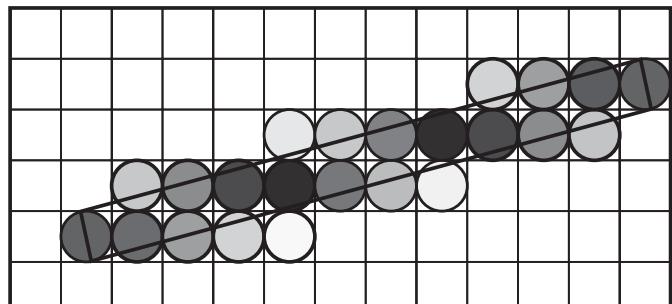


**Fig. 3.3.7 Unweighted area sampling**

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

### 3.3.5 Weighted Area Sampling

- We have seen that in unweighted area sampling equal areas contribute equal intensity, regardless of the distance between the pixel's center and the area; only the total amount of occupied area matters. Thus, a small area in the corner of the pixel contributes just as much as does an equal-sized area near the pixel's



**Fig. 3.3.8 Weighted area sampling**

center. To avoid this problem even better strategy is used in the weighted area sampling.

- In weighted area sampling equal areas contribute unequally i.e. a small area closer to the pixel center has greater intensity than does one at a greater distance. Thus, in weighted area sampling the intensity of the pixel is dependent on the line area occupied and the distance of area from the pixel's center. This is illustrated in Fig. 3.3.8.

### 3.3.6 Filtering Techniques

- Filtering technique is more accurate method for antialiasing lines. This method is similar to applying a weighted pixel mask, but here we consider a continuous weighting surface (or filter function) covering the pixel.
- Fig. 3.3.9 shows examples of box, conical and Gaussian filter functions. In this method, we integrate over the pixel surface to obtain the weighted average intensity.

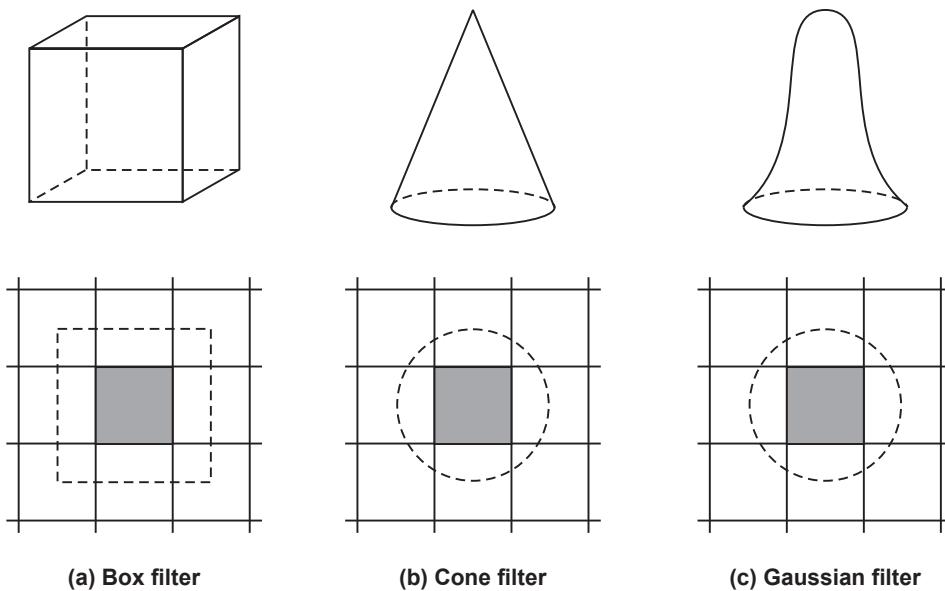


Fig. 3.3.9 Common filter functions used to antialias line paths

### 3.3.7 Pixel Phasing

- Another way to antialias raster objects is to shift the display location of pixel areas. This technique is called **pixel phasing**. It is applied by "micropositioning" the electron beam in relation to object geometry.
- Such technique can be used on raster systems that can address subpixel positions within the screen grid. In this technique stairsteps along a linepath or object

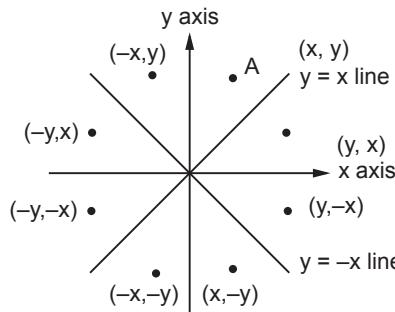
boundary are smoothed out by moving (micropositioning) the electron beam to more nearly approximate positions specified by the object geometry. The electron beam is typically shifted by 1/4, 1/2, or 3/4 of a pixel diameter to plot points closer to the true path of a line or object edge.

### Review Questions

1. List and explain any two antialiasing methods. **SPPU : Dec.-10, May-13, Dec.-14, Marks 4**
2. What is aliasing ? **SPPU : Dec.-14, Marks 2**
3. What is pixel phasing ?

## 3.4 Basic Concepts in Circle Drawing

- A circle is a symmetrical figure. It has eight-way symmetry as shown in the Fig. 3.4.1.
- Any circle generating algorithm can take advantage of the circle symmetry to plot eight points by calculating the co-ordinates of any one point.
- For example, if point A in the Fig. 3.4.1 is calculated with a circle algorithm, seven more points could be found just by reflection.
- There are two standard methods of mathematically representing a circle centered at the origin.



**Fig. 3.4.1 Eight-way symmetry of a circle**

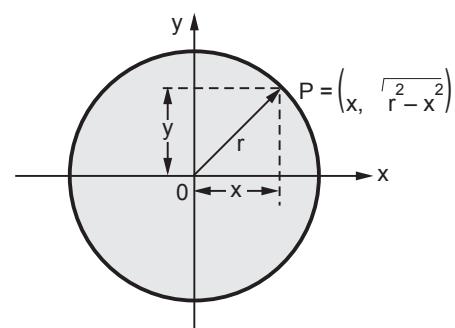
### 3.4.1 Polynomial Method

- In this method circle is represented by a polynomial equation.

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

where       $x$  : the  $x$  co-ordinate  
                $y$  : the  $y$  co-ordinate  
                $r$  : radius of the circle

- Here, polynomial equation can be used to find  $y$  co-ordinate for the known  $x$  co-ordinate. Therefore, the scan converting circle using polynomial method is achieved by stepping  $x$  from 0 to  $r\sqrt{2}$ , and each  $y$



**Fig. 3.4.2 Scan converting circle using polynomial method**

co-ordinate is found by evaluating  $\sqrt{r^2 - x^2}$  for each step of  $x$ . This generates the 1/8 portion ( $90^\circ$  to  $45^\circ$ ) of the circle. Remaining part of the circle can be generated just by reflection.

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

### 3.4.2 Trigonometric Method

- In this method, the circle is represented by use of trigonometric functions  
 $x = r \cos \theta$  and  $y = r \sin \theta$   
 where  $\theta$  : current angle  
 $r$  : radius of the circle  
 $x$  : the  $x$  co-ordinate  
 $y$  : the  $y$  co-ordinate
- The scan converting circle using trigonometric method is achieved by stepping  $\theta$  from 0 to  $\pi/4$  radians and each value of  $x$  and  $y$  is calculated.
- This method is more inefficient than the polynomial method because the computation of the values of  $\sin \theta$  and  $\cos \theta$  is even more time-consuming than the calculations required in the polynomial method.

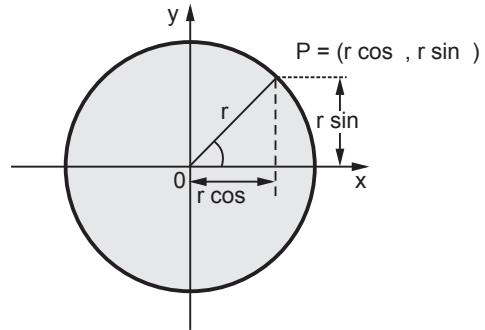


Fig. 3.4.3 Scan converting circle using trigonometric method

### Review Questions

- Explain the basic concepts in circle drawing.
- Give different methods of representing a circle.

## 3.5 Circle Drawing Algorithms

SPPU : May-05,06,07,13,14,16,18,Dec.-07,12,15,17

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

- Vector generation/DDA algorithm and
- Bresenham's algorithm
- Midpoint algorithm

### 3.5.1 Vector Generation/DDA Circle Drawing Algorithm

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

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

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

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

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

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

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

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

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

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

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

$$\therefore \epsilon = 2^{-6} = 0.0156$$

- Applying these incremental steps we have,

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

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

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

Therefore, now we have  $x_{n+1} = x_n + \epsilon y_n$

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

### Algorithm

- Read the radius ( $r$ ), of the circle and calculate value of  $\epsilon$
- $\text{start\_x} = 0$
- $\text{start\_y} = r$
- $x_1 = \text{start\_x}$   
 $y_1 = \text{start\_y}$
- do
  - $x_2 = x_1 + \epsilon y_1$
  - $y_2 = y_1 - \epsilon x_2$

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

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

$x_1 = x_2$  ;  
 $y_1 = y_2$  ;

[Reinitialize the current point ]

```

} while (y1 - start_y) < ε or (start_x - x1) > ε
[check if the current point is the starting point or not. If current point is not
starting point repeat step 4 ; otherwise stop]
5. Stop.

```

**Example 3.5.1** Calculate the pixel position along the circle path with radius  $r = 6$  centered on the origin using DDA circle algorithm from point  $(0, 6)$  to point  $x = y$ .

**Solution :** Let us find the value of  $\epsilon$ . The value of  $\epsilon$  is given as,  $\epsilon = 2^{-n}$  where value of 'n' should satisfy the following expression

$$2^{n-1} \leq r < 2^n$$

Here,  $r = 6 \quad \therefore \text{Value of } n = 3$

$$\therefore \epsilon = 2^{-3} = 0.125$$

Once we know the value of  $\epsilon$  we can determine incremental steps as follows :

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

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

Tabulating the results of step 4 we get,

| x     | y     | Plot   |
|-------|-------|--------|
| 0.000 | 6.000 | (0, 6) |
| 0.750 | 5.906 | (1, 6) |
| 1.488 | 5.720 | (1, 6) |
| 2.203 | 5.445 | (2, 5) |
| 2.884 | 5.084 | (3, 5) |
| 3.519 | 4.644 | (4, 5) |
| 4.100 | 4.132 | (4, 4) |

### Example for Practice

**Example 3.5.2 :** Calculate the pixel position along the circle path with radius  $r = 8$  centered on the origin using-DDA circle algorithm from point  $(0, 8)$  to point  $x = y$ .

### 3.5.2 Bresenham's Circle Drawing Algorithm

- If the circle is to be plotted efficiently, the use of trigonometric and power functions must be avoided.

- It is desirable to perform the calculations necessary to find the scan-converted points with only integer addition, subtraction, and multiplication by powers of 2.
- Bresenham's circle drawing algorithm allows these goals to be met.
- The Bresenham's circle drawing algorithm considers the eight-way symmetry of the circle to generate it. It plots  $1/8^{\text{th}}$  part of the circle, i.e. from  $90^{\circ}$  to  $45^{\circ}$ , as shown in the Fig. 3.5.1. As circle is drawn from  $90^{\circ}$  to  $45^{\circ}$ , the x moves in positive direction and the y moves in negative direction.
- To achieve best approximation to the true circle, we have to select those pixels in the raster that fall the least distance from the true circle. Refer Fig. 3.5.2. Let us observe the  $90^{\circ}$  to  $45^{\circ}$  portion of the circle. It can be noticed that if points are generated from  $90^{\circ}$  to  $45^{\circ}$ , each new point closest to the true circle can be found by applying either of the two options :

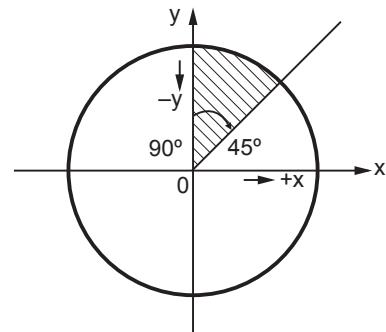


Fig. 3.5.1 1/8 part of circle

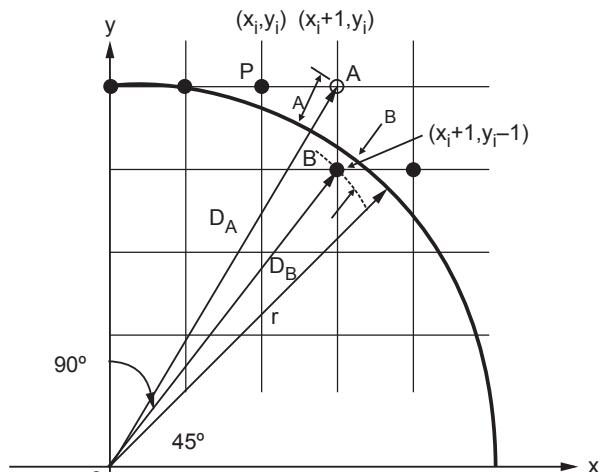


Fig. 3.5.2 Scan conversion with Bresenham's algorithm

- Increment in positive x direction by one unit or
- Increment in positive x direction and negative y direction both by one unit
- Therefore, a method of selecting between these two choices is all that is necessary to find the points closest to the true circle.
- Let us assume point  $P(x_i, y_i)$  in Fig. 3.5.2 as a last scan converted pixel. Now we have two options either to choose pixel A or pixel B. The closer pixel amongst these two can be determined as follows
- The distances of pixels A and B from the origin are given as

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

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

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

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

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

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

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

That is

$$\delta_A = (x_i + 1)^2 + y_i^2 - r^2 \quad \dots (3.5.1)$$

$$\delta_B = (x_i + 1)^2 + (y_i - 1)^2 - r^2 \quad \dots (3.5.2)$$

This function D provides a relative measurement of the distance from the centre of a pixel to the true circle. Since  $D_A$  will always be positive (A is outside of the true circle) and  $D_B$  will always be negative (B is inside of the true circle), a **decision variable**  $d_i$  can be defined as

$$d_i = \delta_A + \delta_B \quad \dots (2.10.3)$$

From equations (3.5.1), (3.5.2) and (3.5.3) we have

$$\begin{aligned} d_i &= (x_i + 1)^2 + y_i^2 - r^2 + (x_i + 1)^2 + (y_i - 1)^2 - r^2 \\ &= 2(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2r^2 \end{aligned} \quad \dots (3.5.4)$$

When  $d_i < 0$ , we have  $|\delta_A| < |\delta_B|$  and pixel A is chosen. When  $d_i \geq 0$ , we have  $|\delta_A| \geq |\delta_B|$  and pixel B is selected. In other words we can write,

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

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

### Derivation of Decision Variable ( $d_{i+1}$ )

$$d_{i+1} = 2(x_{i+1} + 1)^2 + y_{i+1}^2 + (y_{i+1} - 1)^2 - 2r^2$$

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

Since  $x_{i+1} = x_i + 1$  We have,

$$\begin{aligned} d_{i+1} &= d_i + 2(x_i + 1 + 1)^2 + y_{i+1}^2 + (y_{i+1} - 1)^2 \\ &\quad - 2(x_i + 1)^2 - y_i^2 - (y_i - 1)^2 \\ &= d_i + 2(x_i^2 + 4x_i + 4) + y_{i+1}^2 + y_{i+1}^2 - 2y_{i+1} + 1 \\ &\quad - 2(x_i^2 + 2x_i + 1) - y_i^2 - (y_i^2 - 2y_i + 1) \end{aligned}$$

$$\begin{aligned}
 &= d_i + 2x_i^2 + 8x_i + 8 + 2y_{i+1}^2 - 2y_{i+1} + 1 - 2x_i^2 - 4x_i - 2 \\
 &\quad - y_i^2 - y_i^2 + 2y_i - 1 \\
 &= d_i + 4x_i + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) + 6
 \end{aligned}$$

If A is chosen pixel (meaning that the  $d_i < 0$ ) then  $y_{i+1} = y_i$  and so  $d_{i+1} = d_i + 4x_i + 6$ . On the other hand, if B is the chosen pixel (meaning that the  $d_i \geq 0$ ) then  $y_{i+1} = y_i - 1$  and so

$$\begin{aligned}
 d_{i+1} &= d_i + 4x_i + 2[(y_i - 1)^2 - y_i^2] - 2[y_i - 1 - y_i] + 6 \\
 &= d_i + 4x_i + 2[y_i^2 - 2y_i + 1 - y_i^2] - 2(-1) + 6 \\
 &= d_i + 4x_i - 4y_i + 2 + 2 + 6 = d_i + 4(x_i - y_i) + 10
 \end{aligned}$$

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

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

We can now summarize the algorithm for generating all the pixel coordinates in the  $90^\circ$  to  $45^\circ$  octant that are needed when scan-converting a circle of radius  $r$ .

#### Algorithm to plot 1/8 of the circle

- ```

1. Read the radius (r) of the circle.
2. d = 3 - 2r
   [Initialize the decision variable]
3. x = 0, y = r
   [Initialize starting point]
4. do
{
    plot (x, y)
    if (d < 0) then
    {
        d = d + 4x + 6
    }
    else
    {
        d = d + 4(x - y) + 10
        y = y - 1
    }
}

```

```

    }
    x = x + 1
} while (x < y)
5. Stop

```

- The remaining part of circle can be drawn by reflecting point about y axis, x axis and about origin as shown in Fig. 3.5.3.
- By adding seven more plot commands after the plot command in the step 4 of the algorithm, the circle can be plotted. The remaining seven plot commands are :

plot (y, x)
 plot (y, -x)
 plot (x, -y)
 plot (-x, -y)
 plot (-y, -x)
 plot (-y, x) and
 plot (-x, y)

Example 3.5.3 Calculate the pixel position along the circle path with radius $r = 10$ centered on the origin using Bresenham's circle drawing algorithm from point (0,10) to point $x = y$.

Solution : The value of d is given as,

$$d = 3 - 2r = 3 - 2 * 10 = -17$$

Tabulating the results of step 4 we get,

x	y	d
0	10	-17
1	10	-11
2	10	-1
3	10	13
4	9	-5
5	9	17
6	8	11
7	7	13

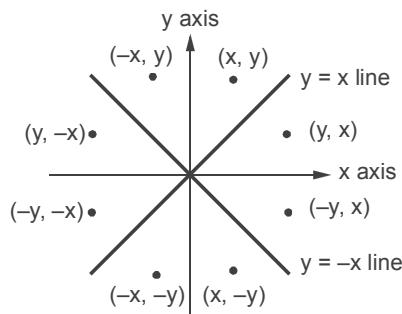


Fig. 3.5.3 Eight-way symmetry of the circle

Example for Practice

Example 3.5.4 : Calculate the pixel position along the circle path with radius $r = 14$ centered on the origin using Bresenham's circle drawing algorithm from point $(0, 14)$ to point $x = y$.

3.5.3 Midpoint Circle Algorithm

The midpoint circle drawing algorithm also uses the eight-way symmetry of the circle to generate it. It plots 1/8 part of the circle, i.e. from 90° to 45° , as shown in the Fig. 3.5.4. As circle is drawn from 90° to 45° , the x moves in the positive direction and y moves in the negative direction. To draw a 1/8 part of a circle we take unit steps in the positive x direction and make use of decision parameter to determine which of the two possible y positions is closer to the circle path at each step. The Fig. 2.6.4 shows the two possible y positions (y_i and $y_i + 1$) at sampling position $x_i + 1$. Therefore, we have to determine whether the pixel at position $(x_i + 1, y_i)$ or at position $(x_i + 1, y_i - 1)$ is closer to the circle. For this purpose decision parameter is used. It uses the circle function ($f_{\text{circle}}(x, y) = x^2 + y^2 - r^2$) evaluated at the midpoint between these two pixels.

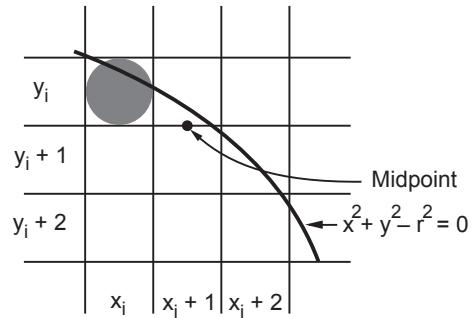


Fig. 3.5.4 Decision parameter to select correct pixel in circle generation algorithm

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

$$\begin{aligned} d_{i+1} &= f_{\text{circle}}(x_{i+1} + 1, y_{i+1} - \frac{1}{2}) = [(x_{i+1} + 1) + 1]^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2 \\ &= (x_{i+1} + 1)^2 + y_{i+1}^2 - y_{i+1} + \frac{1}{4} - r^2 \end{aligned} \quad \dots (3.5.5)$$

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

$$\begin{aligned} d_{i+1} &= f_{\text{circle}}(x_{i+1} + 1, y_{i+1} - \frac{1}{2}) = [(x_{i+1} + 1) + 1]^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - r^2 \\ &= (x_{i+1} + 1)^2 + 2(x_{i+1} + 1) + y_{i+1}^2 - (y_{i+1}) + \frac{1}{4} - r^2 \end{aligned} \quad \dots (3.5.6)$$

Looking at equations (2.6.5) and (2.6.6) we can write

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

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

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

$$\begin{aligned} \therefore d_{i+1} &= d_i + 2(x_i + 1) + 1 \\ &= d_i + 2x_{i+1} + 1 \end{aligned} \quad \dots (3.5.7)$$

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

$$\therefore d_{i+1} = d_i + 2(x_i + 1) + 1 - 2y_{i+1} \quad \dots (3.5.8)$$

The terms $2x_{i+1}$ and $-2y_{i+1}$ in equations (3.5.3) and (3.5.4) can be incrementally calculated as

$$\begin{aligned} 2x_{i+1} &= 2x_i + 2 \\ 2y_{i+1} &= 2y_i - 2 \end{aligned}$$

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

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

Algorithm

1. Read the radius (r) of the circle
2. Initialize starting position as
 $x = 0$
 $y = r$
3. Calculate initial value of decision parameter as
 $d = 1.25 - r$
4. do


```

        {
          plot (x, y)
          if (d < 0)
            {
              x = x + 1
              y = y
              d = d + 2x + 1
            }
          else
            {
              x = x + 1
              y = y - 1
              d = d + 2x + 2y + 1
            }
        }
      
```
- while ($x < y$)

5. Determine symmetry points
6. Stop.

Example 3.5.5 Calculate the pixel position along the circle path with radius $r = 10$ centered on the origin using midpoint circle drawing algorithm from point $(0,10)$ to point $x = y$.

Solution : Initial value of decision parameter

$$d = 1.25 - r = -8.75$$

i	d	x	y
1	-8.75	0	10
2	-5.75	1	10
3	-0.75	2	10
4	6.25	3	10
5	-2.75	4	9
6	8.25	5	9
7	5.25	6	8
8	6.25	7	7

Review Questions

1. Explain vector generation algorithm for circle. **SPPU : May-05, Marks 6**
2. Derive the expression for decision parameter used in Bresenham's circle algorithm . Also explain the Bresenham's circle algorithm. **SPPU : May-06, 07, Dec.-07, 12, Marks 10**
3. Explain Bresenham's circle drawing algorithm with mathematical derivation. **SPPU : May-14, May-16, Marks 6**
4. Explain Bresenham's circle drawing algorithm in detail. Also explain error factor with derivations. **SPPU : May-13, Dec.-15, Marks 12**
5. Explain vector/DDA generation algorithm for circle. **SPPU : May-05, Marks 6**
6. Derive the expression for decision parameter used in Bresenham's circle algorithm . Also explain the Bresenham's circle algorithm. **SPPU : May-06, 07, Dec.-07, 12, Marks 10**
7. Explain Bresenham's circle drawing algorithm with mathematical derivation. **SPPU : May-14, 16, 18 Dec 17 Marks 6**
8. Explain Bresenham's circle drawing algorithm in detail. Also explain error factor with derivations. **SPPU : May-13, Dec.-15, Marks 12**
9. Write and explain the midpoint circle generating algorithm.



UNIT - II

4

Polygons and Polygon Filling

Syllabus

Polygons : Introduction to polygon, types : convex, concave and complex. Inside test.

Polygon Filling : flood fill, seed fill, scan line fill.

Contents

4.1	Introduction to Polygon	Dec.-06, May-11,12,..... Marks 4
4.2	Types : Convex, Concave and Complex	Dec.-06, 09,18, Marks 4
4.3	Representation of Polygon	
4.4	Inside Test	Dec.-06, 09, 10, 12, 14, 15,18 Marks 6
4.5	Polygon Filling Algorithms	May-05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15,18,19 Dec.-05, 06, 07, 08, 09, 10, 11, 12, 15,19..... Marks 16

4.1 Introduction to Polygon

SPPU : Dec.-06, May-11,12

In this chapter we are going to study different types of polygons, their representation and filling algorithms for them.

- A polyline is a chain of connected line segments.
- It is specified by giving the vertices (nodes) $P_0, P_1, P_2 \dots$ and so on.
- The first vertex is called the initial or starting point and the last vertex is called the final or **terminal point**, as shown in the Fig. 4.1.1 (a).
- When starting point and terminal point of any polyline is same, i.e. when polyline is closed then it is called **polygon**. This is illustrated in Fig. 4.1.1 (b).

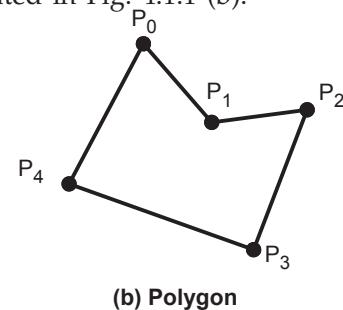
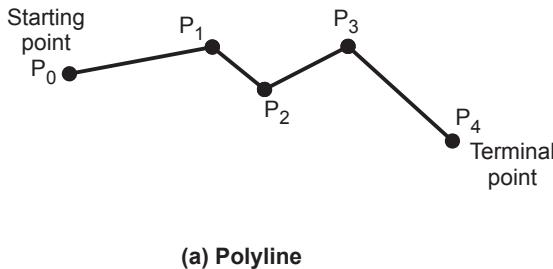


Fig. 4.1.1

Review Questions

1. What is polygon ?
2. Is circle a polygon ? Justify

SPPU : Dec.-06, May-12, Marks 2

SPPU : May-11, Marks 4

4.2 Types : Convex, Concave and Complex

SPPU : May-05,10,11,12, Dec.-06,09,18

- The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie. There are three types of polygons :
 - Convex
 - Concave and
 - Complex
- A convex polygon is a polygon in which the line segment joining any two points within the polygon lies completely inside the polygon. Fig. 4.2.1 shows the examples of convex polygons.

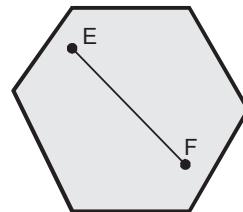
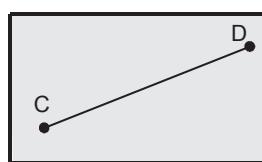
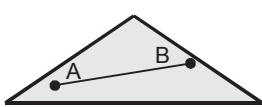


Fig. 4.2.1 Convex polygons

- A concave polygon is a polygon in which the line segment joining any two points within the polygon may not lie completely inside the polygon. Fig. 4.2.2 shows the examples of concave polygons.

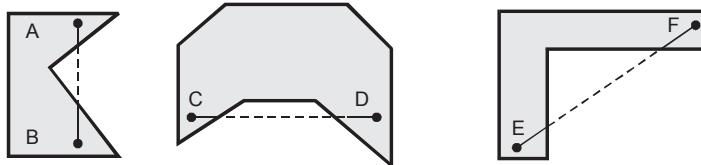


Fig. 4.2.2 Concave polygons

- A polygon is simple if it is described by a single, non-intersecting boundary; otherwise it is said to be complex.

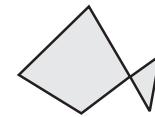


Fig. 4.2.3 Complex polygon

Review Questions

1. With suitable diagram explain concave and convex polygons ?

SPPU : May-05, Dec.-06, 18, Marks 2

2. What are different types of polygons ? Explain with example

SPPU : Dec.-09, May-10, 11, 12, Marks 4

4.3 Representation of Polygon

- There are three approaches to represent polygons according to the graphics systems :
 - Polygon drawing primitive approach
 - Trapezoid primitive approach
 - Line and point approach
- Some graphics devices supports polygon drawing primitive approach. They can directly draw the polygon shapes. On such devices polygons are saved as a unit.
- Some graphics devices support trapezoid primitive. In such devices, trapezoids are formed from two scan lines and two line segments as shown in the Fig. 4.3.1. Here, trapezoids are drawn by stepping down the line segments with two vector generators and, for each scan line, filling in all the pixels between them. Therefore every polygon is broken up into trapezoids and it is represented as a series of trapezoids.

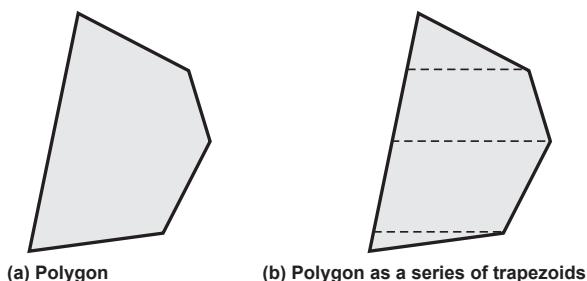


Fig. 4.3.1 Representation of polygon

- Most of the other graphics devices do not provide any polygon support at all. In such cases polygons are represented using lines and points. A polygon is represented as a unit and it is stored in the display file. In a display file polygon can not be stored only with series of line commands because they do not specify how many of the following line commands are the part of the polygon. Therefore, new command is used in the display file to represent polygons. The opcode for new command itself specifies the number of line segments in the polygon. The Fig. 4.3.2 shows the polygon and its representation using display file.

DF_OP	DF_x	DF_y
6	0	2
2	0	4
2	4	6
2	6	4
2	6	2
2	4	0
2	0	2

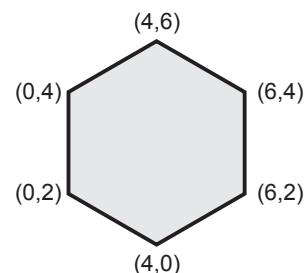


Fig. 4.3.2 Polygon and its representation using display file

Review Question

- Explain various approaches used to represent polygon.

4.4 Inside Test

SPPU : May-05,07,10,13,16,19 Dec.-06,09,10,12,14,15,18

- Once the polygon is entered in the display file, we can draw the outline of the polygon.
- To show polygon as a solid object we have to set the pixels inside the polygon as well as pixels on the boundary of it.
- Now the question is how to determine whether or not a point is inside of a polygon. One simple method of doing this is to construct a line segment between the point in question and a point known to be outside the polygon, as shown in the Fig. 4.4.1. Now count how many intersections of the line segment with the polygon boundary occur. If there are an odd number of intersections, then the point in question is inside; otherwise it is outside. This method is called the **even-odd method** of determining polygon inside points.

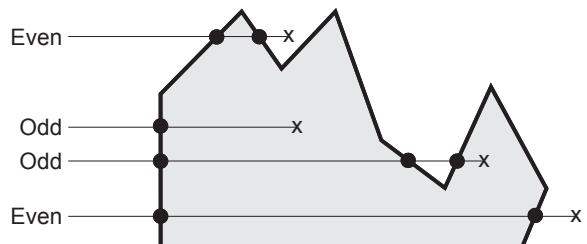


Fig. 4.4.1

- If the intersection point is vertex of the polygon then we have to look at the other endpoints of the two segments which meet at this vertex. If these points lie on the same side of the constructed line, then the point in question counts as an even number of intersections. If they lie on opposite sides of the constructed line, then the point is counted as a single intersection. This is illustrated in Fig. 4.4.2.

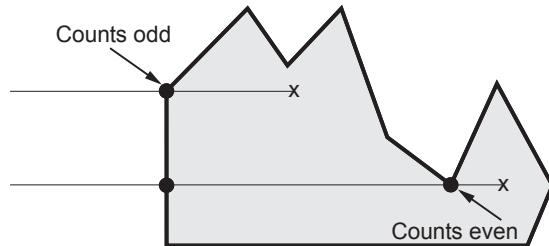


Fig. 4.4.2

- Another approach to do the inside test is the **winding-number method**. Let us consider a point in question and a point on the polygon boundary. Conceptually, we can stretch a piece of elastic between these points and slide the end point on the polygon boundary for one complete rotation. We can then examine the point in question to see how many times the elastic has wound around it. If it is wound at least once, then the point is inside. If there is no net winding, then the point is outside.

- Like even-odd method, in winding number method we have to picturise a line segment running from outside the polygon to the point in question and consider the polygon sides which it crosses. Here, instead of just counting the intersections, we have to give a **direction number** to each boundary line crossed, and we have to sum these direction numbers. The direction number indicates the direction the polygon edge was drawn relative to the line segment we have constructed for the test. This is illustrated in Fig. 4.4.3.

- As shown in the Fig. 4.4.3, point (x_a, y_a) is a test point and line $y = y_a$ is a horizontal line runs from outside the polygon to point (x_a, y_a) . The polygon edges crossed by this line could be drawn in two ways. The edge could be drawn starting below the line, cross it, and end above the line or starting above the line, cross it, and end below the line. In first case we have to give direction number as -1 and in the second case we have give direction number as 1 . After giving the direction numbers we have to take sum of these direction numbers which indicates whether the point is inside the polygon or not. The sum of the direction numbers for the sides that cross the constructed horizontal line segment is called the **winding number** for the point in question. For polygons or two dimensional objects, the point is said to be inside when the value of winding number is nonzero. It is important to note that the line we choose must not pass through any vertices.

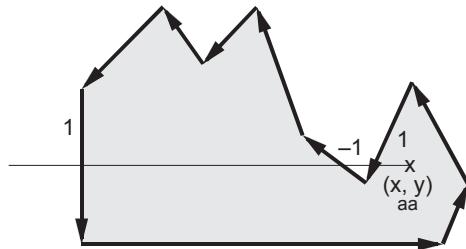


Fig. 4.4.3

Review Questions

1. Explain even-odd method to determine polygon interior points.

SPPU : Dec.-12,15, May-13,16, Marks 4

2. Explain the different methods for testing a pixel inside of polygon.

SPPU : May-05,07,10,13, Dec.-09,10,12,14, Marks 6

3. Explain the winding number method. Does the method supports intersection polygons?

SPPU : Dec.-06, Marks 4

4. Write and explain any one inside test algorithm.

SPPU : Dec.-18, May-19, Marks 4

5. Explain any one inside test algorithm.

SPPU : May-19, Marks 2

4.5 Polygon Filling Algorithms

SPPU : May-05,06,07,08,09,10,11,12,13,14,15,18,19, Dec.-05,06,07,08,09,10,11,12,15,19

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

4.5.1 Seed Fill

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

4.5.1.1 Boundary Fill Algorithm / Edge Fill Algorithm

- In this method, edges of the polygons are drawn. Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check

whether the boundary pixel is reached. If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.

- Boundary defined regions may be either 4-connected or 8-connected as shown in the Fig. 4.5.1.
- If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions : left, right, up and down.
- For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical and four diagonal directions.

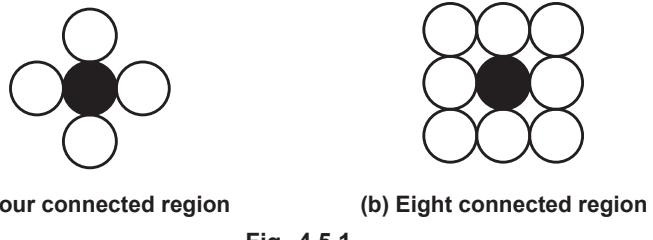


Fig. 4.5.1

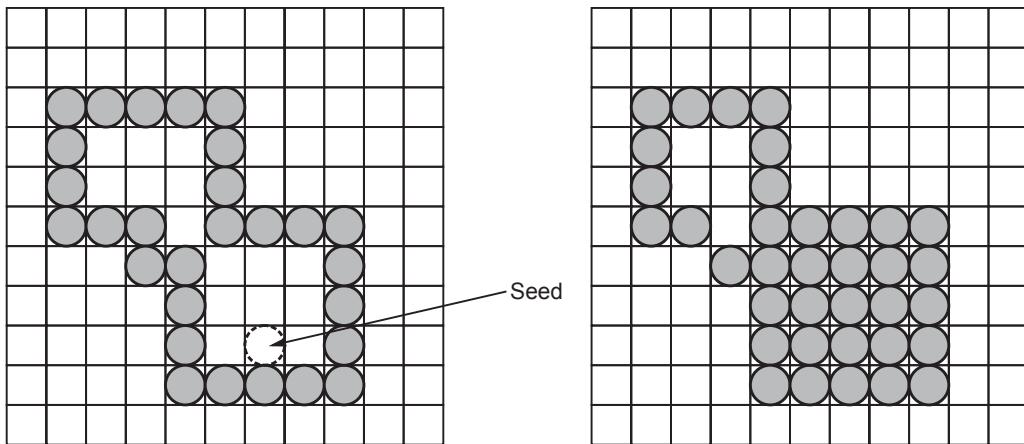


Fig. 4.5.2 Partial filling resulted using 4-connected algorithm

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

Procedure : boundary_fill (x, y, f_colour, b_colour)

```
{
    if (getpixel (x,y) != b_colour && getpixel (x, y) != f_colour)
```

```

    {
        putpixel (x, y, f_colour)
        boundary_fill (x + 1, y, f_colour, b_colour);
        boundary_fill (x, y + 1, f_colour, b_colour);
        boundary_fill (x - 1, y, f_colour, b_colour);
        boundary_fill (x, y - 1, f_colour, b_colour);
    }
}

```

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

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

4.5.1.2 Flood Fill Algorithm

- Sometimes it is required to fill in an area that is not defined within a single colour boundary. In such cases we can fill areas by replacing a specified interior colour instead of searching for a boundary colour. This approach is called a flood-fill algorithm.
- Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels. However, here pixels are checked for a specified interior colour instead of boundary colour and they are replaced by new colour.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled. The following procedure illustrates the recursive method for filling 8-connected region using flood-fill algorithm.

Procedure : flood_fill (x, y, old_colour, new_colour).

```

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

```

```

    flood_fill (x - 1, y + 1, old_colour, new_colour);
}
}

```

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

4.5.2 Scan Line Algorithm

- Recursive algorithm for seed fill methods have got two difficulties :
 - The first difficulty is that if some inside pixels are already displayed in fill colour then recursive branch terminates, leaving further internal pixels unfilled. To avoid this difficulty, we have to first change the colour of any internal pixels that are initially set to the fill colour before applying the seed fill procedures.
 - Another difficulty with recursive seed fill methods is that it cannot be used for large polygons. This is because recursive seed fill procedures require stacking of neighbouring points and in case of large polygons stack space may be insufficient for stacking of neighbouring points.
- To avoid this problem more efficient method can be used. Such method fills horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighbouring points. This is achieved by identifying the rightmost and leftmost pixels of the seed pixel and then drawing a horizontal line between these two boundary pixels. This procedure is repeated with changing the seed pixel above and below the line just drawn until complete polygon is filled. With this efficient method we have to stack only a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.
- Fig. 4.5.3 illustrates the scan line algorithm for filling of polygon.
- For each scan line crossing a polygon, this algorithm locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right and the corresponding positions between each intersection pair are set to the specified fill colour.

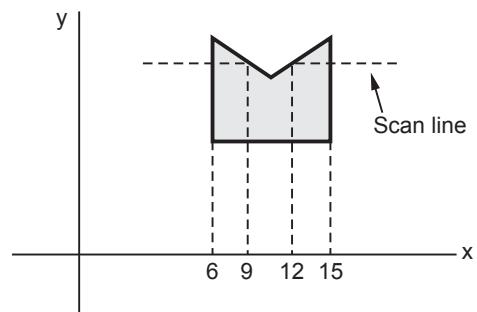


Fig. 4.5.3

- In Fig. 4.5.3, we can see that there are two stretches of interior pixels from $x = 6$ to $x = 9$ and $x = 12$ to $x = 15$.
- The scan line algorithm first finds the largest and smallest y values of the polygon. It then starts with the largest y value and works its way down, scanning from left to right, in the manner of a raster display.
- The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary.
- When intersection points are even, they are sorted from left to right, paired and pixels between paired points are set to the fill colour.
- In some cases intersection point is a vertex. When scan line intersects polygon vertex a special handling is required to find the exact intersection points. To handle such cases, we must look at the other endpoints of the two line segments of the polygon which meet at this vertex. If these points lie on the same (up or down) side of the scan line, then the point in question counts as an even number of intersections. If they lie on opposite sides of the scan line, then the point is counted as single intersection. This is illustrated in Fig. 4.5.4.

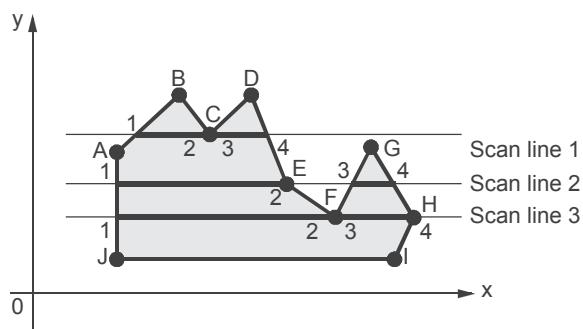


Fig. 4.5.4 Intersection points along the scan line that intersect polygon vertices

- As shown in Fig. 4.5.4, each scan line intersects the vertex or vertices of the polygon.
- For scan line 1, the other end points (B and D) of the two line segments of the polygon lie on the same side of the scan line, hence there are two intersections resulting two pairs : 1 - 2 and 3 - 4. Intersections points 2 and 3 are actually same points.
- For scan line 2 the other endpoints (D and F) of the two line segments of the polygon lie on the opposite sides of the scan line, hence there is a single intersection resulting two pairs : 1 - 2 and 3 - 4.
- For scan line 3, two vertices are the intersection points. For vertex F the other end points E and G of the two line segments of the polygon lie on the same side of the scan line whereas for vertex H, the other endpoints G and I of the two line segments of the polygon lie on the opposite side of the scan line. Therefore, at vertex F there are two intersections and at vertex H there is only one intersection.

- This results two pairs : 1 - 2 and 3 - 4 and points 2 and 3 are actually same points.
- It is necessary to calculate x intersection points for scan line with every polygon side.
- We can simplify these calculations by using **coherence properties**.
- A coherence property of a scene is a property of a scene by which we can relate one part of a scene with the other parts of a scene. Here, we can use a slope of an edge as a coherence property. By using this property we can determine the x intersection value on the lower scan line if the x intersection value for current scan line is known. This is given as

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

where m is the slope of the edge

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

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

- Many times it is not necessary to compute the x intersections for scan line with every polygon side.
- We need to consider only the polygon sides with endpoints straddling the current scan line. See Fig. 4.5.5.
- It will be easier to identify which polygon sides should be tested for x-intersection, if we first sort the sides in order of their maximum y value.
- Once the sides are sorted we can process the scan lines from the top of the polygon to its bottom producing an active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that scan line.
- Fig. 4.5.6 shows sorted edges of the polygon with active edges.
- A scan line algorithm for filling a polygon begins by ordering the polygon sides on the largest y value.
- It begins with the largest y value and scans down the polygon.

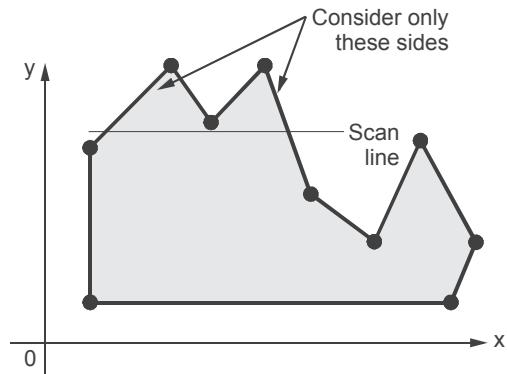
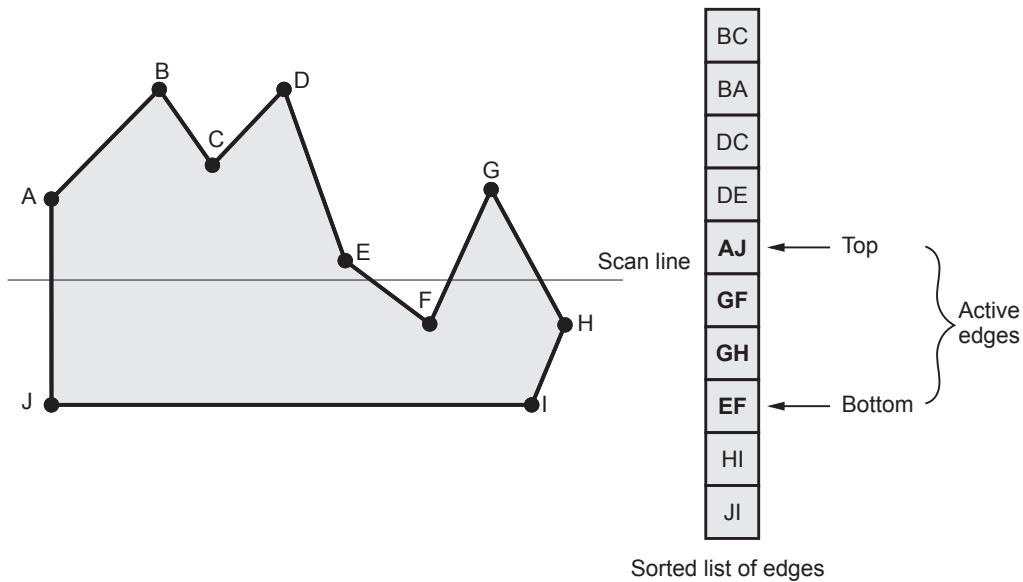


Fig. 4.5.5 Consider only the sides which intersect the scan line

**Fig. 4.5.6 Sorted edges of the polygon with active edges**

- For each y , it determines which sides can be intersected and finds the x values of these intersection points.
- It then sorts, pairs and passes these x values to a line drawing routine.

Scan Line Conversion Algorithm for Polygon Filling :

1. Read n , the number of vertices of polygon
2. Read x and y coordinates of all vertices in array $x[n]$ and $y[n]$.
3. Find y_{\min} and y_{\max} .
4. Store the initial x value (x_1) y values y_1 and y_2 for two endpoints and x increment Δx from scan line to scan line for each edge in the array edges [n] [4].
While doing this check that $y_1 > y_2$, if not interchange y_1 and y_2 and corresponding x_1 and x_2 so that for each edge, y_1 represents its maximum y coordinate and y_2 represents its minimum y coordinate.
5. Sort the rows of array, edges [n] [4] in descending order of y_1 , descending order of y_2 and ascending order of x_2 .
6. Set $y = y_{\max}$
7. Find the active edges and update active edge list :
if ($y > y_2$ and $y \leq y_1$)
 { edge is active }
else
 { edge is not active }

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

$$x_{i+1} \leftarrow x_i + \Delta x$$
where $\Delta x = -\frac{1}{m}$ and $m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e. slope of a line segment]
9. If x intersect is vertex i.e. x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x-intersect [] array.
10. Sort x-intersect [] array in the ascending order,
11. Extract pairs of intersects from the sorted x-intersect [] array.
12. Pass pairs of x values to line drawing routine to draw corresponding line segments
13. Set $y = y - 1$
14. Repeat steps 7 through 13 until $y \geq y_{\min}$.
15. Stop

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

Features of scan line algorithm

1. Scan line algorithm is used for filling of polygons.
2. This algorithm solves the problem of hidden surfaces while generating display scan line.
3. It is used in orthogonal projection.
4. It is non recursive algorithm.
5. In scan line algorithm we have to stack only a beginning position for each horizontal pixel scan, instead of stacking all unprocessed neighbouring positions around the current position. Therefore, it is efficient algorithm.

Review Questions

1. What is polygon filling. **SPPU : May-18, Dec.-19, Marks 2**
2. Explain boundary-fill/edge-fill algorithm for polygon. **SPPU : May-05,09, Dec.-19, Marks 6**
3. Explain polygon filling by seed-fill algorithm. **SPPU : Dec.-05,08, May-08,18, Marks 6**
4. What are the steps involved in filling polygon in scan line method? **SPPU : Dec.-05,06,08, May-06,07,08,18,19, Marks 8**
5. Explain with example and compare seed-fill and edge-fill algorithm for polygon. **SPPU : May-06,10, Dec.-07, 09, Marks 8**
6. Enlist any three polygon filling algorithms. **SPPU : May-13, Marks 4**

- | | |
|----------------------------------------------------------------------------------------------------------------|----------------------------------------|
| 7. Explain how a polygon is filled with pattern. | SPPU : Dec.-12, May-13, Marks 4 |
| 8. List various polygon filling algorithms. Explain scan line algorithm with mathematical formulation | SPPU : May-11, Marks 16 |
| 9. Compare different polygon filling algorithm. | SPPU : Dec.-12, Marks 4 |
| 10. State the characteristics of scan line polygon fill algorithm and compare it with boundary fill algorithm. | SPPU : May-09, 10, Marks 8 |
| 11. Explain Scanline algorithm for polygon filling and explain how it can be extended for hidden removal. | SPPU : Dec.-10, Marks 10 |
| 12. Explain scan line algorithm with example. | SPPU : May-14,19, Marks 6 |
| 13. What is scan line polygon filling algorithm ? Explain with an example. | SPPU : May-12, Marks 10 |
| 14. Describe scan line algorithm to generate solid area on the screen. | SPPU : Dec.-11, Marks 8 |
| 15. Write algorithm to fill the polygon area using flood fill method. | SPPU : May-15, Marks 4 |
| 16. Write flood fill algorithm. | SPPU : Dec.-15, Marks 3 |



UNIT - II

5

Windowing and Clipping

Syllabus

Viewing transformations, 2-D clipping: Cohen- Sutherland algorithm line Clipping algorithm, Sutherland Hodgeman Polygon clipping algorithm, Weiler Atherton Polygon Clipping algorithm.

Contents

5.1	<i>Introduction</i>	May-05,13, Dec.-05,12, Marks 4
5.2	<i>Viewing Transformations</i>	May-05,12, Dec.-10,11,19 Marks 8
5.3	<i>2 D Clipping</i>	May-07,12 Marks 2
5.4	<i>Cohen-Sutherland Line Clipping Algorithm</i>	May-06,07,08,10,11,12,13, 15, 16,17,19	
		Dec.-10, 15,18 Marks 8
5.5	<i>Polygon Clipping</i>	Dec.-05,06,07,08,11,14,17,18	
		May-07,10,14, Marks 8
5.6	<i>Generalized Clipping</i>	Dec.-07,11,12, May-09, Marks 8
5.7	<i>Interior and Exterior Clipping</i>	May-05,13, Dec.-05,12, Marks 4

5.1 Introduction

SPPU : May-05,13, Dec.-05,12

- We have to identify the visible part of the picture for inclusion in the display image. This selection process is not straight forward. Certain lines may lie partly inside the visible portion of the picture and partly outside. These lines cannot be omitted entirely from the display image because the image would become inaccurate. This is illustrated in Fig. 5.1.1.
- The process of selecting and viewing the picture with different views is called **windowing**, and a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded is called **clipping**.

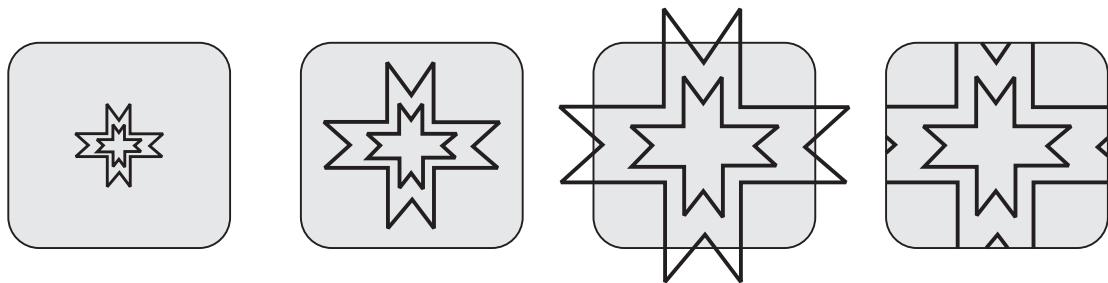


Fig. 5.1.1 Concept of windowing and clipping

Review Questions

1. What is window ?

SPPU : May-05, Marks 2

2. What is windowing and clipping ?

SPPU : May-13, Dec.-05, 12, Marks 4

5.2 Viewing Transformations

SPPU : May-05,12, Dec.-10,11,19

- The picture is stored in the computer memory using any convenient Cartesian co-ordinate system, referred to as **World Co-ordinate System** (WCS).
- When picture is displayed on the display device it is measured in **Physical Device Co-ordinate System** (PDCS) corresponding to the display device. Therefore, displaying an image of a picture involves mapping the co-ordinates of the points and lines that form the picture into the appropriate physical device co-ordinate where the image is to be displayed. This mapping of co-ordinates is achieved with the use of co-ordinate transformation known as **viewing transformation**.
- Sometimes the two dimensional viewing transformation is simply referred to as the **window to view port transformation** or the **windowing transformation**.
- The viewing transformation which maps picture co-ordinates in the WCS to display co-ordinates in PDCS is performed by the following transformations.

- Converting world co-ordinates to viewing co-ordinates.
- Normalizing viewing co-ordinates.
- Converting normalized viewing coordinates to device co-ordinates

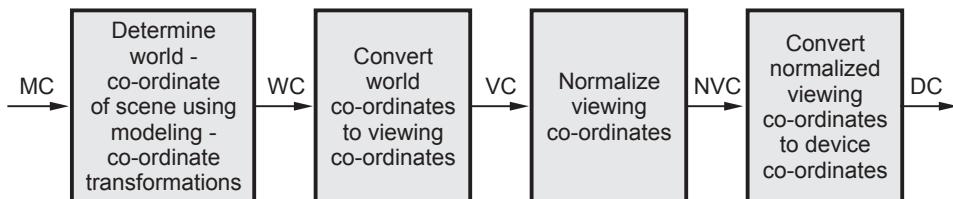


Fig. 5.2.1 (a) Two-dimensional viewing transformation pipeline

- World Co-ordinate System (WCS) is infinite in extent and the device display area is finite.
- To perform a viewing transformation we select a finite world co-ordinate area for display called a **window**.
- An area on a device to which a window is mapped is called a **viewport**.
- The window defines what is to be viewed; the viewport defines where it is to be displayed, as shown in the Fig. 5.2.1 (b).

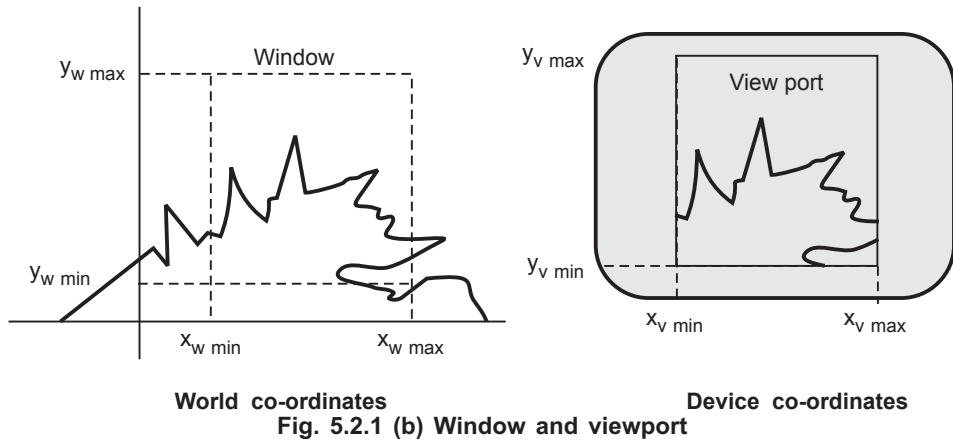


Fig. 5.2.1 (b) Window and viewport

The steps involved in viewing transformations :

1. Construct the scene in world co-ordinates using the output primitives and attributes.
2. Obtain a particular orientation for the window by setting a two-dimensional viewing co-ordinate system in the world-co-ordinate plane and define a window in the viewing co-ordinate system.
3. Use viewing co-ordinates reference frame to provide a method for setting up arbitrary orientations for rectangular windows.
4. Once the viewing reference frame is established, transform descriptions in world co-ordinates to viewing co-ordinates.

5. Define a view port in normalized co-ordinates and map the viewing co-ordinate description of the scene to normalized co-ordinates.
6. Clip all the parts of the picture which lie outside the viewport.

5.2.1 Viewing Co-ordinate Reference Frame

- Fig. 5.2.1 (b) shows the two dimensional viewing pipeline.
- The window defined in world co-ordinate is first transformed into the viewport co-ordinate. For this, viewing co-ordinate reference frame is used. It provides a method for setting up arbitrary orientations for rectangular windows.
- To obtain the matrix for converting world co-ordinate positions to viewing co-ordinates, we have to translate the viewing origin to the world origin and then align the two co-ordinate reference frames by rotation. This is illustrated in Fig. 5.2.2.
- As shown in the Fig. 5.2.2, the composite transformation (translation and rotation) converts world co-ordinates to viewing co-ordinates.

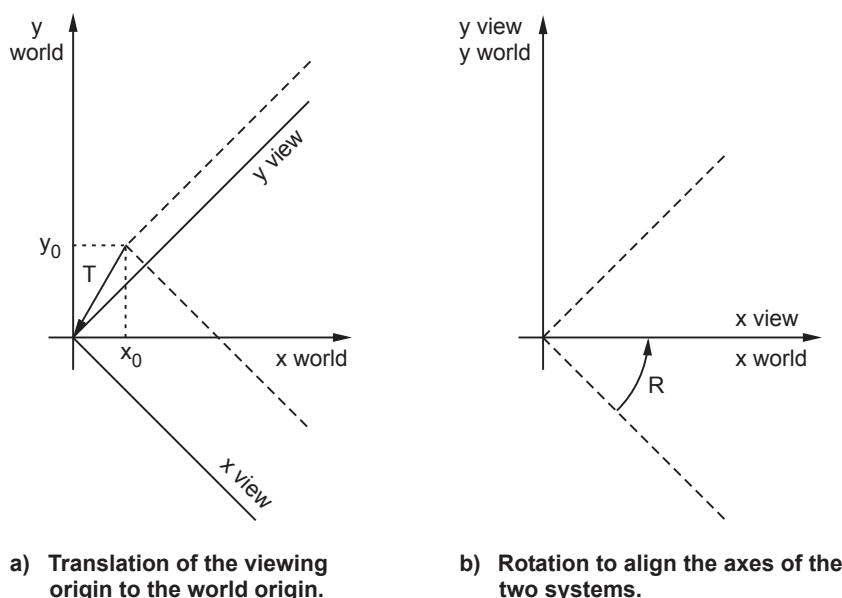


Fig. 5.2.2

- The matrix used for this composite transformation is given by
- $$M_{WC,VC} = T \cdot R$$

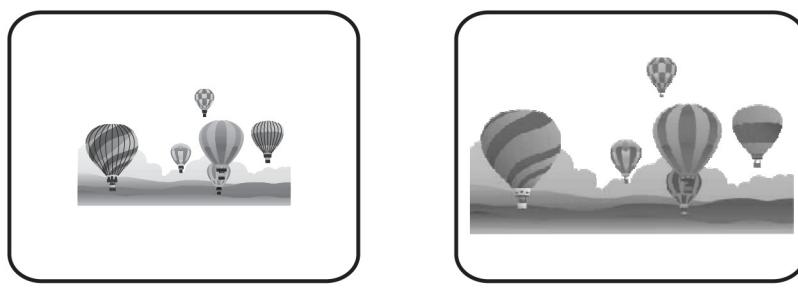
where T is the translation matrix that takes the viewing origin point P_0 to the world origin, and R is the rotation matrix that aligns the axes of the two reference frames.

5.2.2 Transformation to Normalized Co-ordinates

- Once the viewing reference frame is established, we can transform descriptions in world co-ordinate to viewing co-ordinates. We then define a viewport in normalized co-ordinate.

Normalized Co-ordinates

- The different display devices may have different screen sizes as measured in pixels.
- Size of the screen in pixels increases as resolution of the screen increases.
- When picture is defined in the pixel values then it is displayed large in size on the low resolution screen while small in size on the high resolution screen as shown in the Fig. 5.2.3.



(a) More resolution

(b) Less resolution

Fig. 5.2.3 Picture definition in pixels

- To avoid this and to make our programs to be device independent, we have to define the picture co-ordinates in some units other than pixels and use the interpreter to convert these co-ordinates to appropriate pixel values for the particular display device. The device independent units are called the **normalized device co-ordinates**. In these units, the screen measures 1 unit wide and 1 unit length as shown in the Fig. 5.2.4.
- The lower left corner of the screen is the origin, and the upper-right corner is the point (1, 1).
- The point (0.5, 0.5) is the center of the screen no matter what the physical dimensions or resolution of the actual display device may be.
- The interpreter uses a simple linear formula to convert the normalized device co-ordinates to the actual device co-ordinates.

$$x = x_n \times X_W \quad \dots (5.2.1)$$

$$y = y_n \times Y_H \quad \dots (5.2.2)$$

where

x : Actual device x co-ordinate.

- y : Actual device y co-ordinate.
- x_n : Normalized x co-ordinate.
- y_n : Normalized y co-ordinate.
- X_W : Width of actual screen in pixels.
- Y_H : Height of actual screen in pixels.

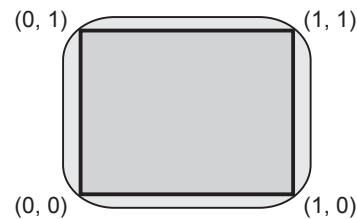


Fig. 5.2.4 Picture definition in normalized device co-ordinates

- The transformation which map the viewing co-ordinate to normalized device co-ordinate is called **normalization transformation**. It involves scaling of x and y, thus it is also referred to as **scaling transformation**.

5.2.3 Window to Viewport Co-ordinate Transformation

- Once the window co-ordinates are transferred to viewing co-ordinates we choose the window extents in viewing co-ordinates and select the viewport limits in normalized co-ordinates. Object descriptions are then transferred to normalize device co-ordinates. In this transformation, the relative placement of the object in the normalized co-ordinates is same as in the viewing co-ordinates.
- Fig. 5.2.5 shows the mapping of object from window to viewport.
- A point at position (x_w, y_w) in the window is mapped into position (x_v, y_v) in the associated viewport.

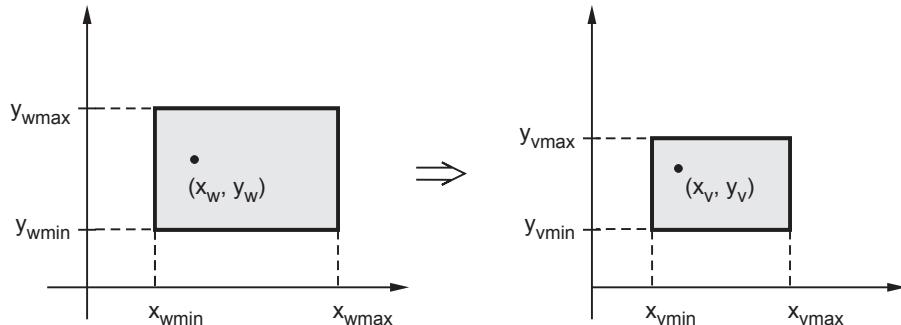


Fig. 5.2.5 Window to viewport mapping

- To maintain the same relative placement in the viewport as in the window, we require that

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}$$

$$\frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

Solving these equations for the viewport position (x_v, y_v) we have,

$$x_v = x_{vmin} + (x_w - x_{wmin}) s_x$$

$$y_v = y_{vmin} + (y_w - y_{wmin}) s_y$$

where scaling factors are

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

- The mapping of window to viewport can also be achieved by performing following transformations :
 1. By performing scaling transformation using a fixed-point position of (x_{wmin}, y_{wmin}) that scales the window area to the size of the viewport area.
 2. By translating the scaled window area to the position of the viewport area.
- While performing these transformations scaling factors s_x and s_y are kept same to maintain the relative proportions of objects.
- The character strings are transformed in two ways; they are either transformed without any change or if they are formed with line segments, they are transformed as a sequence of line transformations.

Workstation Transformation

- The transformation of object description from normalized co-ordinates to device co-ordinates is called **workstation transformation**.
- The workstation transformation is accomplished by selecting a window area in normalized space and a viewport area in the co-ordinates of the display device.
- By using workstation transformation we can partition a view so that different parts of normalized space can be displayed on different output devices, as shown in the Fig. 5.2.6. (See Fig. 5.2.6 on next page).
- The workstation transformation is achieved by performing following steps :
 1. The object together with its window is translated until the lower left corner of the window is at the origin.
 2. Object and window are scaled until the window has the dimensions of the viewport.
 3. Translate the viewport to its correct position on the screen.

This is illustrated in Fig. 5.2.7. (See Fig. 5.2.7 on page 5 - 9)

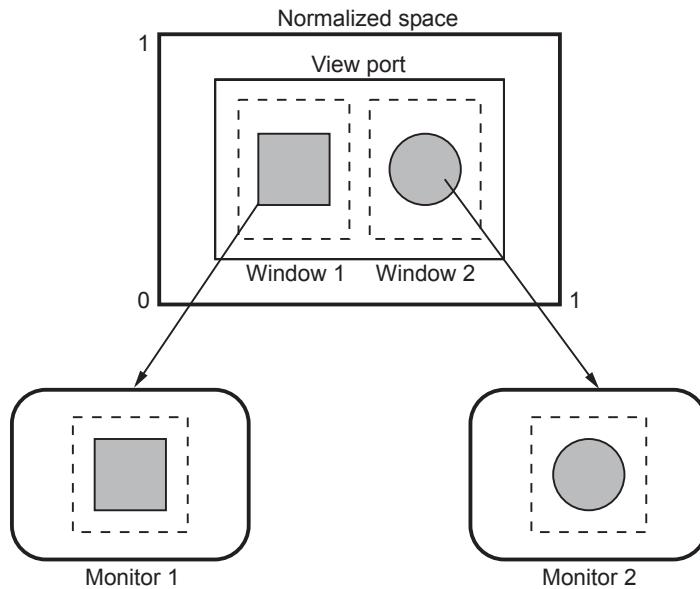


Fig. 5.2.6 Mapping selected portion of the scene in normalized co-ordinates to different monitors with workstation transformations

- The workstation transformation is given as

$$W = T \cdot S \cdot T^{-1} \quad \dots (5.2.3)$$

- The transformation matrices for individual transformation are as given below :

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w\min} & -Y_{w\min} & 1 \end{bmatrix}$$

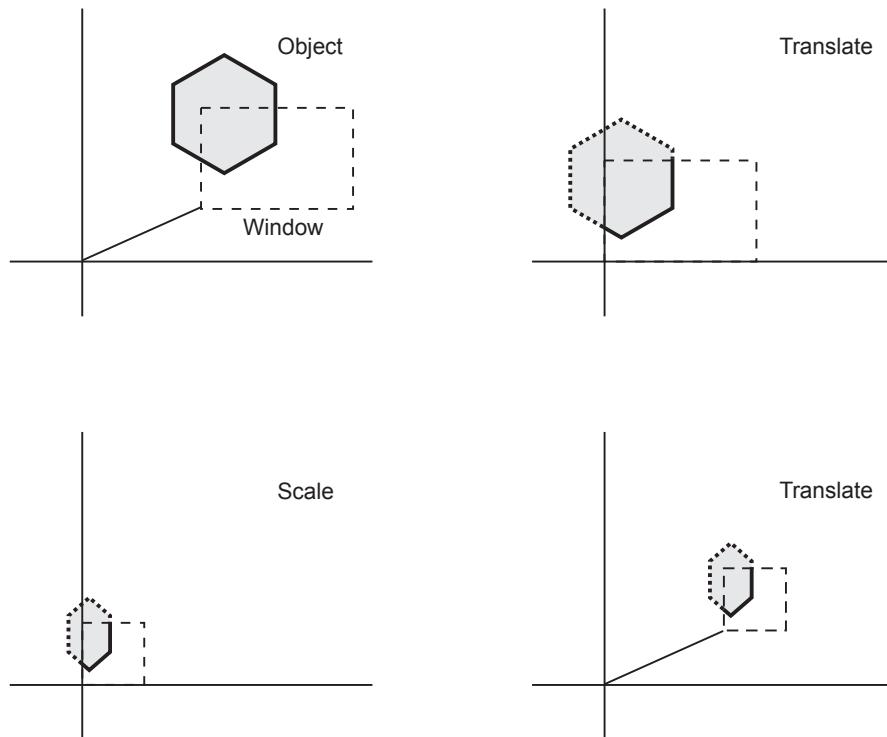
$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{where} \quad S_x = \frac{x_v \max - x_v \min}{x_w \max - x_w \min}$$

$$S_y = \frac{y_v \max - y_v \min}{y_w \max - y_w \min}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_{v\min} & Y_{v\min} & 1 \end{bmatrix}$$

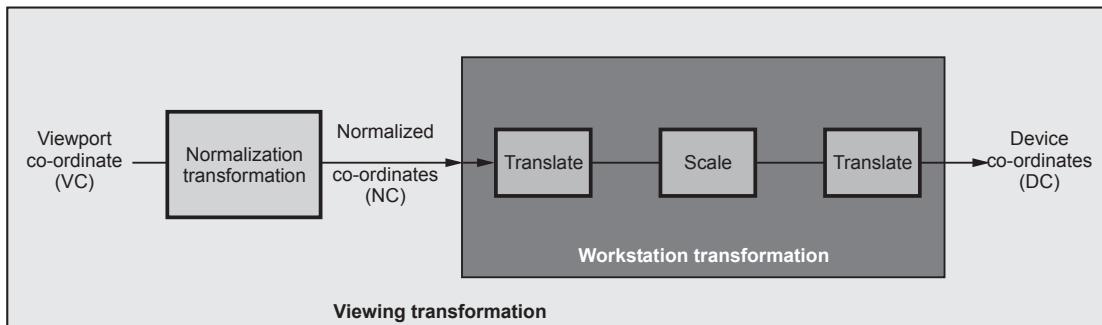
- The overall transformation matrix for W is given as

$$W = T \cdot S \cdot T^{-1}$$

**Fig. 5.2.7 Steps in workstation transformation**

$$\begin{aligned}
 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_{w\ min} & -Y_{w\ min} & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{v\ min} & y_{v\ min} & 1 \end{bmatrix} \\
 &= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v\ min} - X_{w\ min} \cdot S_x & y_{v\ min} - Y_{w\ min} \cdot S_y & 1 \end{bmatrix}
 \end{aligned}$$

- Fig. 5.2.8 shows the complete viewing transformation.

**Fig. 5.2.8 Viewing transformation**

Applications

- By changing the position of the viewport, we can view objects at different positions on the display area of an output device.
- By varying the size of viewports, we can change the size and proportions of displayed objects.
- We can achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport.

Example 5.2.1 Find the normalization transformation window to viewpoint, with window, lower left corner at (1, 1) and upper right corner at (3, 5) onto a viewpoint with lower left corner at (0, 0) and upper right corner at (1/2, 1/2).

Solution : Given : Co-ordinates for window

$$x_w \text{ min} = 1 \quad y_w \text{ min} = 1$$

$$x_w \text{ max} = 3 \quad y_w \text{ max} = 5$$

Co-ordinates for view port

$$x_v \text{ min} = 0 \quad y_v \text{ min} = 0$$

$$x_v \text{ max} = 1/2 = 0.5 \quad y_v \text{ max} = 1/2 = 0.5$$

We know that,

$$S_x = \frac{x_v \text{ max} - x_v \text{ min}}{x_w \text{ max} - x_w \text{ min}} = \frac{0.5 - 0}{3 - 1} = 0.25$$

and $S_y = \frac{y_v \text{ max} - y_v \text{ min}}{y_w \text{ max} - y_w \text{ min}} = \frac{0.5 - 0}{5 - 1} = 0.125$

We know that transformation matrix is given as,

$$\begin{aligned} T \cdot S \cdot T^{-1} &= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_v \text{ min} - x_w \text{ min} S_x & y_v \text{ min} - y_w \text{ min} S_y & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 - (1 \times 0.25) & 0 - (1 \times 0.125) & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ -0.25 & -0.125 & 1 \end{bmatrix} \end{aligned}$$

Example 5.2.2 Find the normalization transformation N that uses the rectangle $A(1, 1), B(5, 3), C(4, 5), D(0, 3)$ as a window and the normalized device screen as a viewport.

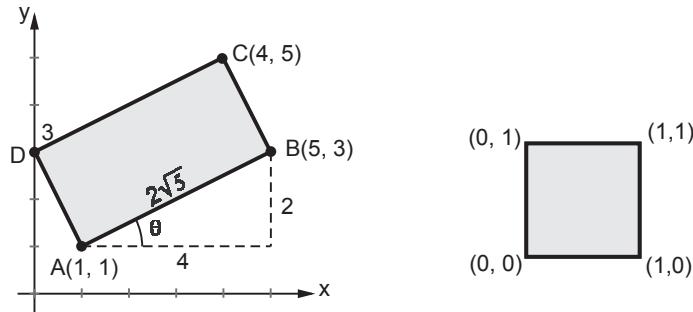
Solution : To align rectangle about A we have to rotate it with the co-ordinate axes. Then we can calculate, S_x and S_y and finally we compose the rotation and the transformation N to find the required normalization transformation N_R .

The slope of the line segment \overline{AB} is,

$$m = \frac{3-1}{5-1} = \frac{1}{2}$$

The Fig. 5.2.9 (a) shows the specified window and viewport. As shown in the Fig. 5.2.9 (a), the angle of rotation is $-\theta$ and it is given by,

$$\tan \theta = \frac{2}{4} = \frac{1}{2}$$



(a) Window

(b) Viewport

Fig. 5.2.9

Thus,

$$\sin \theta = \frac{1}{\sqrt{5}} \text{ and so } \sin(-\theta) = -\frac{1}{\sqrt{5}}, \quad \cos \theta = \frac{2}{\sqrt{5}}, \quad \cos(-\theta) = \frac{2}{\sqrt{5}}$$

Referring section 3.7.1.

The rotation matrix about A (1, 1) is given as,

$$R_{-\theta, A} = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{-1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ \left(1 - \frac{3}{\sqrt{5}}\right) & \left(1 - \frac{1}{\sqrt{5}}\right) & 1 \end{pmatrix}$$

The x extent of the rotated window is the length of \overline{AB} . Similarly, the y extent is the length of \overline{AD} . Using the distance formula we can calculate these length as,

$$d(A, B) = \sqrt{2^2 + 4^2} = \sqrt{20} = 2\sqrt{5}$$

$$d(A, D) = \sqrt{1^2 + 2^2} = \sqrt{5}$$

Also, the x extent of the normalized device screen is 1, as is the y extent. Calculating s_x and s_y ,

$$s_x = \frac{\text{Viewport x extent}}{\text{Window x extent}} = \frac{1}{2\sqrt{5}}$$

$$s_y = \frac{\text{Viewport y extent}}{\text{Window y extent}} = \frac{1}{\sqrt{5}}$$

$$\text{We have, } N = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ -s_x \times W_{\min} + x V_{\min} & -s_y \times W_{\min} + y V_{\min} & 1 \end{bmatrix}$$

where $x W_{\min} = 1$, $x V_{\min} = 0$, $y W_{\min} = 1$ and $y V_{\min} = 0$

$$= \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ \frac{-1}{2\sqrt{5}} \times 1 + 0 & \frac{-1}{\sqrt{5}} \times 1 + 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ \frac{-1}{2\sqrt{5}} & \frac{-1}{\sqrt{5}} & 1 \end{bmatrix}$$

The normalization transformation is then,

$$\begin{aligned} N_R = R_{-\theta A \cdot N} &= \begin{bmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ \left(1 - \frac{3}{\sqrt{5}}\right) & \left(1 - \frac{1}{\sqrt{5}}\right) & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ \frac{-1}{2\sqrt{5}} & \frac{-1}{\sqrt{5}} & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{5} & -\frac{1}{5} & 0 \\ \frac{1}{10} & \frac{2}{5} & 0 \\ \frac{-3}{10} & -\frac{1}{5} & 1 \end{bmatrix} \end{aligned}$$

Example 5.2.3 Find the complete viewing transformation that maps a window in world coordinates with x extent 1 to 10 and y extent 1 to 10 onto a viewport with x extent 1/4 to 3/4 and y extent 0 to 1/2 in normalized device space and then maps a workstation window with x extent 1/4 to 1/2 and y extent 1/4 to 1/2 in the normalized device space into a workstation viewport with x extent 1 to 10 and y extent 1 to 10 on the physical display device.

Solution : For normalization transformation

$$T = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ -s_x x_{wmin} + x_{vmin} & -s_y y_{wmin} + y_{vmin} & 1 \end{bmatrix}$$

The given parameters are $x_{wmin} = 1$, $x_{wmax} = 10$, $y_{wmin} = 1$,

$y_{wmax} = 10$, $x_{vmin} = 1/4$, $x_{vmax} = 3/4$, $y_{vmin} = 0$, and $y_{vmax} = \frac{1}{2}$

$$\therefore s_x = \frac{\text{Viewport } x \text{ extent}}{\text{Window } x \text{ extent}} = \frac{\frac{3}{4} - \frac{1}{4}}{10 - 1} = \frac{\frac{1}{2}}{9} = \frac{1}{18}$$

$$s_y = \frac{\text{Viewport } y \text{ extent}}{\text{Window } y \text{ extent}} = \frac{\frac{1}{2} - 0}{10 - 1} = \frac{\frac{1}{2}}{9} = \frac{1}{18}$$

Substituting values of s_x and s_y we have

$$T_1 = \begin{bmatrix} \frac{1}{18} & 0 & 0 \\ 0 & \frac{1}{18} & 0 \\ -\frac{1}{18} + \frac{1}{4} & -\frac{1}{18} + 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{18} & 0 & 0 \\ 0 & \frac{1}{18} & 0 \\ \frac{7}{36} & -\frac{1}{18} & 1 \end{bmatrix}$$

The given parameters for workstation transformation are $x_{wmin} = \frac{1}{4}$, $x_{wmax} = \frac{1}{2}$,

$y_{wmin} = \frac{1}{4}$, $y_{wmax} = \frac{1}{2}$, $x_{vmin} = 1$, $x_{vmax} = 10$, $y_{vmin} = 1$ and $y_{vmax} = 10$.

$$s_x = \frac{10 - 1}{\frac{1}{2} - \frac{1}{4}} = \frac{9}{\frac{1}{4}} = 36$$

$$s_y = \frac{10 - 1}{\frac{1}{2} - \frac{1}{4}} = \frac{9}{\frac{1}{4}} = 36$$

$$\therefore T_2 = \begin{bmatrix} 36 & 0 & 0 \\ 0 & 36 & 0 \\ -36 \cdot \frac{1}{4} + 1 & -36 \cdot \frac{1}{4} + 1 & 1 \end{bmatrix} = \begin{bmatrix} 36 & 0 & 0 \\ 0 & 36 & 0 \\ -8 & -8 & 1 \end{bmatrix}$$

The complete viewing transformation will be $T = T_1 \times T_2$

$$T = \begin{bmatrix} \frac{1}{18} & 0 & 0 \\ 0 & \frac{1}{18} & 0 \\ \frac{7}{36} & -\frac{1}{18} & 1 \end{bmatrix} \begin{bmatrix} 36 & 0 & 0 \\ 0 & 36 & 0 \\ -8 & -8 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -1 & -10 & 1 \end{bmatrix}$$

Review Questions

1. What is window and viewport ?
2. State the application of viewing transformation.
3. Describe viewing transformation.

SPPU : May-05, Marks 4

SPPU : May-05, Marks 4

SPPU : Dec.-10, 11,19, May-12, Marks 8

5.3 2 D Clipping

SPPU : May-07, 12

- The procedure that identifies the portions of a picture that are either inside or outside of a specified region of space is referred to as clipping.
- The region against which an object is to be clipped is called a **clip window** or **clipping window**. It usually is in a rectangular shape, as shown in the Fig. 5.3.1.
- The clipping algorithm determines which points, lines or portions of lines lie within the clipping window. These points, lines or portions of lines are retained for display. All others are discarded.

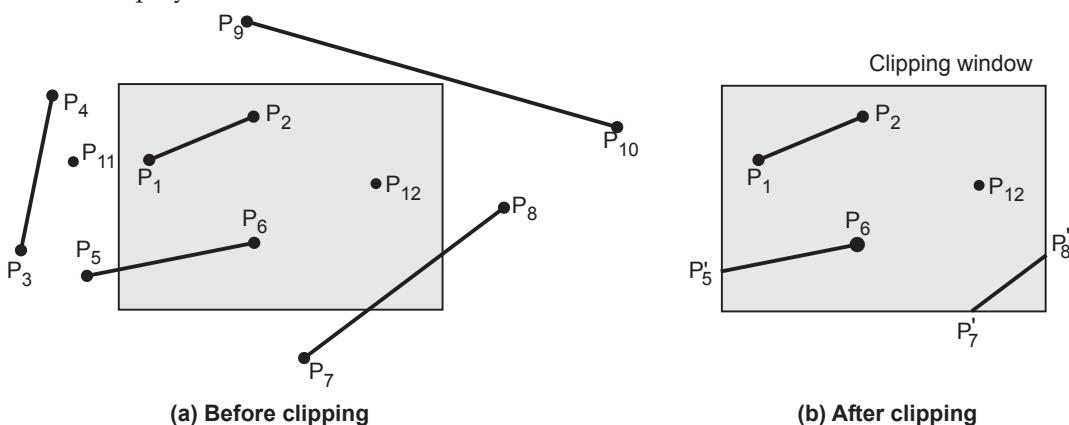


Fig. 5.3.1

5.3.1 Point Clipping

- The points are said to be interior to the clipping window if
$$x_{w \min} \leq x \leq x_{w \max} \quad \text{and} \quad y_{w \min} \leq y \leq y_{w \max}$$

The equal sign indicates that points on the window boundary are included within the window.

5.3.2 Line Clipping

- The lines are said to be interior to the clipping window and hence visible if both end points are interior to the window, e.g. line P₁ P₂ in Fig. 5.3.1.
- If both end points of a line are exterior to the window, the line is not necessarily completely exterior to the window, e.g. line P₇ P₈ in Fig. 5.3.1.
- If both end points of a line are completely to the right of, completely to the left of, completely above, or completely below the window, then the line is completely exterior to the window and hence invisible. For example, line P₃ P₄ in Fig. 5.3.1.
- The lines which cross one or more clipping boundaries require calculation of multiple intersection points to decide the visible portion of them.
- To minimize the intersection calculations and to increase the efficiency of the clipping algorithm, initially, completely visible and invisible lines are identified and then the intersection points are calculated for remaining lines.
- There are many line clipping algorithms. Let us discuss a few of them.

Review Questions

- What is point clipping*
- What is line clipping ?*

SPPU : May-12, Marks 2

5.4 Cohen-Sutherland Line Clipping Algorithm

SPPU : May-06,07,08,10,11,12,13,15,16,17,19, Dec.-10,15,18

- This is one of the oldest and most popular line clipping algorithm developed by Dan Cohen and Ivan Sutherland.
- To speed up the processing this algorithm performs initial tests that reduce the number of intersections that must be calculated.
- This algorithm uses a four digit (bit) code to indicate which of nine regions contain the end point of line.
- The four bit codes are called **region codes** or **outcodes**. These codes identify the location of the point relative to the boundaries of the clipping rectangle as shown in the Fig. 5.4.1.
- Each bit position in the region code is used to indicate one of the four relative co-ordinate positions of the point with respect to the clipping window : to the left, right, top or

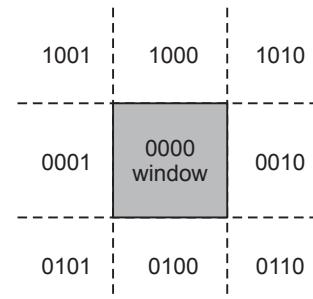


Fig. 5.4.1 Four-bit codes for nine regions

bottom. The rightmost bit is the first bit and the bits are set to 1 based on the following scheme :

- Set Bit 1 - if the end point is to the **left** of the window
- Set Bit 2 - if the end point is to the **right** of the window
- Set Bit 3 - if the end point is **below** the window
- Set Bit 4 - if the end point is **above** the window

Otherwise, the bit is set to zero.

- Once we have established region codes for all the line endpoints, we can determine which lines are completely inside the clipping window and which are clearly outside.
- Any lines that are completely inside the window boundaries have a region code of 0000 for both endpoints and we trivially accept these lines.
- Any lines that have a 1 in the same bit position in the region codes for each endpoint are completely outside the clipping rectangle, and we trivially reject these lines.
- A method used to test lines for total clipping is equivalent to the logical **AND** operator.
- If the result of the logical AND operation with two end point codes is not 0000, the line is completely outside the clipping region.
- The lines that cannot be identified as completely inside or completely outside a clipping window by these tests are checked for intersection with the window boundaries.

Example 5.4.1 Consider the clipping window and the lines shown in Fig. 5.4.2. Find the region codes for each end point and identify whether the line is completely visible, partially visible or completely invisible.

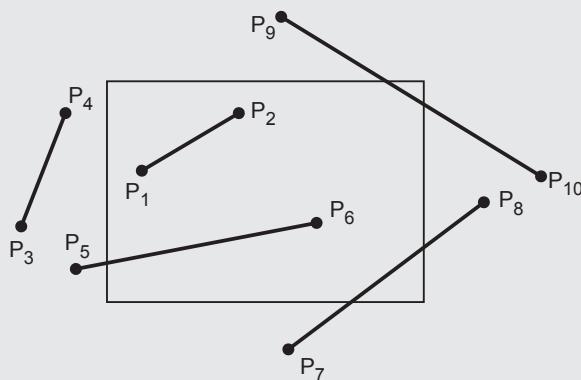


Fig. 5.4.2

Solution : The Fig. 5.4.3 shows the clipping window and lines with region codes. These codes are tabulated and end point codes are logically ANDed to identify the visibility of the line in Table 5.4.1.

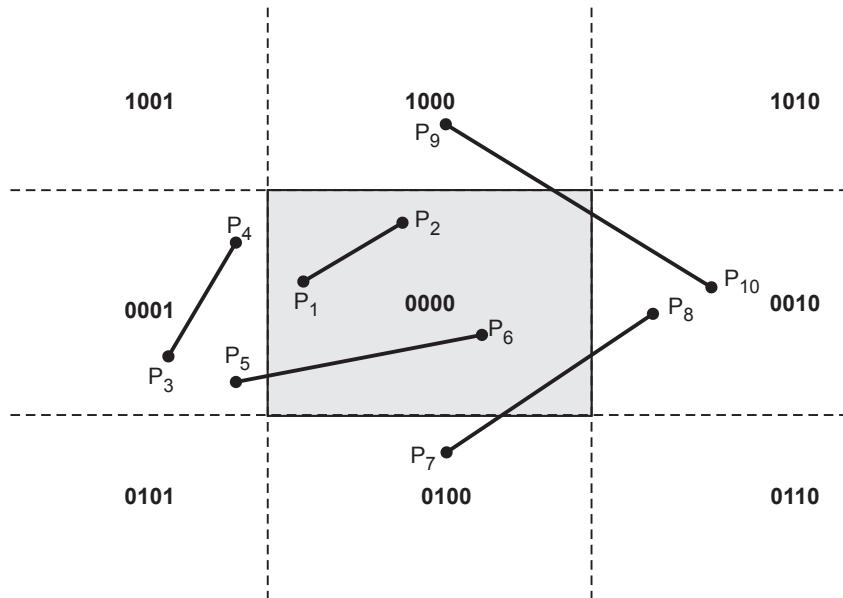


Fig. 5.4.3

Line	End point codes	Logical ANDing	Result
P ₁ P ₂	0000 0000	0000	Completely visible
P ₃ P ₄	0001 0001	0001	Completely invisible
P ₅ P ₆	0001 0000	0000	Partially visible
P ₇ P ₈	0100 0010	0000	Partially visible
P ₉ P ₁₀	1000 0010	0000	Partially visible

Table 5.4.1

- The Cohen-Sutherland algorithm begins the clipping process for a partially visible line by comparing an outside endpoint to a clipping boundary to determine how much of the line can be discarded. Then the remaining part of the line is checked against the other boundaries, and the process is continued until either the line is totally discarded or a section is found inside the window.

This is illustrated in Fig. 5.4.4.

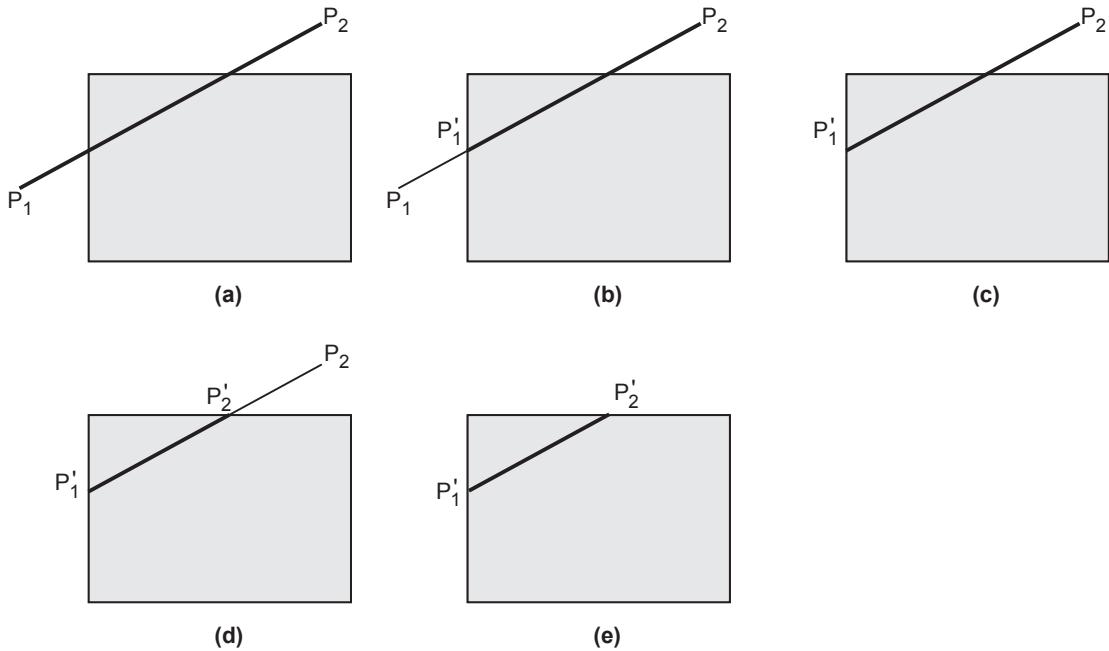


Fig. 5.4.4 Sutherland-Cohen subdivision line clipping

- As shown in the Fig. 5.4.4, line P_1P_2 is a partially visible and point P_1 is outside the window. Starting with point P_1 , the intersection point P'_1 is found and we get two line segments $P_1 - P'_1$ and $P'_1 - P_2$.
- We know that, for $P_1 - P'_1$ one end point i.e. P_1 is outside the window and thus the line segment $P_1 - P'_1$ is discarded.
- The line is now reduced to the section from P'_1 to P_2 . Since P_2 is outside the clip window, it is checked against the boundaries and intersection point P'_2 is found. Again the line segment is divided into two segments giving $P'_1 - P'_2$ and $P'_2 - P_2$. We know that, for $P'_2 - P_2$ one end point i.e. P_2 is outside the window and thus the line segment $P'_2 - P_2$ is discarded.
- The remaining line segment $P'_1 - P'_2$ is completely inside the clipping window and hence made visible.
- The intersection points with a clipping boundary can be calculated using the slope-intercept form of the line equation.
- The equation for line passing through points $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ is

$$y = m(x - x_1) + y_1 \quad \text{or} \quad y = m(x - x_2) + y_2 \quad \dots (5.4.1)$$

where $m = \frac{y_2 - y_1}{x_2 - x_1}$ (slope of the line)

- Therefore, the intersections with the clipping boundaries of the window are given as :

- Left : $x_L, y = m(x_L - x_1) + y_1 ; m \neq \infty$
- Right : $x_R, y = m(x_R - x_1) + y_1 ; m \neq \infty$
- Top : $y_T, x = x_1 + \left(\frac{1}{m}\right)(y_T - y_1) ; m \neq 0$
- Bottom : $y_B, x = x_1 + \left(\frac{1}{m}\right)(y_B - y_1) ; m \neq 0$

Sutherland and Cohen subdivision line clipping algorithm :

1. Read two end points of the line say $P_1 (x_1, y_1)$ and $P_2 (x_2, y_2)$.
2. Read two corners (left-top and right-bottom) of the window, say (Wx_1, Wy_1) and (Wx_2, Wy_2) .
3. Assign the region codes for two endpoints P_1 and P_2 using following steps :

Initialize code with bits 0000

```

Set Bit 1 - if (x < Wx1)
Set Bit 2 - if (x > Wx2)
Set Bit 3 - if (y < Wy2)
Set Bit 4 - if (y > Wy1)

```

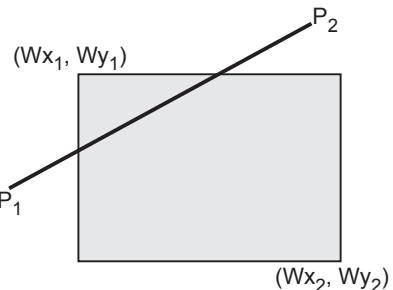


Fig. 5.4.5

4. Check for visibility of line $P_1 P_2$
 - a) If region codes for both endpoints P_1 and P_2 are zero then the line is completely visible. Hence draw the line and go to step 9.
 - b) If region codes for endpoints are not zero and the logical ANDing of them is also non-zero then the line is completely invisible, so reject the line and go to step 9.
 - c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Determine the intersecting edge of the clipping window by inspecting the region codes of two endpoints.
 - a) If region codes for both the end points are non-zero, find intersection points P'_1 and P'_2 with boundary edges of clipping window with respect to point P_1 and point P_2 , respectively
 - b) If region code for any one end point is non-zero then find intersection point P'_1 or P'_2 with the boundary edge of the clipping window with respect to it.

6. Divide the line segments considering intersection points.
7. Reject the line segment if any one end point of it appears outside the clipping window.
8. Draw the remaining line segments.
9. Stop.

Example 5.4.2 Use the Cohen-Sutherland outcode algorithm to clip two lines $P_1(40, 15) - P_2(75, 45)$ and $P_3(70, 20) - P_4(100, 10)$ against a window $A(50, 10), B(80, 10), C(80, 40), D(50, 40)$.

Solution : Line 1 : $P_1(40, 15)$ $P_2(75, 45)$ $x_L = 50$ $y_B = 10$ $x_R = 80$ $y_T = 40$

Point	Endcode	ANDing	Position
P_1	0 0 0 1	0 0 0 0	Partially visible
P_2	0 0 0 0		

$$x_L, y = m(x_L - x) + y_1 = \frac{6}{7}(50 - 40) + 15 \quad m = \frac{45 - 15}{75 - 40} = \frac{6}{7}$$

$$= 23.57$$

$$y_T, x = x_1 + \left(\frac{1}{m}\right)(y_T - y_1) = 40 + \left(\frac{7}{6}\right)(40 - 15)$$

$$= 69.16$$

$$I_1 = (50, 23.57) \quad I_2 = (69.06, 40)$$

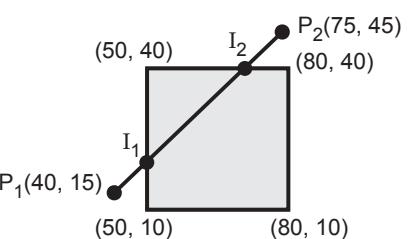


Fig. 5.4.6

Line 2 : $P_3(70, 20)$ $P_4(100, 10)$

Point	Endcode	ANDing	Position
P_3	0 0 0 0	0 0 0 0	Partially visible
P_4	0 0 1 0		

$$\text{Slope } m' = \frac{10 - 20}{100 - 70} = \frac{-10}{30} = \frac{-1}{3}$$

$$x_R, y = m(x_R - x_1) + y_1 = \frac{-1}{3}(80 - 70) + 20 = 16.66$$

$$I = (80, 16.66)$$

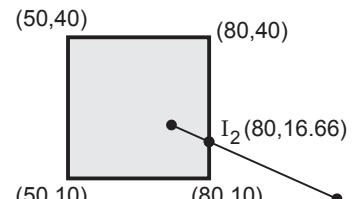


Fig. 5.4.6 (a)

Example 5.4.3 Use outcode based line clipping method to clip a line starting from $(-13, 5)$ and ending at $(17, 11)$ against the window having its lower left corner at $(-8, -4)$ and upper right corner at $(12, 8)$.

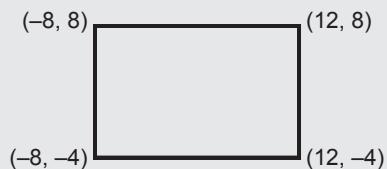


Fig. 5.4.7

Solution : $x_L = -8 \quad y_B = -4$
 $x_R = 12 \quad y_T = 8$

Given line : $P_1(-13, 5), P_2(17, 11)$

Point	Endcode	ANDing	Position
P_1	0 0 0 1	0 0 0 0	Partly visible
P_2	1 0 1 0		

Slope of line : $m = \frac{6}{30} = \frac{1}{5}$

$$\begin{aligned} x_L, y &= m(x_L - x) + y_1 = \frac{1}{5}(-8 - (-13)) + 5 \\ &= \frac{1}{5}(-8 + 13) + 5 = 1 + 5 = 6 \\ y_T, x &= x_1 + \frac{1}{m}(y_T - y) + x = -13 + 5(8 - 5) \\ &= -13 + 15 = 2 \end{aligned}$$

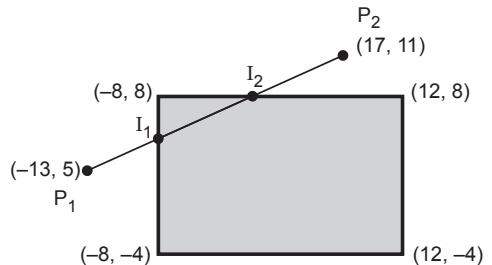


Fig. 5.4.7 (a)

$$I_1 = (x_L, x_L, y) = (-8, 6) \quad I_2 = (Y_{T,x}, Y_T) = (2, 8)$$

Example 5.4.4 Let R be the rectangular window whose lower left-hand corner is at $L(-3, 1)$ and upper right-hand corner is at $R(2, 6)$. If the line segment is defined with two end points with $A(-4, 2)$ and $B(-1, 7)$,

- The region codes of the two end points.
- Its clipping category and
- Stages in the clipping operations using Cohen-sutherland algorithm.

Solution :

a) Region of code for point A = 0001

Region of code for point B = 1000

b) Since (0001) AND (1000) = 0000 line segment AB is partially visible.

c) Stages in the clipping line segment AB are :

1. To push the 1 to 0 in code for A (0001), we clip against the window boundary line $x_{\min} = -3$.

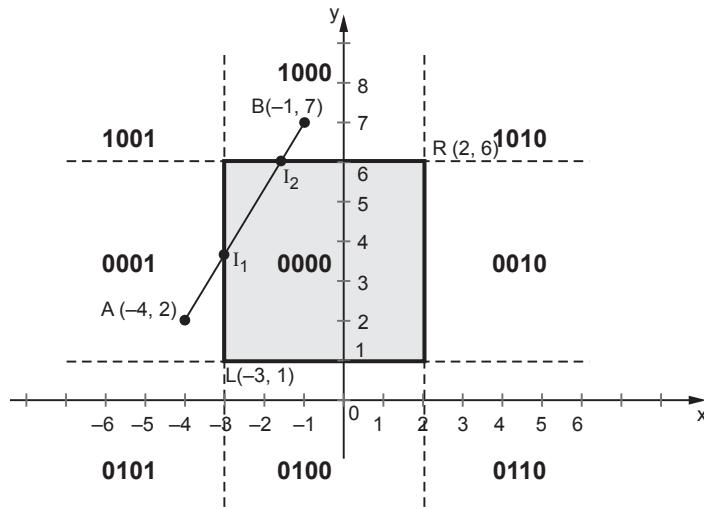


Fig. 5.4.8

$$2. \quad m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7 - 2}{-1 - (-4)} = \frac{5}{3}$$

3. We obtain the intersection point as,

$$I_1 = x_{\min}, \quad m(x_{\min} - x_1) + y_1 = -3, \quad \frac{5}{3}[-3 - (-4)] + 2$$

$$= -3, \frac{11}{3}$$

$$I_2 = x_1 + \left(\frac{1}{m}\right)(y_{\max} - y_1), \quad y_{\max} = -4 + \frac{3}{5}(6 - 2), 6$$

$$= \frac{-8}{5}, 6$$

4. Clip (do not display) segments AI₁ AI₂.

5. Display segment I₁I₂ since both end points lie in the window.

Example 5.4.5 Clip the line PQ having coordinates P(4, 1) and Q(6, 4) against the clip window having vertices A(3, 2), B(7, 2), C(7, 6) and D(3, 6) using Cohen Sutherland line clipping algorithm.

Solution : Line : A (4, 1), B (6, 4), $x_L = 3$, $y_B = 2$, $x_R = 7$, $y_T = 6$

Point	Endcode	ANDing	Position
P	0 1 0 0	0 0 0 0	Partially visible
Q	0 0 0 0		

$$\text{Line slope } m = \frac{4-1}{6-4} = \frac{3}{2}$$

$$y_{B,x} = x_1 + \left(\frac{1}{m} \right) (y_B - y_1)$$

$$= x_P + \left(\frac{1}{m} \right) (2 - y_P)$$

$$= 4 + \frac{2}{3}(2-1) = 4 + \frac{2}{3} = \frac{14}{3}$$

$$I = (y_{B,x}, y_B) = \left(\frac{14}{3}, 2 \right)$$

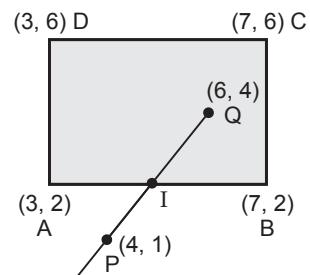


Fig. 5.4.9

Example for Practice

Example 5.4.6 : Find the clipping co-ordinates to clip the line segment $P_1 P_2$ against the window $ABCD$ using Cohen Sutherland line clipping algorithm
 $P_1 = (10, 30)$ and $P_2 = (80, 90)$ window $ABCD$ - A (20, 20), B (90, 20), C (90, 70) and D (20, 70).

Review Question

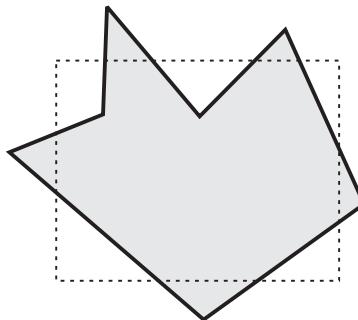
- Explain Cohen-Sutherland algorithm with the help of suitable example.

SPPU : May-06, 07, 08, 10, 11, 12, 13, 15, 16, 17, 19, Dec.-10, 15, 18, Marks 8

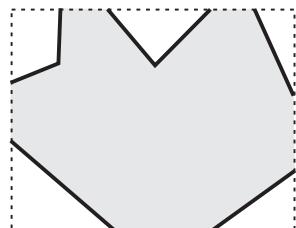
5.5 Polygon Clipping

SPPU : Dec.-05, 06, 07, 08, 11, 14, 17, 18, May-07, 10, 14

- A polygon is nothing but the collection of lines. Therefore, we might think that line clipping algorithm can be used directly for polygon clipping. However, when a closed polygon is clipped as a collection of lines with line clipping algorithm, the original closed polygon becomes one or more open polygon or discrete lines as shown in the Fig. 5.5.1. Thus, we need to modify the line clipping algorithm to clip polygons.
- We consider a polygon as a closed solid area. Hence after clipping it should remain closed. To achieve this we require an algorithm that



(a) Before clipping



(b) After clipping

Fig. 5.5.1 Polygon clipping done by line clipping algorithm

will generate additional line segment which make the polygon as a closed area.

- For example, in Fig. 5.5.2 the lines a - b, c - d, d - e, f - g, and h - i are added to polygon description to make it closed.

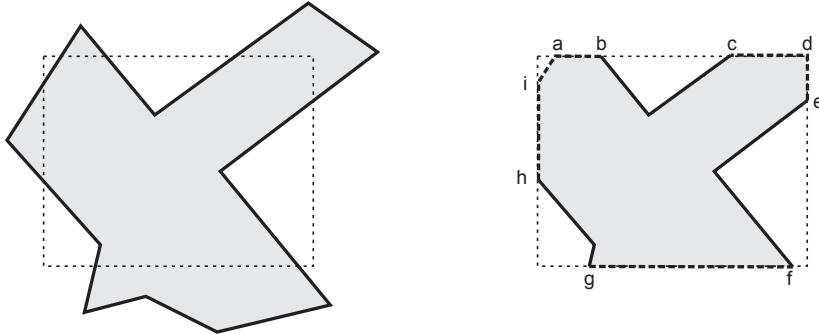


Fig. 5.5.2 Modifying the line clipping algorithm for polygon

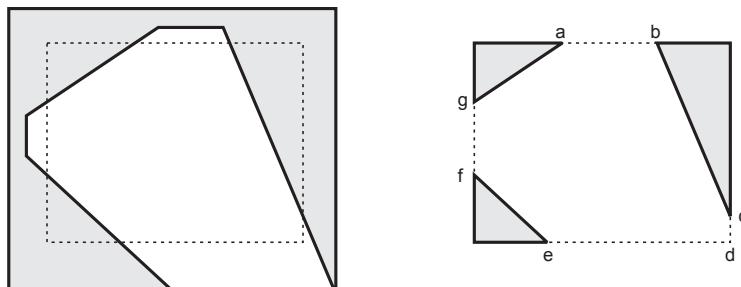


Fig. 5.5.3 Disjoint polygons in polygon clipping

- Adding lines c - d and d - e is particularly difficult. Considerable difficulty also occurs when clipping a polygon results in several disjoint smaller polygons as shown in the Fig. 5.5.3. For example, the lines a - b, c - d, d - e and g - f are frequently included in the clipped polygon description which is not desired.

5.5.1 Sutherland - Hodgeman Polygon Clipping

- A polygon can be clipped by processing its boundary as a whole against each window edge. This is achieved by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the original set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could then be successively passed to a right boundary clipper, a top boundary clipper and a bottom boundary clipper, as shown in Fig. 5.5.4. At each step a new set of polygon vertices is generated and passed to the next window boundary clipper. This is the fundamental idea used in the Sutherland - Hodgeman algorithm.

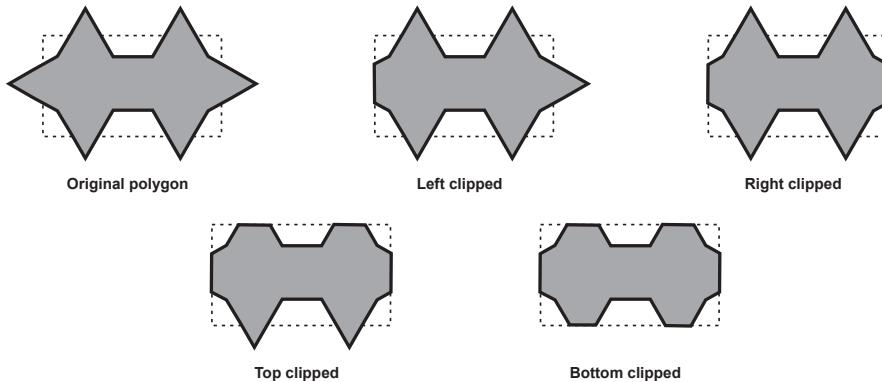


Fig. 5.5.4 Clipping a polygon against successive window boundaries

- The output of the algorithm is a list of polygon vertices all of which are on the visible side of a clipping plane. Such each edge of the polygon is individually compared with the clipping plane.

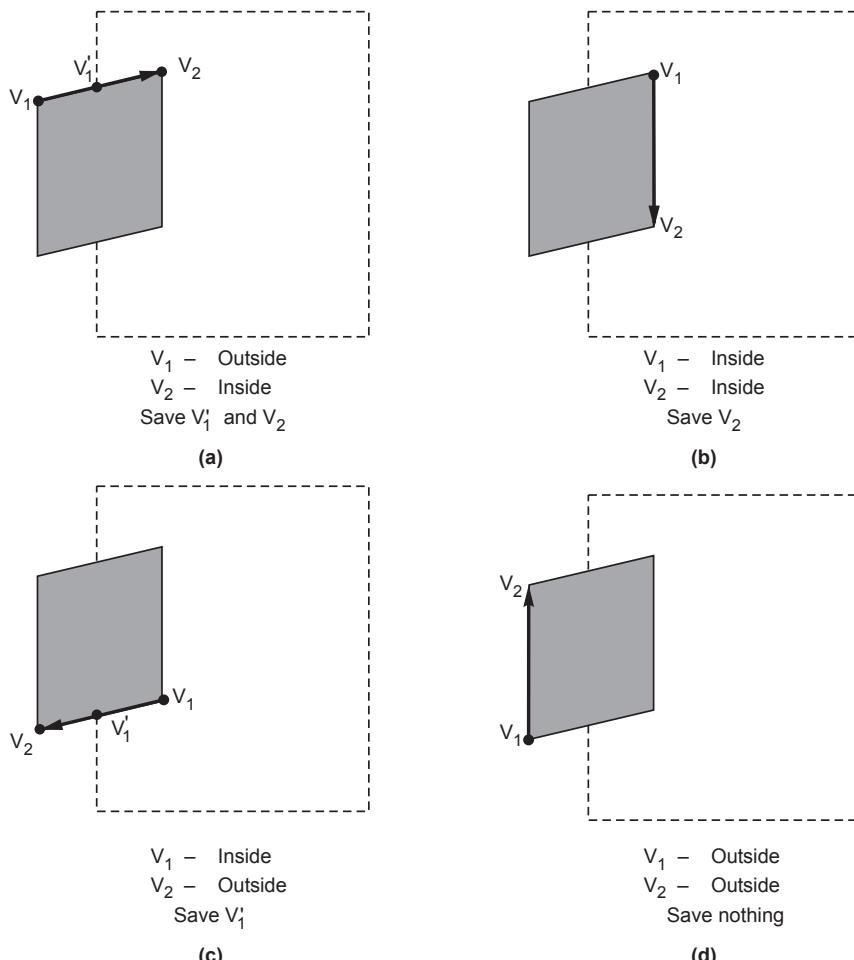


Fig. 5.5.5 Processing of edges of the polygon against the left window boundary

This is achieved by processing two vertices of each edge of the polygon around the clipping boundary or plane. This results in four possible relationships between the edge and the clipping boundary or plane. (See Fig. 5.5.5).

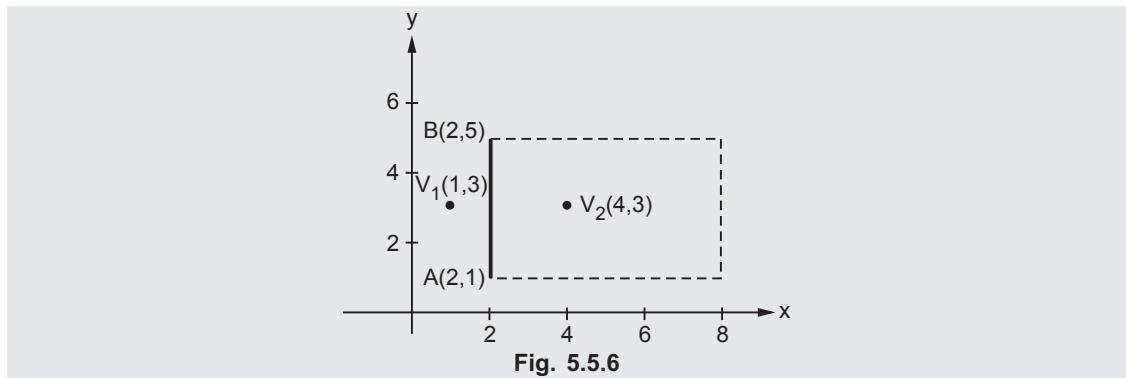
1. If the first vertex of the edge is outside the window boundary and the second vertex of the edge is inside then the intersection point of the polygon edge with the window boundary and the second vertex are added to the output vertex list (See Fig. 5.5.5 (a)).
 2. If both vertices of the edge are inside the window boundary, only the second vertex is added to the output vertex list. (See Fig. 5.5.5 (b)).
 3. If the first vertex of the edge is inside the window boundary and the second vertex of the edge is outside, only the edge intersection with the window boundary is added to the output vertex list. (See Fig. 5.5.5 (c)).
 4. If both vertices of the edge are outside the window boundary, nothing is added to the output list. (See Fig. 5.5.5 (d)).
- Once all vertices are processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.
 - Going through above four cases we can realize that there are two key processes in this algorithm.
 1. Determining the visibility of a point or vertex (Inside - Outside test) and
 2. Determining the intersection of the polygon edge and the clipping plane.
 - One way of determining the visibility of a point or vertex is described here.
 - Consider that two points A and B define the window boundary and point under consideration is V, then these three points define a plane.
 - Two vectors which lie in that plane are AB and AV. If this plane is considered in the xy plane, then the vector cross product $AV \times AB$ has only a z component given by $(x_V - x_A)(y_B - y_A) - (y_V - y_A)(x_B - x_A)$.
 - The sign of the z component decides the position of point V with respect to window boundary.

If z is : Positive – Point is on the **right side** of the window boundary

Zero – Point is **on** the window boundary

Negative – Point is on the **left side** of the window boundary

Example 5.5.1 Consider the clipping boundary as shown in the Fig. 5.5.6 and determine the positions of points V_1 and V_2 .



Solution : Using the cross product for V_1 we get,

$$\begin{aligned}
 & (x_V - x_A) (y_B - y_A) - (y_V - y_A) (x_B - x_A) \\
 &= (1 - 2) (5 - 1) - (3 - 1) (2 - 2) \\
 &= (-1) (4) - 0 \\
 &= -4
 \end{aligned}$$

The result of the cross product for V_1 is negative hence V_1 is on the left side of the window boundary.

Using the cross product for V_2 we get, $(4 - 2) (5 - 1) - (3 - 1) (2 - 2)$

$$\begin{aligned}
 &= (2) (4) - 0 \\
 &= 8
 \end{aligned}$$

The result of the cross product for V_2 is positive hence V_1 is on the right side of the window boundary.

The second key process in Sutherland - Hodgeman polygon clipping algorithm is to determine the intersection of the polygon edge and the clipping plane. Any of the line intersection (clipping) techniques discussed in the previous sections such as Cyrus-Beck or mid point subdivision can be used for this purpose.

Sutherland-Hodgeman Polygon Clipping Algorithm

1. Read coordinates of all vertices of the polygon.
2. Read coordinates of the clipping window
3. Consider the left edge of the window
4. Compare the vertices of each edge of the polygon, individually with the clipping plane

5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier.
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

Example 5.5.2 For a polygon and clipping window shown in Fig. 5.5.7 give the list of vertices after each boundary clipping.

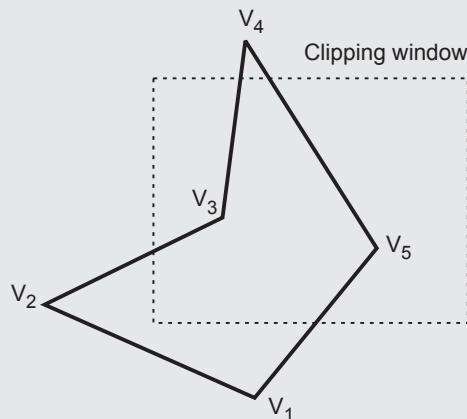


Fig. 5.5.7

Solution : Original polygon vertices are V_1, V_2, V_3, V_4, V_5 . After clipping each boundary the new vertices are given in Fig. 5.5.7 (a).

After left clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$

After right clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$

After top clipping : $V_1, V'_1, V'_2, V'_3, V'_3, V'_4, V_5$

After bottom clipping : $V''_2, V'_2, V_3, V'_3, V'_4, V_5, V'_5$

- The Sutherland-Hodgeman polygon clipping algorithm clips convex polygons correctly, but in case of concave polygons, clipped polygon may be displayed with extraneous lines, as shown in Fig. 5.5.8.

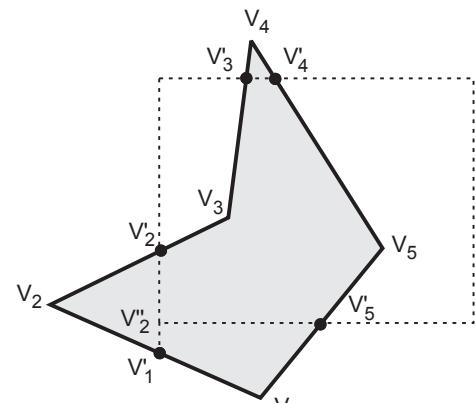


Fig. 5.5.7(a)

- The problem of extraneous lines for concave polygons in Sutherland-Hodgeman polygon clipping algorithm can be solved by separating concave polygon into two or more convex polygons and processing each convex polygon separately.

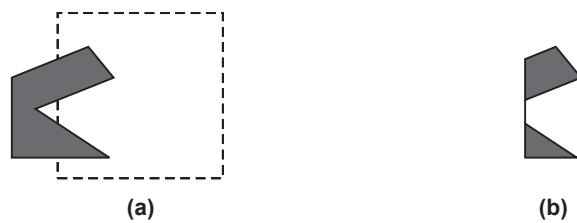


Fig. 5.5.8 Clipping the concave polygon in (a) with the Sutherland-Hodgeman algorithm produces the two connected areas in (b)

5.5.2 Weiler-Atherton Algorithm

The clipping algorithms previously discussed require a convex polygon. In context of many applications, e.g. hidden surface removal, the ability to clip to concave polygon is required. A powerful but somewhat more complex clipping algorithm developed by Weiler and Atherton meets this requirement. This algorithm defines the polygon to be clipped as a **subject polygon** and the clipping region is the clip polygon.

The algorithm describes both the subject and the clip polygon by a circular list of vertices. The boundaries of the subject polygon and the clip polygon may or may not intersect. If they intersect, then the intersections occur in pairs. One of the intersections occurs when a subject polygon edge enters the inside of the clip polygon and one when it leaves. As shown in the Fig. 5.5.9, there are four intersection vertices I₁, I₂, I₃ and I₄. In these intersections I₁ and I₃ are entering intersections, and I₂ and I₃ are leaving intersections. The clip polygon vertices are marked as C₁, C₂, C₃ and C₄.

In this algorithm two separate vertices lists are made one for clip polygon and one for subject polygon including intersection points. The Table 5.5.1 shows these two lists for polygons shown in Fig. 5.5.10.

The algorithm starts at an entering intersection (I₁) and follows the subject polygon vertex list in the downward

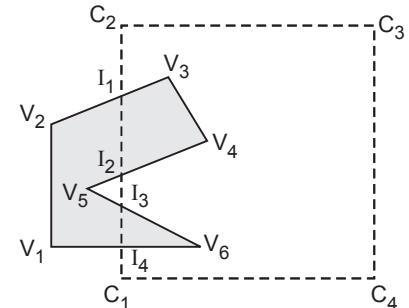


Fig. 5.5.9

	For subject polygon	For clip polygon
Start	V ₁	C ₁
	V ₂	I ₄
	I ₁	I ₃
	V ₃	Finish
	V ₄	I ₂
	I ₂	I ₁
	V ₅	C ₂
Start	I ₃	C ₃
	V ₆	C ₄
	I ₄	
	V ₁	

Table 5.5.1 List of polygon vertices

direction (i.e. I_1, V_3, V_4, I_2). At the occurrence of leaving intersection the algorithm follows the clip polygon vertex list from the leaving intersection vertex in the downward direction (i.e. I_2, I_1). At the occurrence of the entering intersection the algorithm follows the subject polygon vertex list from the entering intersection vertex. This process is repeated until we get the starting vertex. This process we have to repeat for all remaining entering intersections which are not included in the previous traversing of vertex list. In our example, entering vertex I_3 was not included in the first traversing of vertex list,. Therefore, we have to go for another vertex traversal from vertex I_3 .

The above two vertex traversals gives two clipped inside polygons. There are :

I_1, V_3, V_4, I_2, I_1 and I_3, V_6, I_4, I_3

5.5.3 Liang-Barsky Polygon Clipping

This section gives a brief outline of the Liang-Barsky polygon clipping algorithm. This algorithm is optimized for rectangular clipping window but is extensible to arbitrary convex windows. The algorithm is based on concepts from their two and three dimensional line clipping algorithm. The algorithm is claimed to be twice as fast as the Sutherland-Hodgeman clipping algorithm.

The Liang-Barsky algorithm assumes that the clipping region and polygon are coplanar. Each edge of the clipping window divides the plane into two half planes. The side of the half plane containing the window is called the **inside** or the **visible half**. The other half plane is the **outside** or **invisible half** plane. The four window edges divide the clipping plane into nine regions, as shown in the Fig. 5.5.10.

Entering and Leaving Vertices

Assume that $[P_i, P_{i+1}]$ is an edge of a polygon. If it is not horizontal or vertical then the infinite line, L_i , containing the polygon edge intersects each of the four window edges. In fact, as shown in the Fig. 5.5.11, such an infinite line always starts in a corner region, labelled 2, and ends in the diagonally opposite corner region, whether it intersects the window or not. For example, line A and C intersect the window while line B does not. As shown in the Fig. 5.5.11, each line has four window edge intersections. The first intersection of the infinite line, L_i , with a window edge represents a transition from an invisible (outside) to a visible (inside) side of the window edge. Such an intersection is called an **entering intersection**.

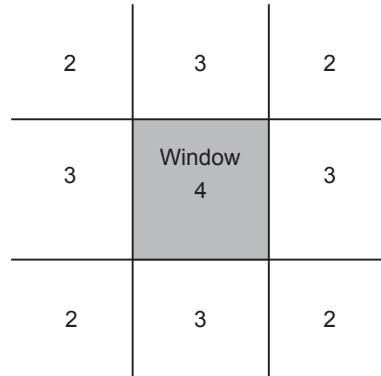


Fig. 5.5.10 Nine regions of clipping plane

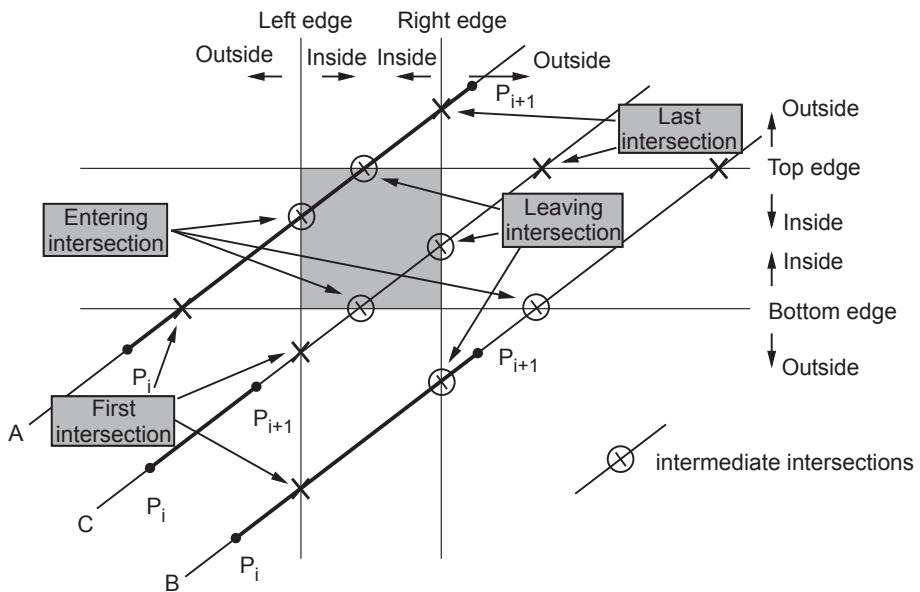


Fig. 5.5.11 Entering and leaving intersections

The last intersection of the infinite line, L_i , with a window edge represents a transition from a visible to an invisible side of the window edge and is called a **leaving intersection**. The two intermediate intersections can occur in either entering-leaving order as illustrated by the lines A and C, or in leaving-entering order, as illustrated by the line B in the Fig. 5.5.11.

If the intermediate intersections occur in entering-leaving order, then the infinite line, L_i , intersects the window. However, the visibility of any part or all of the polygon edge, $P_i P_{i+1}$, depends on the location of $P_i P_{i+1}$ along L_i , as shown by the polygon edges on lines A and C in Fig. 5.5.11.

If the intermediate intersections occur in leaving-entering order (as in case of line B), no part of the infinite line, L_i , is visible in the window.

Turning Vertices

If a subsequent polygon edge reenters the window through a different window edge, then it is necessary to include one or more of the window corners in the output

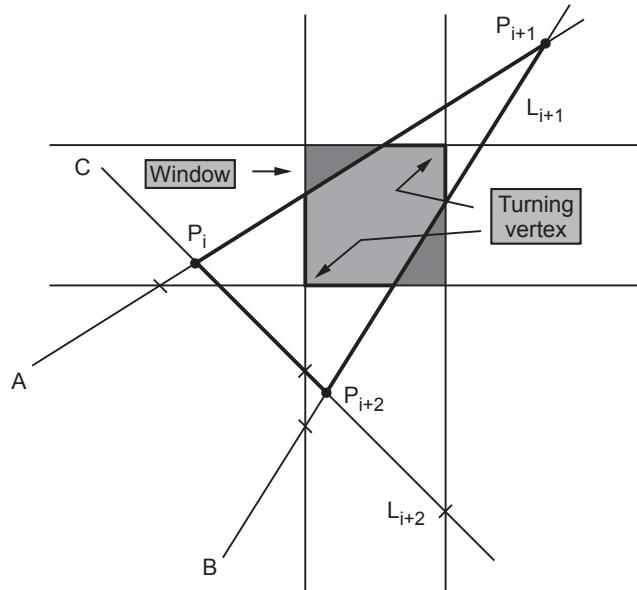


Fig. 5.5.12 Turning vertices

polygon. This is illustrated in Fig. 5.5.12. Liang and Barsky call such a window corner a **turning vertex**.

A necessary, but not sufficient, condition that a turning vertex exist if an entering intersection occurs on $P_{i+1} P_{i+2}$ and outside the window as shown in the Fig. 5.5.12. When this condition occurs, the Liang-Barsky polygon clipping algorithm adds the turning vertex closest to the entering vertex to the output polygon. This is a necessary but not sufficient condition, because, the entire polygon could lie outside the window. Thus, extraneous turning vertices can be added to the output polygon. This is the main drawback of the Liang-Barsky algorithm.

However, it is important to note that unless the first entering intersection is a window corner and hence coincident with the second entering intersection, and thus that the first entering intersection cannot be in the window, yields a sufficient condition for including a turning vertex in the output polygon.

It is possible to calculate the actual intersections of the Line L_i and the window edges, as well as the ordering of the intersections by using the parametric equation of the polygon edge.

$$P(t) = P_i + (P_{i+1} - P_i)t$$

where

$0 < t \leq 1$	represents the polygon edge except for the initial point, P_i , and
$-\infty < t < \infty$	represents the infinite line containing the polygon edge

Development of the Algorithm

Liang-Barsky algorithm assumes a regular clipping region and hence it can be developed by using the components of the parametric line, i.e.

$$x = x_i + (x_{i+1} - x_i)t = x_i + \Delta x t$$

$$y = y_i + (y_{i+1} - y_i)t = y_i + \Delta y t$$

For a regular clipping region, the edges are given by,

$$x_{\text{left}} \leq x \leq x_{\text{right}}$$

$$y_{\text{bottom}} \leq y \leq y_{\text{top}}$$

Using subscripts e and l to denote entering and leaving intersections, the parametric values for the four intersections with the window edges are

$$t_{xe} = (x_e - x_i)/\Delta x_i \quad \Delta x_i \neq 0$$

$$t_{xl} = (x_l - x_i)/\Delta x_i \quad \Delta x_i \neq 0$$

$$\begin{aligned}
 t_{ye} &= (y_e - y_i)/\Delta y_i & \Delta y_i \neq 0 \\
 t_{yl} &= (y_l - y_i)/\Delta y_i & \Delta y_i \neq 0 \\
 \text{where } x_e &= \begin{cases} x_{left} & \Delta x_i > 0 \\ x_{right} & \Delta x_i \leq 0 \end{cases} \\
 x_l &= \begin{cases} x_{right} & \Delta x_i > 0 \\ x_{left} & \Delta x_i \leq 0 \end{cases} \\
 y_e &= \begin{cases} y_{bottom} & \Delta y_i > 0 \\ y_{top} & \Delta y_i \leq 0 \end{cases} \\
 y_l &= \begin{cases} y_{top} & \Delta y_i > 0 \\ y_{bottom} & \Delta y_i \leq 0 \end{cases}
 \end{aligned}$$

The first and second entering and leaving intersections are given by

$$\begin{aligned}
 t_{e1} &= \min(t_{xe}, t_{ye}) \\
 t_{e2} &= \max(t_{xe}, t_{ye}) \\
 t_{l1} &= \min(t_{xl}, t_{yl}) \\
 t_{l2} &= \max(t_{xl}, t_{yl})
 \end{aligned}$$

Let us observe the conditions that decides whether to contribute, not to contribute, partly contribute, or to add turning vertex to the output polygon.

Conditions for no contribution to the output polygon

1. Termination of edge before the first entering intersection ($1 < t_{e1}$), see line A in Fig. 5.5.13.
2. Starting of edge after the first entering intersection ($0 \geq t_{e1}$) and ending of edge before the second entering intersection ($1 < t_{e2}$), see line B in Fig. 5.5.13.
3. Starting of edge after the second entering intersection ($0 \geq t_{e2}$) and after the first leaving intersection ($0 \geq t_{l1}$), see line C in Fig. 5.5.13.

Conditions for contribution to the output polygon

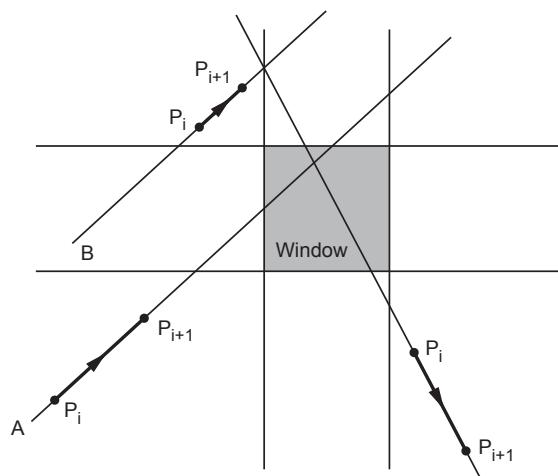


Fig. 5.5.13 Conditions for no contribution of the output polygon

4. If the second entering intersection occurs before the first leaving intersection ($t_{e2} \leq t_{l1}$), and P_i lies on L_i before the first leaving intersection ($0 < t_{l1}$), and P_{i+1} lies on L_i after the second entering intersection ($1 \geq t_{e2}$), the edge is either partially or wholly contained within the window and hence partially or wholly visible. See lines D, E and F in Fig. 5.5.14.

If the edge is partially visible, we can obtain the intersection point and contribution to the output polygon using parametric equation.

Conditions for addition of a turning vertex to the output polygon

5. If the first entering intersection lies on $P_i P_{i+1}$ ($0 < t_{e1} \leq 1$), then the turning vertex is (x_e, y_e) , e.g. line G in Fig. 5.5.15.
6. If the second entering intersection lies on $P_i P_{i+1}$ ($0 < t_{e2} \leq 1$), and is outside the window ($t_{l1} < t_{e2}$), then the turning vertex is either (x_e, y_e) when the second entering intersection lies on a vertical edge of the window ($t_{xe} > t_{ye}$), e.g. line H in the Fig. 5.5.15, or (x_e, y_e) when the second entering intersection lies on a horizontal edge of the window ($t_{ye} > t_{xe}$), e.g. line I in Fig. 5.5.15.

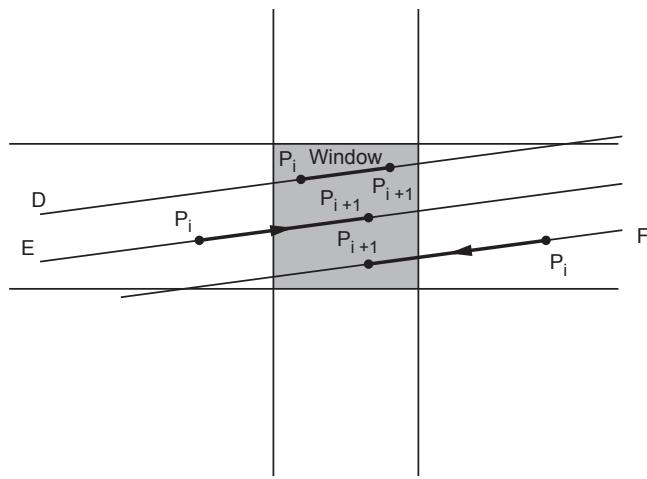


Fig. 5.5.14 Condition for contribution to the output polygon

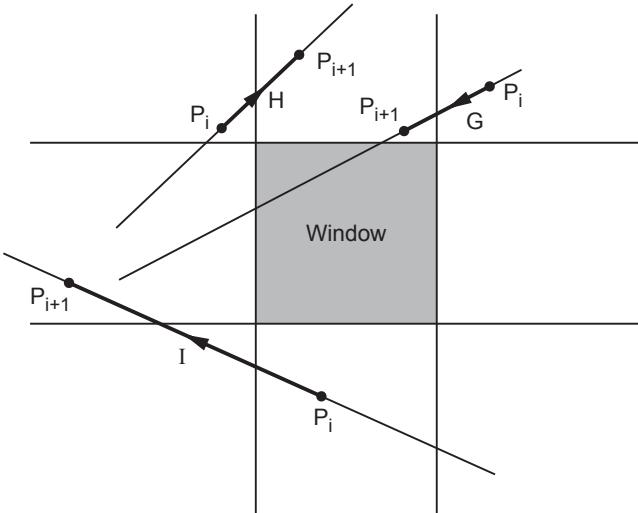


Fig. 5.5.15 Conditions for addition of a turning vertex

The Table 5.5.2 summarizes the six output conditions for polygon edges.

Sr.No.	Condition	Contribution
1.	$1 < t_{e1}$	No
2.	$0 \geq t_{e1}$ and $1 < t_{e2}$	No
3.	$0 \geq t_{e2}$ and $0 \geq t_{l1}$	No
4.	$t_{e2} \leq t_{l1}$ and $0 \leq t_{l1}$ and $1 \geq t_{e2}$	Visible segment
5.	$0 \leq t_{e1} \leq 1$	Turning vertex (x_e, y_e)
6.	$0 < t_{e2} \leq 1$ and $t_{l1} < t_{e2}$	Turning vertex (x_e, y_l) for $t_{xe} > t_{ye}$ (x_l, y_e) for $t_{ye} > t_{xe}$

Table 5.5.2 Output conditions for polygon edges

The conditions listed in the Table 5.5.2 are independent of the others except the condition 5. If condition 5 generates turning vertex, condition 4 may generate a visible segment or condition 6 may generate another turning vertex. If the polygon edge penetrates the window, a visible segment may be generated by condition 4; or if the polygon edge is completely outside the window, another turning vertex may be generated by condition 6. Therefore, in the algorithm we must evaluate the condition 5 prior to evolution of conditions 4 and 6.

Horizontal and vertical edges

Upto this point, we have ignored the horizontal and vertical polygon edges. These polygon edges are easily detected by looking for Δy or $\Delta x = 0$ respectively. For the horizontal line, $\Delta y = 0$ and $t_{ye} \rightarrow -\infty$ and $t_{yl} \rightarrow +\infty$. Thus, the horizontal line is characterized by

$$-\infty \leq t_{xe} \leq t_{xl} \leq +\infty$$

Similarly the vertical line is characterized by

$$-\infty \leq t_{ye} \leq t_{yl} \leq +\infty$$

Algorithm

For each polygon edge $P(i) P(i + 1)$

- Determine the direction of the edge and find whether it is diagonal, vertical or horizontal.
- Determine the order of the edge-window intersections.
- Determine the t-values of the entering edge-window intersections and the t-values for the first leaving intersection.

- Analyze the edge.

```

if  $t_1 \geq t_{\text{enter\_1}}$  then [condition 2 or 3 or 4 or 6 and not 1]
    if  $t_0 \geq t_{\text{enter\_1}}$  then [condition 5]
        call output (turning vertex);
    end if
    if  $t_1 \geq t_{\text{enter\_2}}$  then [condition 3 or 4 or 6 and not 1 or 2]
        [determine second leaving t-value there is output]
        if  $(0 < t_{\text{enter\_2}})$  or  $(0 < t_{\text{leave\_1}})$  then
            [condition 4 or 6, not 3]
            if  $t_{\text{enter\_2}} \leq t_{\text{leave\_1}}$  then [condition 4 - visible segment]
                call output (appropriate edge intersection)
            else [end condition 4 and begin condition 6-turning vertex]
                call output (appropriate turning vertex)
            end if [end condition 6]
        end if [end condition 4 or 6]
    end if [end condition 3 or 4 or 6]
    end if [end condition 2 or 3 or 4 or 6]
next i      end edge  $P[i]P[i + 1]$ 

```

Example 5.5.3 Find clipping co-ordinates for a line $p_1 p_2$ using Liang-Barsky algorithm where $p_1 = (50, 25)$ and $p_2 = (80, 50)$, against window with $(x_{w\min}, y_{w\min}) = (20, 10)$ and $(x_{w\max}, y_{w\max}) = (70, 60)$.

Solution : $x_1 = 50, y_1 = 25, x_2 = 80, y_2 = 50$

$$x_{\min} = 20, y_{\min} = 10, x_{\max} = 70, y_{\max} = 60$$

$$p_1 = -(x_2 - x_1) = -(80 - 50) = -30$$

$$p_2 = (x_2 - x_1) = (80 - 50) = 30$$

$$p_3 = -(y_2 - y_1) = -(50 - 25) = -25$$

$$p_4 = (y_2 - y_1) = (50 - 25) = 25$$

$$q_1 = (x_1 - x_{\min}) = (50 - 20) = 30$$

$$q_2 = (x_{\max} - x_1) = (70 - 50) = 20$$

$$q_3 = (y_1 - y_{\min}) = (25 - 10) = 15$$

$$q_4 = (y_{\max} - y_1) = (60 - 25) = 35$$

$$q_1/p_1 = 30/-30 = -1.000$$

$$q_2/p_2 = 20/30 = 0.667$$

$$q_3/p_3 = 15/-25 = -0.600$$

$$q_4/p_4 = 35/25 = 1.400$$

$$t_1 = \max(-1.000, -0.600) = 0.00 \quad \text{Since for these values } p < 0$$

$$t_2 = \min(0.667, 1.400) = 0.667 \quad \text{Since for these values } p > 0$$

Note : Minimum value of $t_1 = 0$

$$xx_1 = x_1 + t_1 * dx = 50 + 0.00 \times 30.00 = 50.00$$

$$yy_1 = y_1 + t_1 * dy = 25 + 0.00 \times 25.00 = 25.00$$

$$xx_2 = x_1 + t_2 * dx = 50 + 0.67 \times 30.00 = 70.00$$

$$yy_2 = y_1 + t_2 * dy = 25 + 0.67 \times 25.00 = 41.67$$

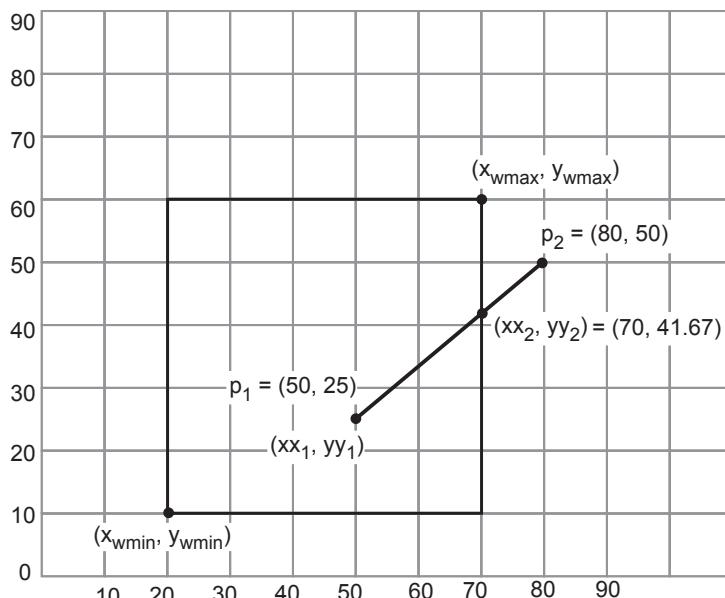


Fig. 5.5.16

Example 5.5.4 Find clipping co-ordinates for a line AB, where A = (5, 20) and B = (30, 30).

Co-ordinates of window are : $(x_{w\min}, y_{w\min}) = (15, 15)$ and $(x_{w\max}, y_{w\max}) = (45, 45)$.

Solve using Liang-Barsky line clipping algorithm.

Solution : $x_1 = 5, y_1 = 20, x_2 = 30, y_2 = 30$

$$x_{\min} = 15, y_{\min} = 15, x_{\max} = 45, y_{\max} = 45$$

$$p_1 = -(x_2 - x_1) = -(30 - 5) = -25$$

$$p_2 = (x_2 - x_1) = (30 - 5) = 25$$

$$p_3 = -(y_2 - y_1) = -(30 - 20) = -10$$

$$p_4 = (y_2 - y_1) = (30 - 20) = 10$$

$$q_1 = (x_1 - x_{\min}) = (5 - 15) = -10$$

$$q_2 = (x_{\max} - x_1) = (45 - 5) = 40$$

$$q_3 = (y_1 - y_{\min}) = (20 - 15) = 5$$

$$q_4 = (y_{\max} - y_1) = (45 - 20) = 25$$

$$q_1/p_1 = -10/-25 = 0.400$$

$$q_2/p_2 = 40/25 = 1.600$$

$$q_3/p_3 = 5/-10 = -0.500$$

$$q_4/p_4 = 25/10 = 2.500$$

$$t_1 = \max(0.400, -0.500) = 0.400 \quad \text{Since for these values } p < 0$$

$$t_2 = \min(1.600, 2.500) = 1.600 \quad \text{Since for these values } p > 0$$

Note : Maximum value of $t_2 = 1$

$$xx_1 = x_1 + t_1 * dx = 5 + 0.40 \times 25.00 = 15.00$$

$$yy_1 = y_1 + t_1 * dy = 20 + 0.40 \times 10.00 = 24.00$$

$$xx_2 = x_1 + t_2 * dx = 5 + 1.00 \times 25.00 = 30.00$$

$$yy_2 = y_1 + t_2 * dy = 20 + 1.00 \times 10.00 = 30.00$$

(See Fig. 5.5.17 on next page)

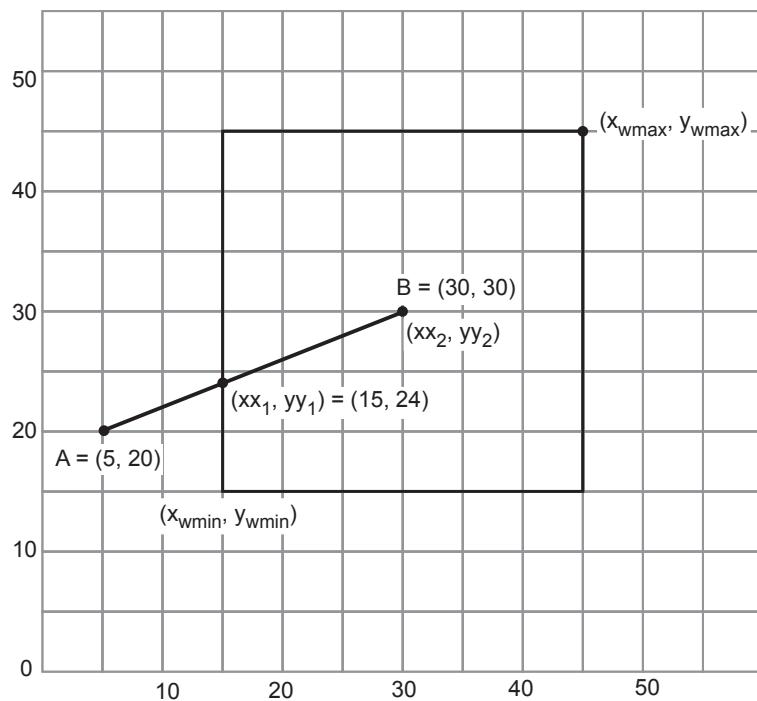


Fig. 5.5.17

Examples for Practice

Example 5.5.5 : Find clipping co-ordinates for a line AB, where A = (1, 3) and B = (5, 12). Co-ordinates of window are : $(x_{w\min}, y_{w\min}) = (2, 1)$ and $(x_{w\max}, y_{w\max}) = (9, 10)$. Solve using Liang-Barsky line clipping algorithm.

Example 5.5.6 : Find clipping co-ordinates for a line $p_1 p_2$ using Liang-Barsky algorithm where $p_1 = (90, 20)$ and $p_2 = (20, 90)$, against window with $(x_{w\min}, y_{w\min}) = (10, 10)$ and $(x_{w\max}, y_{w\max}) = (60, 60)$.

Example 5.5.7 : Find clipping co-ordinates for a line $p_1 p_2$ using Liang-Barsky algorithm where $p_1 = (60, 20)$ and $p_2 = (110, 40)$, against window with $(x_{w\min}, y_{w\min}) = (30, 10)$ and $(x_{w\max}, y_{w\max}) = (90, 50)$.

Example 5.5.8 : Find clipping co-ordinates for a line $p_1 p_2$ using Liang-Barsky algorithm where $p_1 = (5, -3)$ and $p_2 = (3, 4)$, against window with $(x_{w\min}, y_{w\min}) = (-1, 2)$ and $(x_{w\max}, y_{w\max}) = (6, 5)$.

Review Questions

1. Can line clipping algorithm be used for polygon clipping ? Justify ? **SPPU : Dec.-05, Marks 8**
2. Describe Sutherland - Hodgeman polygon clipping algorithm with example. **SPPU : Dec.-06, 07, 08, 11, 14, 17, 18, May-10, 14, Marks 8**
3. What is polygon clipping ? **SPPU : May-07, Marks 2**

5.6 Generalized Clipping**SPPU : Dec.-07, 11, 12, May-09**

- We have seen that in Sutherland - Hodgeman polygon clipping algorithm we need separate clipping routines, one for each boundary of the clipping window. But these routines are almost identical. They differ only in their test for determining whether a point is inside or outside the boundary.
- It is possible to generalize these routines so that they will be exactly identical and information about the boundary is passed to the routines through their parameters.
- Using recursive technique the generalized routine can be 'called' for each boundary of the clipping window with a different boundary specified by its parameters.
- This form of algorithm allows us to have any number of boundaries to the clipping window, thus the generalized algorithm with recursive technique can be used to clip a polygon along an arbitrary convex clipping window.

Review Question

1. Explain the concept of generalized clipping with the help of suitable example.

SPPU : Dec.-07, 11, 12, May-09, Marks 8**5.7 Interior and Exterior Clipping****SPPU : May-05, 13, Dec.-05, 12**

- So far we have discussed only algorithms for clipping point, line and polygon to the interior of a clipping region by eliminating every thing outside the clipping region. However, it is also possible to clip a point, line or polygon to the exterior of a clipping region, i.e., the point, portion of line and polygon which lie outside the clipping region. This is referred to as **exterior clipping**.
- Exterior clipping is important in a multiwindow display environment, as shown in Fig. 5.7.1.
- Fig. 5.7.1 shows the overlapping windows with window 1 and window 3 having priority over window 2. The objects within the window are clipped to the interior of that window.

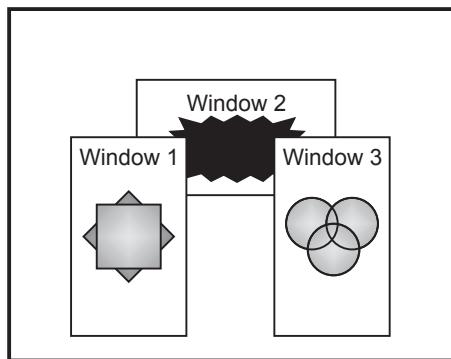


Fig. 5.7.1 Clipping in multiwindow environment

- When other higher-priority windows such as window 1 and/or window 3 overlap these objects, the objects are also clipped to the exterior of the overlapping windows.

Review Question

1. *What is interior and exterior clipping.*

SPPU : May-05, 13, Dec.-05, 12, Marks 4



Notes

SUBJECT CODE : 210244

As per Revised Syllabus of
SAVITRIBAI PHULE PUNE UNIVERSITY
Choice Based Credit System (CBCS)
S.E. (Computer) Semester - I

COMPUTER GRAPHICS

(For END SEM Exam - 70 Marks)

Atul P. Godse

M.S. Software Systems (BITS Pilani)

B.E. Industrial Electronics

Formerly Lecturer in Department of Electronics Engg.

Vishwakarma Institute of Technology

Pune

Dr. Deepali A. Godse

M.E., Ph.D. (Computer Engg.)

Head of Information Technology Department,
Bharati Vidyapeeth's College of Engineering for Women,
Pune

Dr. Rajesh Prasad

Ph.D. (Computer Science & Engg.)

Professor (Department of Computer Engineering),
Sinhgad Institute of Technology & Science,
Narhe, Pune.



COMPUTER GRAPHICS

(For END SEM Exam - 70 Marks)

Subject Code : 210244

S.E. (Computer) Semester - I

First Edition : September 2020

© Copyright with A. P. Godse and Dr. D. A. Godse

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :

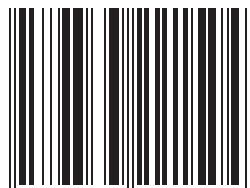


Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :

Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-93-332-2150-4



9 789333 221504

SPPU 19

PREFACE

The importance of **Computer Graphics** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Computer Graphics**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors

A.P. Godse

Dr. D. A. Godse

Dr. Rajesh Prasad

Dedicated to God

SYLLABUS

Computer Graphics - 210244

Credit Scheme	Examination Scheme and Marks
03	End_Semester (TH) : 70 Marks

Unit III 2D, 3D Transformations and Projections

2-D transformations : Introduction, homogeneous coordinates, 2-D transformations-Translation, scaling, rotation and shear, rotation about an arbitrary point.

3-D transformations : introduction, 3-D transformations - Translation, scaling, rotation and shear, rotation about an arbitrary axis.

Projections : Parallel (Oblique: Cavalier, Cabinet and orthographic: isometric, diametric, trimetric) and Perspective (Vanishing Points - 1 point, 2 point and 3 point) (**Chapters - 6, 7**)

Unit IV Light, Colour, Shading and Hidden Surfaces

Colour models : Properties of Light, CIE chromaticity Diagram, RGB, HSV, CMY.

Illumination Models : Ambient Light, Diffuse reflection, Specular Reflection, and the Phong model, Combined diffuse and Specular reflections with multiple light sources, warn model,

Shading Algorithms : Halftone, Gouraud and Phong Shading.

Hidden Surfaces : Introduction, Back face detection and removal, Algorithms: Depth buffer (z), Depth sorts (Painter), Area subdivision (Warnock) (**Chapters - 8, 9, 10**)

Unit V Curves and Fractals

Curves : Introduction, Interpolation and Approximation, Blending function, 8-Spline curve, Bezier curve,

Fractals : Introduction, Classification, Fractal generation: snowflake, Triadic curve, Hilbert curve, Applications. (**Chapter - 11**)

Unit VI Introduction to Animation and Gaming

Segment : Introduction, Segment table, Segment creation, closing, deleting and renaming. Visibility.

Animation : Introduction, Conventional and computer based animation, Design of animation sequences, Animation languages, Key- frame, Morphing, Motion specification.

Gaming : Introduction, Gaming platform (NVIDIA, i8060), Advances in Gaming.
(Chapters - 12, 13, 14)

TABLE OF CONTENTS

Unit - III

Chapter - 6 2D Transformations	(6 - 1) to (6 - 50)
6.1 Introduction	6 - 2
6.2 Transformations and Matrices.....	6 - 2
6.3 Transformation Conventions	6 - 4
6.4 Two - dimensional Transformations	6 - 5
6.4.1 Translation	6 - 5
6.4.2 Rotation	6 - 6
6.4.3 Scaling.....	6 - 7
6.5 Homogeneous Co-ordinates	6 - 9
6.5.1 Homogeneous Co-ordinates for Translation	6 - 10
6.5.2 Homogeneous Co-ordinates for Rotation	6 - 11
6.5.3 Homogeneous Co-ordinates for Scaling.....	6 - 11
6.6 Composite Transformations	6 - 14
6.6.1 Rotation about an Arbitrary Point	6 - 14
6.7 Reflection and Shear Transformations	6 - 33
6.7.1 Reflection	6 - 34
6.7.2 Shear	6 - 35
6.7.2.1 X Shear	6 - 35
6.7.2.2 Y Shear	6 - 36
6.7.2.3 Shearing Relative to Other Reference Line	6 - 36
6.8 Inverse Transformation.....	6 - 49

Chapter - 7 3D Transformations and Projections (7 - 1) to (7 - 40)

7.1 Introduction	7 - 2
7.2 3D Geometry.....	7 - 2
7.3 Primitives	7 - 5
7.4 Translation	7 - 7
7.5 Scaling	7 - 8
7.6 Rotation	7 - 9
7.7 Reflection	7 - 11
7.8 Shears	7 - 12
7.9 Rotation about Arbitrary Axis	7 - 12
7.10 Reflection with Respect to Given Plane.....	7 - 18
7.10.1 Reflection with respect to xy Plane	7 - 18
7.10.2 Reflection with respect to Any Plane.....	7 - 18
7.11 Concept of Parallel and Perspective Projections	7 - 20
7.11.1 Parallel Projection	7 - 21
7.11.2 Perspective Projection.....	7 - 21
7.12 Types of Parallel Projections.....	7 - 22
7.12.1 Orthographic Projection	7 - 22
7.12.1.1 Axonometric Orthographic Projection	7 - 22
7.12.1.2 Types of Axonometric Projection	7 - 23
7.12.1.3 Multiview Orthographic Projection	7 - 27
7.12.2 Oblique Projection.....	7 - 28
7.13 Types of Perspective Projections	7 - 30
7.13.1 One-Point Perspective Projection	7 - 30
7.13.2 Two-Point Perspective Projection	7 - 31
7.13.3 Three-Point Perspective Projection	7 - 32
7.14 Summary of Various Types of Projections	7 - 32

Unit - IV

Chapter - 8 Colour Models	(8 - 1) to (8 - 10)
8.1 Introduction	8 - 2
8.2 Properties of Light.....	8 - 2
8.3 CIE Chromaticity Diagram	8 - 3
8.4 RGB	8 - 6
8.5 HSV.....	8 - 7
8.6 CMY	8 - 8
<hr/>	
Chapter - 9 Illumination Models and Shading Algorithms	(9 - 1) to (9 - 16)
9.1 Introduction	9 - 2
9.2 Light Sources.....	9 - 2
9.3 Ambient Light	9 - 3
9.4 Diffuse Illumination and Reflection	9 - 4
9.5 Specular Reflection	9 - 6
9.6 Phong Model.....	9 - 7
9.7 Combined Diffuse and Specular Reflections with Multiple Light Sources	9 - 8
9.8 Warn Model.....	9 - 9
9.9 Shading Algorithms.....	9 - 9
9.9.1 Constant-Intensity Shading	9 - 9
9.9.2 Gouraud Shading	9 - 10
9.9.3 Phong Shading.....	9 - 13
9.9.4 Halftone Shading.....	9 - 15
<hr/>	
Chapter - 10 Hidden Surfaces	(10 - 1) to (10 - 12)
10.1 Introduction	10 - 2
10.2 Back Face Detection and Removal Algorithm.....	10 - 2

10.3 Depth Buffer (Z) Algorithm	10 - 4
10.4 Depth Sorts (Painter) Algorithm	10 - 6
10.5 Area Subdivision (Warnock) Algorithm.....	10 - 9

Unit - V

Chapter - 11 Curves and Fractals	(11 - 1) to (11 - 34)
-----------------------------------------	------------------------------

11.1 Introduction	11 - 2
11.2 Curve Generation.....	11 - 2
11.2.1 Circular Arc Generation using DDA Algorithm.	11 - 2
11.3 Interpolation, Approximation and Blending Function	11 - 4
11.4 Interpolating Polygons.....	11 - 8
11.5 Spline Representations	11 - 9
11.5.1 Interpolation and Approximation.	11 - 9
11.5.2 Geometric and Parametric Continuity.....	11 - 10
11.5.3 Spline Specifications	11 - 11
11.6 Bezier Curves	11 - 12
11.7 B-Spline Curve.....	11 - 20
11.8 Fractals	11 - 26
11.8.1 Classification of Fractals	11 - 27
11.8.2 Topological Dimension	11 - 27
11.8.3 Fractal Dimension	11 - 28
11.8.4 Hilbert's Curve	11 - 29
11.8.5 Koch Snow Flake Curve/Triadic Cuve.....	11 - 31

Unit - VI

Chapter - 12 Segment	(12 - 1) to (12 - 14)
-----------------------------	------------------------------

12.1 Introduction.....	12 - 2
12.2 Segment Table	12 - 2

12.3 Functions for Segmenting the Display File.....	12 - 5
12.3.1 Segment Creation	12 - 5
12.3.2 Closing a Segment	12 - 6
12.3.3 Deleting a Segment	12 - 7
12.3.4 Renaming a Segment	12 - 8
12.4 More about Segments	12 - 9
12.5 Display File Structures	12 - 10
12.6 Image Transformation	12 - 12

Chapter - 13 Animation	(13 - 1) to (13 - 16)
-------------------------------	------------------------------

13.1 Introduction.....	13 - 2
13.2 Conventional and Computer Based Animation	13 - 2
13.2.1 Conventional Animation	13 - 2
13.2.2 Computer based Animation.....	13 - 4
13.3 Design of Animation Sequences	13 - 5
13.4 Animation Languages	13 - 6
13.4.1 Linear-List Notations	13 - 6
13.4.2 General-Purpose Languages	13 - 6
13.4.3 Graphical Languages	13 - 7
13.5 Keyframes and Morphing	13 - 8
13.5.1 Morphing	13 - 8
13.5.2 Tweening.....	13 - 13
13.6 Motion Specification.....	13 - 14

Chapter - 14 Gaming	(14 - 1) to (14 - 10)
----------------------------	------------------------------

14.1 Introduction.....	14 - 2
14.2 Gaming Platforms	14 - 2
14.2.1 NVIDIA GPU.....	14 - 2
14.2.1.1 Connection between CPU and GPU.	14 - 3
14.2.1.2 GPU Architecture	14 - 4

14.2.1.3 Features of NVIDIA Gaming Platform	14 - 5
14.2.2 i860	14 - 5
14.2.2.1 Features	14 - 5
14.2.2.2 Block Diagram.	14 - 7
14.3 Advances in Gaming	14 - 9

UNIT - III

6

2D Transformations

Syllabus

Introduction, homogeneous coordinates, 2-D transformations-Translation, scaling, rotation and shear, rotation about an arbitrary point.

Contents

6.1	<i>Introduction</i>	
6.2	<i>Transformations and Matrices</i>	
6.3	<i>Transformation Conventions</i>	
6.4	<i>Two - dimensional Transformations</i>	Dec.-07,09,14, May-12,14,15, Marks 8
6.5	<i>Homogeneous Co-ordinates</i>	Dec.-05,10,12,18 May-07,12,16 Marks 10
6.6	<i>Composite Transformations</i>	Dec.-05,06,07,09,10,11,15,16,17,19 May-06,07,08,11, 13,15,16,17, 18,19 Marks 18
6.7	<i>Reflection and Shear Transformations</i>	May-07,08,09,12,14,16,19 Dec.-08,12,14 Marks 8
6.8	<i>Inverse Transformation</i>	Dec.-06,18, May-07, 08, . . . Marks 8

6.1 Introduction

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

In this chapter we discuss the 2D transformations.

6.2 Transformations and Matrices

- In computer graphics images are generated from a series of line segments which are represented by the co-ordinates of their end points. Due to this we can make changes in an image by performing mathematical operations on these co-ordinates.
- When we change the image we call image is transformed.
- Before going to study various image transformations let us review some of the mathematical tools we shall need, namely matrix multiplication.
- We know that matrix is a two dimensional array of number. For example,

$$\left| \begin{array}{cc} 0 & 1 \\ 2 & 3 \end{array} \right| \quad \left| \begin{array}{cccc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{array} \right| \quad \left| \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \right| \quad \left| \begin{array}{ccc} 0 & -1 & 3 \\ 0 & 1 & 2 \\ 5 & 4 & 2 \end{array} \right|$$

are four different matrices.

- The elements in the matrix are identified by specifying the row and column number. Suppose matrix A is

$$A = \left| \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{array} \right|$$

- Then the element 8 is identified as $A(3, 2)$, i.e. element 8 is in the 3rd row and 2nd column of the matrix A.

Matrix Multiplication

- Matrix multiplication is more complex than the simple product of two numbers; it involves simple products and sums of the matrix elements.
- We can multiply two matrices A and B together only if the number of column of the first matrix A is the same as the number of rows of the second matrix B. If B matrix is

$$B = \begin{bmatrix} 1 & 2 \\ -2 & 0 \\ 3 & 1 \end{bmatrix}$$

then we can multiply matrix A and B because matrix A has 3 columns and matrix B has 3 rows.

- Unlike multiplication of numbers, the multiplication of matrix is not commutative; that is, while we can multiply A times B, we cannot multiply B times A, because B has 2 columns and A has 3 rows. When we multiply two matrices we get resultant matrices such as it has same number of rows as the first matrix of the two being multiplied and the same number of columns as the second matrix. Therefore, multiplication of A and B gives 3×2 resultant C matrix.
- The generalized formula for matrix multiplication is given by,

$$C(i,k) = \sum_j A(i,j) B(j,k)$$

- In our example, the element C (1, 1) is found by multiplying each element of the first row of A by the corresponding element of the first column of B and adding these products together.

$$\begin{aligned} C(1,1) &= A(1,1)B(1,1) + A(1,2)B(2,1) + A(1,3)B(3,1) \\ &= (1)(1) + (2)(-2) + (3)(3) \\ &= 6 \end{aligned}$$

- Similarly,

$$\begin{aligned} C(1,2) &= A(1,1)B(1,2) + A(1,2)B(2,2) + A(1,3)B(3,2) \\ &= (1)(2) + (2)(0) + (3)(1) \\ &= 5 \end{aligned}$$

- Performing similar arithmetic we have

$$C = \begin{vmatrix} 6 & 5 \\ 12 & 14 \\ 18 & 23 \end{vmatrix}$$

- Multiplication is associative. This means that if we have several matrices to multiply together, it does not matter which we multiply first. For example, if A, B and C are the three matrices then

$$A(B,C) = (AB)C$$

This is very useful property. It allows us to combine several graphics transformations into a single transformation, thereby making our calculations more efficient.

Identity Matrix

- The square matrix which has all the elements equal to 0 except the elements of the main diagonal, which are all 1 is called identity matrix. For example,

$$\begin{vmatrix} 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \quad \text{and so on.}$$

- The identity matrix is denoted as matrix I and

$$A = A * I$$

Let us see the general procedures for applying translation, rotation and scaling parameters to reposition and resize the two dimensional objects.

Review Questions

- Explain how to perform matrix multiplication.
- What is identity matrix ?

6.3 Transformation Conventions

- Usually two conventions are used to represent data and to perform transformations with matrix multiplication : **Post-multiplication transformation** and **Pre-multiplication transformation**.
- In this text a right-hand co-ordinate system is used, the object is rotated in a fixed co-ordinate system, clockwise about an axis as seen by an observer at the origin looking outward along the positive axis and position vectors are represented as row matrices.
- The equation (6.3.1) represents a general transformation, where T represents any transformation such as rotation, scaling or translation. Since in this equation position vectors are represented as row matrices, the transformation matrix appears after the data or position vector matrix. This is a post-multiplication transformation

$$x' = [x] [T] \quad \dots (6.3.1)$$

- If we choose to represent the position vectors as a column matrix we have,
- $[x'] = [T]^{-1} [x]$
- For example, using homogeneous co-ordinates for positive rotation by an angle θ of an object about the origin (z-axis) using a post-multiplication transformation gives,

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

- However, using a pre-multiplication transformation we can perform same transformation using

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

Review Questions

1. Write a note on transformation conventions.
2. Explain post-multiplication and pre-multiplication transformations.

6.4 Two - dimensional Transformations SPPU : Dec.-07,09,14, May-12,14,15

6.4.1 Translation

- Translation is a process of changing the position of an object in a straight-line path from one co-ordinate location to another.
- We can translate a two dimensional point by adding translation distances, t_x and t_y , to the original co-ordinate position (x, y) to move the point to a new position (x', y') , as shown in the Fig. 6.4.1

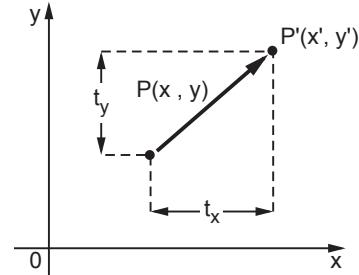


Fig. 6.4.1

$$x' = x + t_x \quad \dots(6.4.1)$$

$$y' = y + t_y \quad \dots (6.4.2)$$

- The translation distance pair (t_x, t_y) is called a **translation vector** or **shift vector**.
- It is possible to express the translation equations (6.4.1) and (6.4.2) as a single matrix equation by using column vectors to represent co-ordinate positions and the translation vector :

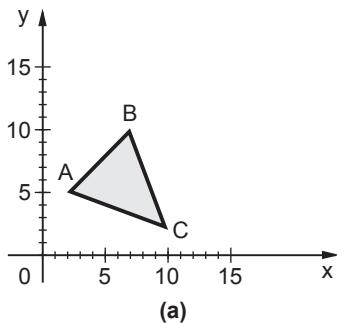
$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

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

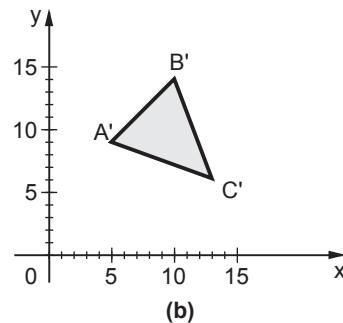
$$P' = P + T \quad \dots (6.4.3)$$

Example 6.4.1 Translate a polygon with co-ordinates A (2, 5), B (7, 10) and C (10,2) by 3 units in x direction and 4 units in y direction.

Solution : $A' = A + T = \begin{bmatrix} 2 \\ 5 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$



(a)



(b)

Fig. 6.4.2 Translation of polygon

$$B' = B + T = \begin{bmatrix} 7 \\ 10 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$C' = C + T = \begin{bmatrix} 10 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 13 \\ 6 \end{bmatrix}$$

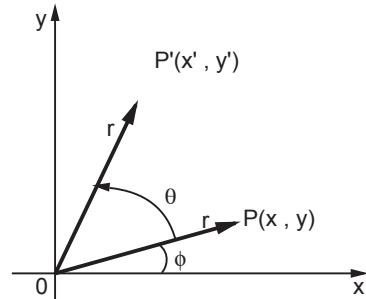
6.4.2 Rotation

- A two dimensional rotation is applied to an object by repositioning it along a circular path in the xy plane.
- To generate a rotation, we specify a rotation angle θ and the position of the rotation point about which the object is to be rotated.
- Let us consider the rotation of the object about the origin, as shown in the Fig. 6.4.3.
- Here, r is the constant distance of the point from the origin, angle ϕ is the original angular position of the point from the horizontal, and θ is the rotation angle.
- Using standard trigonometric equations, we can express the transformed co-ordinates in terms of angles θ and ϕ as

$$\left. \begin{aligned} x' &= r \cos(\phi + \theta) = r \cos\phi \cos\theta - r \sin\phi \sin\theta \\ y' &= r \sin(\phi + \theta) = r \cos\phi \sin\theta + r \sin\phi \cos\theta \end{aligned} \right\} \quad \dots(6.4.4)$$

- The original co-ordinates of the point in polar co-ordinates are given as

$$\left. \begin{aligned} x &= r \cos\phi \\ y &= r \sin\phi \end{aligned} \right\} \quad \dots(6.4.5)$$

**Fig. 6.4.3**

- Substituting equations (6.4.5) into (6.4.4), we get the transformation equations for rotating a point (x, y) through an angle θ about the origin as :

$$\left. \begin{array}{l} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{array} \right\} \quad \dots (6.4.6)$$

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

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

$$\therefore P' = P \cdot R \quad \dots (6.4.7)$$

where R is rotation matrix and it is given as

$$R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \quad \dots (6.4.8)$$

- It is important to note that positive values for the rotation angle define counterclockwise rotations about the rotation point and negative values rotate objects in the clockwise sense.
- For negative values of θ i.e. for clockwise rotation, the rotation matrix becomes

$$\begin{aligned} R &= \begin{bmatrix} \cos(-\theta) & \sin(-\theta) \\ -\sin(-\theta) & \cos(-\theta) \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad \because \cos(-\theta) = \cos \theta \quad \text{and} \\ &\qquad \qquad \qquad \sin(-\theta) = -\sin \theta \end{aligned} \quad \dots (6.4.9)$$

Example 6.4.2 A point $(4, 3)$ is rotated counterclockwise by an angle of 45° . Find the rotation matrix and the resultant point.

Solution :

$$R = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ \\ -\sin 45^\circ & \cos 45^\circ \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$\therefore P^1 = [4 \ 3] \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 4/\sqrt{2} - 3/\sqrt{2} & 4/\sqrt{2} + 3/\sqrt{2} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 7/\sqrt{2} \end{bmatrix}$$

6.4.3 Scaling

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

$$x' = x \cdot S_x$$

$$\text{and } y' = y \cdot S_y \quad \dots (6.4.10)$$

- Scaling factor S_x scales object in the x direction and scaling factor S_y scales object in the y direction. The equation (6.4.10) can be written in the matrix form as given below :

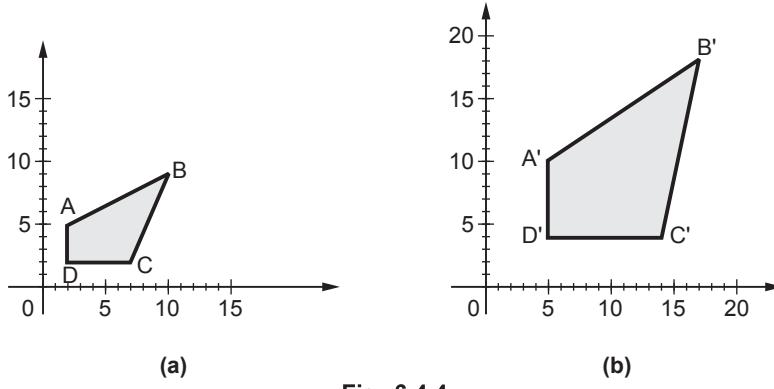


Fig. 6.4.4

$$\begin{aligned} [x' \ y'] &= [x \ y] \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \\ &= [x \cdot S_x \quad y \cdot S_y] \\ &= P \cdot S \end{aligned} \quad \dots \ (6.4.11)$$

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

Example 6.4.3 Scale the polygon with co-ordinates $A (2, 5)$, $B (7, 10)$ and $C (10, 2)$ by two units in x direction and two units in y direction.

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

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

The object matrix is : A $\begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix}$

$$\therefore \begin{matrix} A' \\ B' \\ C' \end{matrix} \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ x'_3 & y'_3 \end{bmatrix} = \begin{bmatrix} 2 & 5 \\ 7 & 10 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 4 & 10 \\ 14 & 20 \\ 20 & 4 \end{bmatrix}$$

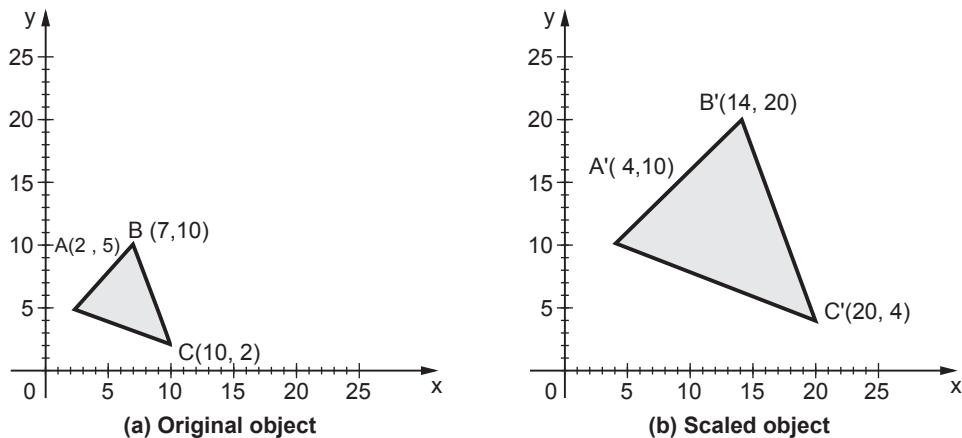


Fig. 6.4.5

Review Questions

1. Write 2D transformation matrices of translation, scaling. Give the derivation of 2D rotation matrix. **SPPU : May-14, Marks 4**
 2. Describe w.r.t. 2D transformation :
i) Scaling ii) Rotation iii) Translation **SPPU : Dec.-07, 09, May-12, Marks 8**
 3. Write transformation matrix for scaling and rotation **SPPU : Dec.-14, Marks 3**
 4. Write the transformation matrix for translation and scaling. **SPPU : May-15, Marks 2**

6.5 Homogeneous Co-ordinates

SPPU : Dec.-05,10,12,18 , May-07,12,16

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

$$P' = P \cdot M_1 + M_2 \quad \dots \quad (6.5.1)$$

- **For translation :** $P' = P \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$

i.e. M_1 = Identity matrix

M_7 = Translation vector

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

i.e. M_1 = Rotational matrix, $M_2 = 0$

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

i.e. M_1 = Scaling matrix

$$M_2 = 0$$

- To produce a sequence of transformations with above equations, such as translation followed by rotation and then scaling, we must calculate the transformed co-ordinates one step at a time. First, co-ordinates are translated, then these translated co-ordinates are scaled, and finally, the scaled co-ordinates are rotated. But this sequential transformation process is not efficient. A more efficient approach is to combine sequence of transformations into one transformation so that the final co-ordinate positions are obtained directly from initial co-ordinates. This eliminates the calculation of intermediate co-ordinate values.
- In order to combine sequence of transformations we have to eliminate the matrix addition associated with the translation terms in M_2 . To achieve this we have to represent matrix M_1 as 3×3 matrix instead of 2×2 introducing an additional dummy co-ordinate W. Here, points are specified by three numbers instead of two. This co-ordinate system is called **homogeneous co-ordinate system** and it allows us to express all transformation equations as matrix multiplication.
- The homogeneous co-ordinate is represented by a triplet (X_W, Y_W, W) , where

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

- For two dimensional transformations, we can have the homogeneous parameter W to be any non zero value. But it is convenient to have $W = 1$. Therefore, each two dimensional position can be represented with homogeneous co-ordinate as $(x, y, 1)$.
- Summarizing it all up, we can say that the homogeneous co-ordinates allow combined transformation, eliminating the calculation of intermediate co-ordinate values and thus save required time for transformation and memory required to store the intermediate co-ordinate values.

Let us see the homogeneous co-ordinates for three basic transformations.

6.5.1 Homogeneous Co-ordinates for Translation

- The homogeneous co-ordinates for translation are given as

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \quad \dots (6.5.2)$$

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ &= [x + t_x \ y + t_y \ 1] \end{aligned} \quad \dots (6.5.3)$$

6.5.2 Homogeneous Co-ordinates for Rotation

- The homogeneous co-ordinates for rotation are given as

$$R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots (6.5.4)$$

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x \cos\theta - y \sin\theta \quad x \sin\theta + y \cos\theta \quad 1] \end{aligned} \quad \dots (6.5.5)$$

6.5.3 Homogeneous Co-ordinates for Scaling

- The homogeneous co-ordinate for scaling are given as

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

Therefore, we have

$$\begin{aligned} [x' \ y' \ 1] &= [x \ y \ 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x \cdot S_x \quad y \cdot S_y \quad 1] \end{aligned} \quad \dots (6.5.6)$$

Note In this text, the object matrix is written first and it is then multiplied by the required transformation matrix.

If we wish to write the transformation matrix first and then the object matrix we have to take the transpose of both the matrices and post-multiply the object matrix ie.

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

Example 6.5.1 Give a 3×3 homogeneous co-ordinate transformation matrix for each of the following translations

- Shift the image to the right 3-units
- Shift the image up 2 units
- Move the image down $\frac{1}{2}$ unit and right 1 unit
- Move the image down $\frac{2}{3}$ unit and left 4 units

Solution : We know that homogenous co-ordinates for translation are

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

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

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

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

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

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

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

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

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

Example 6.5.2 Find the transformation matrix that transforms the given square ABCD to half its size with centre still remaining at the same position. The co-ordinates of the square are : A(1, 1), B (3, 1), C (3, 3), D (1, 3) and centre at (2, 2). Also find the resultant co-ordinates of square.

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

- Translate the square so that its center coincides with the origin.

2. Scale the square with respect to the origin.

3. Translate the square back to the original position.

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

$$\begin{aligned} \therefore T_1 \cdot S \cdot T &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 3 & 1 \\ 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1.5 & 1.5 & 1 \\ 2.5 & 1.5 & 1 \\ 2.5 & 2.5 & 1 \\ 1.5 & 2.5 & 1 \end{bmatrix} \end{aligned}$$

Example 6.5.3 Find a transformation of triangle A (1, 0), B (0, 1), C (1, 1) by

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

Solution : The rotation matrix is

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

The translation matrix is

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

a)

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

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

$$\begin{aligned}
 \text{b) } T \cdot R &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \end{bmatrix} \\
 \therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & \sqrt{2} & 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 3/\sqrt{2} & 1 \\ -1/\sqrt{2} & 3/\sqrt{2} & 1 \\ 0 & 2\sqrt{2} & 1 \end{bmatrix}
 \end{aligned}$$

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

Review Questions

- What is homogeneous co-ordinate system ? Explain need of homogeneous co-ordinates. Compare homogeneous and normalized co-ordinates. **SPPU : Dec.-05,18, May-07, 12, 16, Marks 8**
- What is the need of homogenous coordinates ? Give the homogenous coordinates for translation, rotation and scaling. **SPPU : Dec.-10, 12, Marks 10**

6.6 Composite Transformations

SPPU : Dec.-05,06,07,09,10,11,15,16,17,19, May-06,07,08,11,13,15,16,17,18,19

- The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.

6.6.1 Rotation about an Arbitrary Point

- To rotate an object about an arbitrary point, (x_p, y_p) we have to carry out three steps :
 - Translate point (x_p, y_p) to the origin
 - Rotate it about the origin and
 - Finally, translate the center of rotation back where it belongs (See Fig. 6.6.1 on next page)
- Matrix multiplication is not commutative, i.e. multiplying matrix A by matrix B will not always yield the same result as multiplying matrix B by matrix A. Therefore, in obtaining composite transformation matrix, we must be careful to

order the matrices so that they correspond to the order of the transformations on the object.

Let us find the transformation matrices to carry out individual steps.

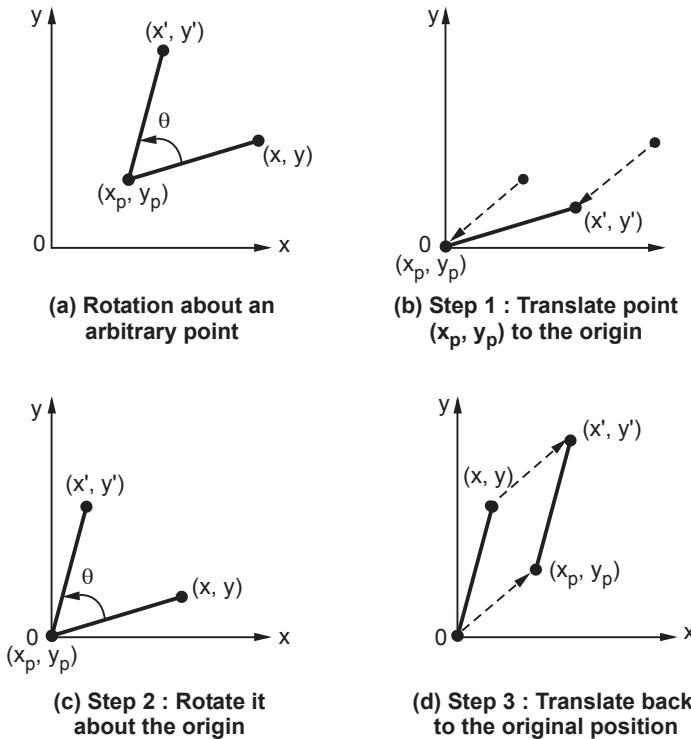


Fig. 6.6.1

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

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

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

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

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

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

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

$$\begin{aligned} T_1 \cdot R \cdot T_2 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_p & -y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta & -x_p \sin\theta - y_p \cos\theta & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_p & y_p & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_p \cos\theta + y_p \sin\theta + x_p & -x_p \sin\theta - y_p \cos\theta + y_p & 1 \end{bmatrix} \dots (6.6.1) \end{aligned}$$

Example 6.6.1 Perform a counterclockwise 45° rotation of triangle A (2, 3), B (5, 5), C (4, 3) about point (1, 1).

Solution : From equation 6.6.1 we have

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

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

$$\begin{aligned} T_1 \cdot R \cdot T_2 &= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix} \\ \therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} &= \begin{bmatrix} 2 & 3 & 1 \\ 5 & 5 & 1 \\ 4 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1 & -\sqrt{2}+1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{\sqrt{2}}+1 & \frac{3}{\sqrt{2}}+1 & 1 \\ 1 & \frac{8}{\sqrt{2}}+1 & 1 \\ \frac{1}{\sqrt{2}}+1 & \frac{5}{\sqrt{2}}+1 & 1 \end{bmatrix} \end{aligned}$$

Example 6.6.2 Perform a 45° rotation of triangle A (0, 0), B (1, 1), C (5, 2).

- i) About the origin
- ii) About P (-1, -1).

SPPU : Dec.-05, 09, Marks 10

Solution : Assume that rotation is clockwise.

i) About the origin :

The rotation matrix for clockwise rotation is given as

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

where $\theta = 45^\circ$

$$\therefore R = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ \sqrt{2} & 0 & 1 \\ 7/\sqrt{2} & -3/\sqrt{2} & 1 \end{bmatrix}$$

\therefore After rotation A = (0, 0), B = $(\sqrt{2}, 0)$ and C = $(7/\sqrt{2}, -3/\sqrt{2})$

ii) About P (-1, -1) :

The rotation matrix for clockwise rotation about an arbitrary point is given as

$$T_1 R T_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -X_p & -Y_p & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ X_p & Y_p & 1 \end{bmatrix}$$

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

Substituting value of $\theta = 45^\circ$ we have

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

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

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

$$= \begin{bmatrix} \sqrt{2} - 1 & -1 & 1 \\ \sqrt{2} + \sqrt{2} - 1 & -1 & 1 \\ \frac{9}{\sqrt{2}} - 1 & \frac{-3}{\sqrt{2}} - 1 & 1 \end{bmatrix}$$

\therefore After rotation $A = (\sqrt{2} - 1, -1)$, $B = (\sqrt{2} + \sqrt{2} - 1, -1)$ and $C = \left(\frac{9}{\sqrt{2}} - 1, \frac{-3}{\sqrt{2}} - 1\right)$

Example 6.6.3 Consider the square A(1, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square by 45° anticlockwise direction followed by reflection about X-axis. **SPPU : May-06, Marks 8**

Solution :

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

$$\theta = 45^\circ$$

$$\begin{aligned} \therefore T &= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \therefore \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 1 \\ 0 & 0 & 1 \\ -1/\sqrt{2} & -1/\sqrt{2} & 1 \\ 0 & -\sqrt{2} & 1 \end{bmatrix} \end{aligned}$$

$$\therefore A'(1/\sqrt{2}, -1/\sqrt{2}), B'(0, 0), C'(-1/\sqrt{2}, -1/\sqrt{2}) \text{ and } D'(0, -\sqrt{2})$$

Example 6.6.4 Consider the square P(0, 0), Q(0, 10), R(10, 10), S(10, 0). Rotate the square about fixed point R(10, 10) by an angle 45° (anticlockwise) followed by scaling by 2 units in X direction and 2 units in Y direction. **SPPU : May-06, Marks 8**

Solution :

$$T = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -X_p \cos\theta + Y_p \sin\theta + X_p & -X_p \sin\theta - Y_p \cos\theta + Y_p & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\theta = 45^\circ$$

$$\therefore T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 10 & -10(\sqrt{2})+10 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \sqrt{2} & \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{2} & 0 \\ 20 & -20\sqrt{2} + 20 & 1 \end{bmatrix}$$

$$\begin{bmatrix} P' \\ Q' \\ R' \\ S' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 10 & 1 \\ 10 & 10 & 1 \\ 10 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{2} & \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{2} & 0 \\ 20 & -20\sqrt{2} + 20 & 1 \end{bmatrix}$$

$$\begin{bmatrix} P' \\ Q' \\ R' \\ S' \end{bmatrix} = \begin{bmatrix} 20 & -20\sqrt{2} + 20 & 1 \\ -10\sqrt{2} + 20 & -10\sqrt{2} + 20 & 1 \\ 20 & 20 & 1 \\ 10\sqrt{2} + 20 & -10\sqrt{2} + 20 & 1 \end{bmatrix}$$

Example 6.6.5 Find out co-ordinates of figure bounded by (0, 0), (1, 5), (6, 3), (-3, -4) column reflected along line whose equation is $y = 2x + 4$ and sheared by 2 units in x and 2 units in y-direction.

SPPU : Dec.-06, Marks 8

Solution : (Refer example 6.7.6) From the solution of this example we have transformation matrix for a point to be reflected about a line $y = mx + c$ as,

$$R_T = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & 0 \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & 0 \\ \frac{-2cm}{m^2+1} & \frac{2c}{m^2+1} & 1 \end{bmatrix}$$

Now substituting $m = 2$ and $c = 4$ we have,

$$R_T = \begin{bmatrix} \frac{1-4}{4+1} & \frac{4}{5} & 0 \\ \frac{4}{5} & \frac{3}{5} & 0 \\ \frac{-16}{5} & \frac{8}{5} & 1 \end{bmatrix} = \begin{bmatrix} -0.6 & 0.8 & 0 \\ 0.8 & 0.6 & 0 \\ -3.2 & 1.6 & 1 \end{bmatrix}$$

Now combine this matrix with shear matrix to get the resultant matrix. By multiplying each point with resultant matrix we can get the coordinates of figure after desired transformation.

Example 6.6.6 Give the 3×3 homogeneous transformation matrix for each of the following transformation sequences i) Rotate counterclockwise about the origin by 45° and then scale the x -direction by one half as large.
ii) Scale the y -direction by twice as tall, shift down by 1 unit and then rotate clockwise by 30° .

Solution : The homogeneous co-ordinate transformation matrix for counter clockwise rotation of a point or object about the origin is given as,

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

The homogeneous co-ordinate transformation matrix for scaling a point or object is given as

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

In the given problem $\theta = 45^\circ$, $S_x = 1.5$ and $S_y = 1$.

Therefore, overall transformation matrix can be given as,

$$\begin{aligned} [x' y' 1] &= [x, y, 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x, y, 1] \begin{bmatrix} S_x \cdot \cos\theta & S_y \cdot \sin\theta & 0 \\ -S_x \cdot \sin\theta & S_y \cdot \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x, y, 1] \begin{bmatrix} 1.5 \cos 45 & 1 \sin 45 & 0 \\ -1.5 \sin 45 & 1 \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= [x, y, 1] \begin{bmatrix} 1.06 & 0.707 & 0 \\ -1.06 & 0.707 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

ii) The transformation sequence for part (ii) is scaling, translation and rotation. Here, $S_x = 1$, $S_y = 2$, $t_x = 0$, $t_y = -1$ and $\theta = -30^\circ$. We know that, homogeneous co-ordinate transformation matrix for translation is given as,

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

Therefore, overall transformation matrix can be given as,

$$\begin{aligned} [x', y', 1] &= [x, y, 1] \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ &= [x, y, 1] \begin{bmatrix} S_x \cdot \cos \theta & S_x \cdot \sin \theta & 0 \\ -S_y \cdot \sin \theta & S_y \cdot \cos \theta & 0 \\ t_x & t_y & 1 \end{bmatrix} \end{aligned}$$

Substituting value of 0, S_x , S_y , t_x and t_y we get,

$$\begin{aligned} [x', y', 1] &= [x, y, 1] \begin{bmatrix} 1 \cdot \cos(-30) & 1 \cdot \sin(-30) & 0 \\ -2 \cdot \sin(-30) & 2 \cdot \cos(-30) & 0 \\ 0 & -1 & 1 \end{bmatrix} \\ &= [x, y, 1] \begin{bmatrix} 0.866 & -0.5 & 0 \\ 1 & 1.732 & 0 \\ 0 & -1 & 1 \end{bmatrix} \end{aligned}$$

Example 6.6.7 Write a 2×2 transformation matrix for each of the following rotations about the origin.

- a) Counter clockwise by π ; b) Counter clockwise by $\pi/2$;
- c) Clockwise by $\pi/2$; d) Counter clockwise by $5\pi/4$.

Solution : The transformation matrix for counter clockwise rotation of θ about the origin is given as

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

For clockwise rotation θ is negative.

- i) Transformation matrix for rotation of counter clockwise by π (180°)

$$\begin{aligned} R &= \begin{bmatrix} \cos 180 & \sin 180 \\ -\sin 180 & \cos 180 \end{bmatrix} \\ &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \end{aligned}$$

ii) Transformation matrix for rotation of counter clockwise by $\pi/2(90^\circ)$

$$R = \begin{bmatrix} \cos 90 & \sin 90 \\ -\sin 90 & \cos 90 \end{bmatrix}$$

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

iii) Transformation matrix for rotation of clockwise by $\pi/2(90^\circ)$

$$R = \begin{bmatrix} \cos(-90) & \sin(-90) \\ -\sin(-90) & \cos(-90) \end{bmatrix}$$

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

iv) Transformation matrix for rotation of counter clockwise by $5\pi/4(225^\circ)$

$$R = \begin{bmatrix} \cos(225) & \sin(225) \\ -\sin(225) & \cos(225) \end{bmatrix}$$

$$= \begin{bmatrix} -0.707 & -0.707 \\ 0.707 & -0.707 \end{bmatrix}$$

Example 6.6.8 Write the general form of the scaling matrix with respect to a fixed point $P(h, k)$.

Solution : To determine the general form of the scaling matrix with respect to a fixed point $P(h, k)$ we have to perform three steps :

1. Translate point $P(h, k)$ at the origin by performing translation (T_1).
2. Scale the point or object by performing scaling (S).
3. Translate the origin back by performing reverse translation (T_2).

Therefore, the general form of the scaling matrix is a composition of T_1ST_2 matrices. It can be given as,

$$\begin{aligned} S_p &= T_1ST_2 \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ -h & -k & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h & k & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -h & -k & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ h & k & 1 \end{bmatrix} \\ &= \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ -S_xh + h & S_yk + k & 1 \end{bmatrix} \end{aligned}$$

Example 6.6.9 Prove that two scaling transformations commute, that is $S_1S_2 = S_2S_1$.

SPPU : May-07, Dec.-07, Marks 8

Solution : We have

$$S_1S_2 = \begin{bmatrix} S_{x1} & 0 \\ 0 & S_{y1} \end{bmatrix} \begin{bmatrix} S_{x2} & 0 \\ 0 & S_{y2} \end{bmatrix} \text{ and}$$

$$S_2S_1 = \begin{bmatrix} S_{x2} & 0 \\ 0 & S_{y2} \end{bmatrix} \begin{bmatrix} S_{x1} & 0 \\ 0 & S_{y1} \end{bmatrix}$$

$$\therefore S_1S_2 = \begin{bmatrix} S_{x1}S_{x2} & 0 \\ 0 & S_{y1}S_{y2} \end{bmatrix} \text{ and}$$

$$S_2S_1 = \begin{bmatrix} S_{x2}S_{x1} & 0 \\ 0 & S_{y2}S_{y1} \end{bmatrix}$$

Since multiplication is commutative

$$S_{x1}S_{x2} = S_{x2}S_{x1} \text{ and } S_{y1}S_{y2} = S_{y2}S_{y1}. \text{ Therefore,}$$

$$S_1S_2 = S_2S_1$$

Example 6.6.10 Prove that two 2D rotations about the origin commute; that is $R_1R_2 = R_2R_1$

SPPU : Dec.-10, Marks 8

Solution : We have

$$R_1R_2 = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix} \text{ and}$$

$$R_2R_1 = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix}$$

$$\therefore R_1R_2 = \begin{bmatrix} \cos\theta_1 \cdot \cos\theta_2 - \sin\theta_1 \sin\theta_2 & \cos\theta_1 \cdot \sin\theta_2 - \sin\theta_1 \cos\theta_2 \\ -\sin\theta_1 \cdot \cos\theta_2 - \cos\theta_1 \sin\theta_2 & -\sin\theta_1 \cdot \sin\theta_2 + \cos\theta_1 \cos\theta_2 \end{bmatrix} \text{ and}$$

$$R_2R_1 = \begin{bmatrix} \cos\theta_2 \cdot \cos\theta_1 - \sin\theta_2 \sin\theta_1 & \cos\theta_2 \cdot \sin\theta_1 + \sin\theta_2 \cdot \cos\theta_1 \\ -\sin\theta_2 \cdot \cos\theta_1 - \cos\theta_2 \sin\theta_1 & -\sin\theta_2 \cdot \sin\theta_1 + \cos\theta_2 \cdot \cos\theta_1 \end{bmatrix}$$

Since, multiplication is commutative $\cos\theta_1 \cdot \cos\theta_2 = \cos\theta_2 \cdot \cos\theta_1$. Therefore,
 $R_1R_2 = R_2R_1$.

Example 6.6.11 Find out the final co-ordinates of a figure bounded by the co-ordinates (1, 1), (3, 4), (5, 7), (10, 3) when rotated about a point (8, 8) by 30° in clockwise direction and scaled by two units in x-direction and three units y direction.

Solution : From equation 6.6.1, we have the transformation matrix for rotation about an arbitrary point given as,

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

In this case, it is clockwise rotation therefore we take value of θ negative.

$$\begin{aligned} \therefore T_1 \cdot R \cdot T_2 &= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} \cos(-30) & \sin(-30) & 0 \\ -\sin(-30) & \cos(-30) & 0 \\ -8 \times \cos(-30) + 8 \times \sin(-30) + 8 & -8 \times \sin(-30) - 8 \times \cos(-30) + 8 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 1 & 1 \\ 3 & 4 & 1 \\ 5 & 7 & 1 \\ 10 & 3 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ -2.928 & 5.072 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix} \\ \therefore T_1 \cdot R \cdot T_2 \cdot S &= \begin{bmatrix} -1.562 & 5.438 & 1 \\ 1.67 & 7.036 & 1 \\ 4.902 & 8.634 & 1 \\ 7.232 & 2.67 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -3.124 & 16.314 & 1 \\ 3.34 & 21.108 & 1 \\ 9.804 & 25.902 & 1 \\ 14.464 & 8.01 & 1 \end{bmatrix} \end{aligned}$$

Example 6.6.12 Prove that successive 2D rotations are additive; i.e.,

$$R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$$

Solution : From equation (6.4.8) we can write rotation matrix $R(\theta_1)$ as,

$$\begin{aligned} R(\theta_1) &= \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \text{ and } R(\theta_2) = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix} \\ \therefore R(\theta_1) \cdot R(\theta_2) &= \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \times \begin{bmatrix} \cos\theta_2 & \sin\theta_2 \\ -\sin\theta_2 & \cos\theta_2 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_1 \cdot \cos\theta_2 + \sin\theta_1 \cdot (-\sin\theta_2) & \cos\theta_1 \cdot \sin\theta_2 + \sin\theta_1 \cdot \cos\theta_2 \\ -\sin\theta_1 \cdot \cos\theta_2 + \cos\theta_1 \cdot (-\sin\theta_2) & -\sin\theta_1 \cdot \sin\theta_2 + \cos\theta_1 \cdot \cos\theta_2 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) \\ -\sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) \end{bmatrix} \end{aligned}$$

Since,

$$\cos(\theta_1 + \theta_2) = \cos\theta_1 \cos\theta_2 - \sin\theta_1 \sin\theta_2$$

$$\sin(\theta_1 + \theta_2) = \cos\theta_1 \sin\theta_2 + \sin\theta_1 \cos\theta_2$$

Example 6.6.13 Prove that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not.

SPPU : May-08, Marks 8

Solution : The matrix notation for scaling along S_x and S_y is as given below.

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

The matrix notation for rotation is as given below

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

$$\begin{aligned} S \cdot R &= \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} = \begin{bmatrix} S_x \cos\theta & S_x \sin\theta \\ -S_y \sin\theta & S_y \cos\theta \end{bmatrix} \\ &= \begin{bmatrix} S_x \cos\theta & S_x \sin\theta \\ -S_x \sin\theta & S_x \cos\theta \end{bmatrix} \quad (\because S_x = S_y) \dots (I) \end{aligned}$$

$$\text{or} \quad = \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} \quad (\because \theta = n\pi \text{ where } n \text{ is integer}) \dots (II)$$

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

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

$$= \begin{bmatrix} S_x \cos \theta & S_x \sin \theta \\ -S_x \sin \theta & S_x \cos \theta \end{bmatrix} \quad \because S_x = S_y \dots \text{(III)}$$

or

$$= \begin{bmatrix} -S_x & 0 \\ 0 & -S_y \end{bmatrix} \quad \because \theta = n\pi \text{ where } n \text{ is integer} \dots \text{(IV)}$$

From equations (I) and (III) and equations (II) and (IV) it is proved that 2D rotation and scaling commute if $S_x = S_y$ or $\theta = n\pi$ for integral n and that otherwise they do not.

Example 6.6.14 A 2D rectangular block with 1 unit height and 2 units width has one vertex "A" at origin. The block is shifted by 1 unit in X-direction and scaled by 2 units along Y-direction. Draw initial state of the rectangle and transformed final state of given rectangle. Give complete mathematical formulation. SPPU : May-11, Marks 18

Solution :

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

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

Co-ordinates of rectangle are :

A(0,0), B(0,1), C(2,1), D(2,0)

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 0 & 1 \end{bmatrix}$$

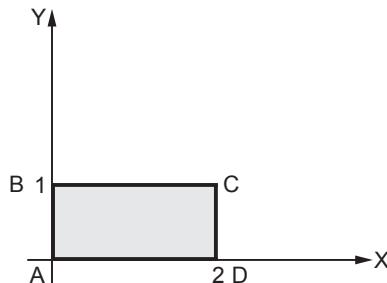


Fig. 6.6.2 Initial state

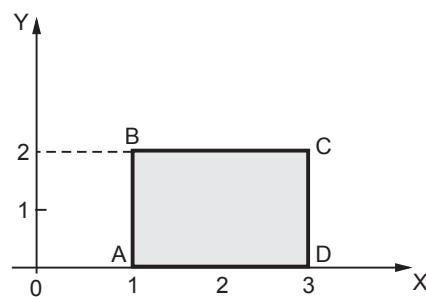


Fig. 6.6.3 Final state

Example 6.6.15 Fig. 6.6.4 and Fig. 6.6.5 show basic 2D blocks. Apply translation and scaling transformations to get the Fig. 6.6.6. Draw diagrams of all intermediate steps.

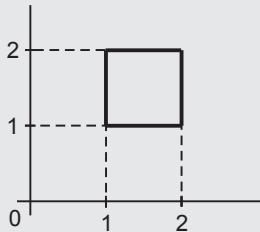


Fig. 6.6.4

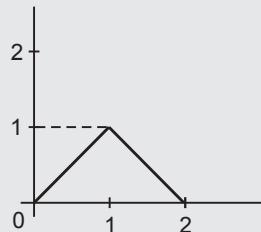


Fig. 6.6.5

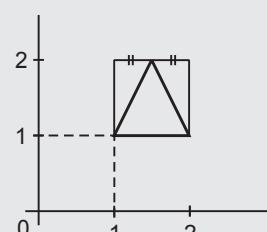


Fig. 6.6.6

Solution : The scaling matrix is

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

Here $S_x = 0.5$, $S_y = 1$

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

The translation matrix is

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

Here, $t_x = 1$, $t_y = 1$

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

\therefore The overall transformation matrix is

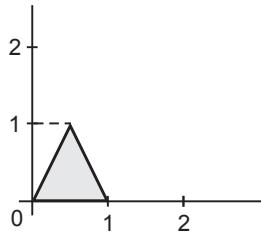
$$\text{S.T.} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

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

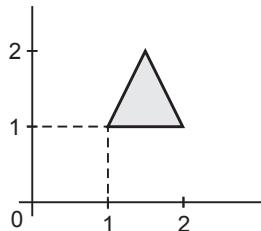
By applying this transformation to a triangle $(0, 0), (1, 1), (2, 0)$, we get

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

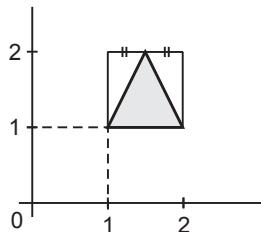
After applying scaling transformation to Fig. 6.6.5,



After applying translation transformation to above figure,



No transformation is applied to Fig. 6.6.4.



\therefore By applying overall transformation we get,

Example 6.6.16 Consider the square $A(1, 0), B(0, 0), C(0, 1)$ and $D(1, 1)$. Show the steps to rotate the given square by 45 degrees clockwise about point $A(1, 0)$.

SPPU : Dec.-10, Marks 10

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

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

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

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

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

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

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

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

Here, $\theta = 45^\circ$, $x_p = 1$ and $y_p = 0$.

Substituting values we get,

$$\begin{aligned} T_1 \cdot R \cdot T_2 &= \begin{bmatrix} \cos 45 & -\sin 45 & 0 \\ \sin 45 & \cos 45 & 0 \\ -\cos 45 + 1 & \sin 45 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} + 1 & \frac{1}{\sqrt{2}} & 1 \end{bmatrix} \end{aligned}$$

Example 6.6.17 Magnify the triangle with vertices $A(0, 0)$, $B(1, 1)$, $C(5, 2)$ to twice its size as well as rotate it by 45° . Derive the translation matrices. **SPPU : Dec.-11, May-13, Marks 8**

Solution :

$$\text{Scaling matrix is given by } S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and}$$

$$\text{Rotation matrix is given by } R = \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore SR = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x \cos \theta & s_x \sin \theta & 0 \\ -s_y \sin \theta & s_y \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where $s_x = 2$, $s_y = 2$ and $\theta = 45^\circ$

$$\therefore SR = \begin{bmatrix} 2 \cos 45 & 2 \sin 45 & 0 \\ -2 \sin 45 & 2 \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \cos 45 & 2 \sin 45 & 0 \\ -2 \sin 45 & 2 \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 2.828 & 1 \\ 4.2426 & 9.898 & 1 \end{bmatrix}$$

Example 6.6.18 Perform scaling on a triangle $(1, 1)$, $(8, 1)$ and $(1, 9)$ with scaling factor of 2 in both x and y directions. Find the final coordinates of triangle.

SPPU : May-17, Marks 2

Solution :

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 8 & 1 & 1 \\ 1 & 9 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 1 \\ 16 & 2 & 1 \\ 2 & 18 & 1 \end{bmatrix}$$

The transformed points are $A' = (2, 2)$, $B' = (16, 2)$ and $C' = (2, 18)$

Example 6.6.19 For Origin centered unit square, rotate 45° clockwise, scale by a factor 2 in x -direction. Find resultant coordinates of square(write required matrices).

SPPU : Dec.-17, Marks 4

Solution :

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

$$\begin{aligned}
 &= \begin{bmatrix} S_x \cos \theta & -S_x \sin \theta & 0 \\ S_y \sin \theta & S_y \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
 \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \cos 45 & -2 \sin 45 & 0 \\ 2 \sin 45 & 2 \cos 45 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2\sqrt{2} & 0 & 1 \\ 0 & 2\sqrt{2} & 1 \\ -2\sqrt{2} & 0 & 1 \\ 0 & -2\sqrt{2} & 1 \end{bmatrix}
 \end{aligned}$$

Example 6.6.20 Rotate origin centered square with 2 unit length of each side, in clockwise direction with rotation angle of 90°.

SPPU : May-18, Marks 3

Solution :

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

Example 6.6.21 Consider a square P(0, 0), Q(0, 10), R(10, 10), S(10, 0). Rotate the square anticlockwise about fixed point R(10, 10) by an angle 45 degree. SPPU : May-19, Marks 4

Solution :

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

Note : Here, rotation is anticlockwise

Given : $\theta = 45^\circ$, $x_p = 10$ and $y_p = 10$

$$T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 10 & -10(\sqrt{2})+10 & 1 \end{bmatrix}$$

$$\begin{bmatrix} P' \\ Q' \\ R' \\ S' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 10 & 1 \\ 10 & 10 & 1 \\ 10 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 10 & -\frac{20}{\sqrt{2}}+10 & 1 \end{bmatrix} = \begin{bmatrix} 10 & \frac{-20}{\sqrt{2}}+10 & 1 \\ \frac{-10}{\sqrt{2}}+10 & \frac{-10}{\sqrt{2}}+10 & 1 \\ 10 & \frac{10}{\sqrt{2}} & 1 \\ \frac{10}{\sqrt{2}}+10 & \frac{-10}{\sqrt{2}}+10 & 1 \end{bmatrix}$$

Example 6.6.22 Find a transformation of a triangle A(1, 0) B(0, 1) C(1, 1) by translating one unit in x and y directions and then rotating 45° about the origin.

SPPU : Dec.-19, Marks 6

Solution : The rotation matrix is

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

The translation matrix is

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

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

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

Review Questions

- What is composite transformation ?
- Explain the concept of 2D rotation about an arbitrary point with matrix representation.

SPPU : May-15,Dec.-17, Marks 6

- Explain rotation about arbitrary point. Generate transformation matrix for same.

SPPU : Dec.-15, Marks 6

- Derive transformation matrix for rotation about arbitrary point.

SPPU : May-16, Marks 4

- Derive matrix for rotation about arbitrary point. Also rotate point (3, 3) with respect to (1, 1) by 90 degree.

SPPU : Dec.-16, Marks 4

6.7 Reflection and Shear Transformations

SPPU : May-07,08,09,12,14,16,19, Dec.-08,12,14

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

6.7.1 Reflection

- A reflection is a transformation that produces a mirror image of an object relative to an axis of reflection.
- We can choose an axis of reflection in the xy plane or perpendicular to the xy plane.
- Table 6.7.1 gives examples of some common reflections.

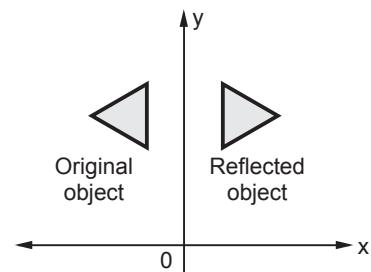


Fig. 6.7.1 Reflection about y axis

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

Reflection about line $y = -x$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$		
-----------------------------------	-----------------------------------------------------------------------	--	--

Table 6.7.1 Common reflections**Reflections about Arbitrary Line**

- Reflections about any line $y = mx + b$ in the xy plane can be accomplished with a combination of translate-rotate-reflect transformations. In general, we first translate the line so that it passes through the origin. Then we can rotate the line onto one of the coordinate axes and reflect about that axis. Finally, we restore the line to its original position with the inverse rotation and translation transformations.

6.7.2 Shear

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

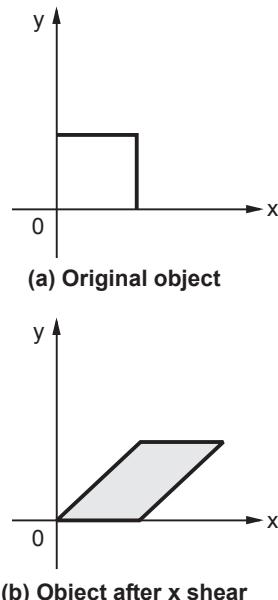
6.7.2.1 X Shear

- The x shear preserves the y co-ordinates, but changes the x values which causes vertical lines to tilt right or left as shown in the Fig. 6.7.2.
- The transformation matrix for x shear is given as

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

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

$$y' = y \quad \dots (6.7.1)$$

**Fig. 6.7.2**

6.7.2.2 Y Shear

- The y shear preserves the x co-ordinates, but changes the y values which causes horizontal lines to transform into lines which slope up or down, as shown in the Fig. 6.7.3.
- The transformation matrix for y shear is given as

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

$\therefore x' = x$ and

$$y' = y + Sh_y \cdot x \quad \dots (6.7.2)$$

6.7.2.3 Shearing Relative to Other Reference Line

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

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

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

Example 6.7.1 Apply the shearing transformation to square with A(0, 0), B (1, 0), C (1, 1) and D (0, 1) as given below

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

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

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

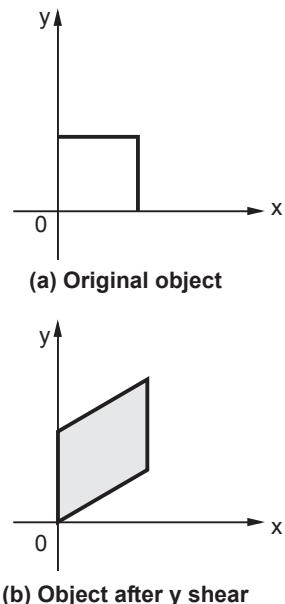


Fig. 6.7.3

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

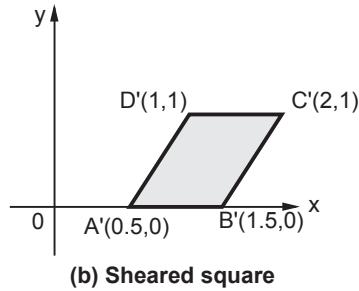
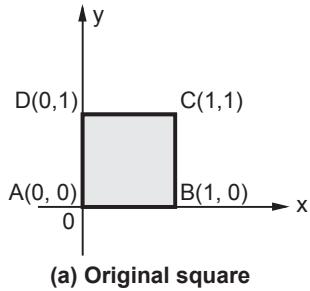


Fig. 6.7.4

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

$$\begin{aligned} \begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} &= \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & -Sh_y \cdot x_{ref} & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0.5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0.5 & 1 \\ 1 & 1 & 1 \\ 1 & 2 & 1 \\ 0 & 1.5 & 1 \end{bmatrix} \end{aligned}$$

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

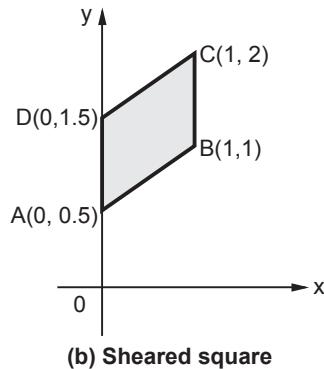
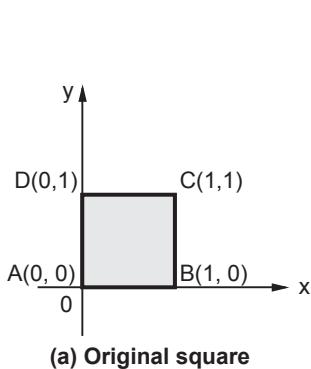


Fig. 6.7.5

Example 6.7.2 Show how shear transformation may be expressed in terms of rotation and scaling.

SPPU : May-09, Marks 3

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

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

We have scaling matrix and rotation matrix as given below

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

If we combine scale matrix and rotation matrix we have,

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

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

$$Sh_x = -S_y \sin\theta$$

$$Sh_y = S_x \sin\theta$$

$$S_x \cos\theta = 1 \quad \text{and}$$

$$S_y \cos\theta = 1$$

$$\therefore S_x = \frac{1}{\cos\theta} \quad \text{and}$$

$$S_y = \frac{1}{\cos\theta}$$

Substituting values of S_x and S_y we get,

$$Sh_x = -\frac{1}{\cos\theta} \cdot \sin\theta = -\tan\theta$$

$$Sh_y = \frac{1}{\cos\theta} \cdot \sin\theta = \tan\theta$$

Therefore, the shear transformation matrix expressed in terms of rotation and scales is

$$\begin{bmatrix} 1 & \tan\theta & 0 \\ -\tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \dots \because S_x \cos\theta = S_y \cos\theta = 1$$

where θ : Angle of rotation

S_x : x scale and

S_y : y scale

Example 6.7.3 Show how reflection in the line $y = x$ and in the line $y = -x$ can be performed by a scaling operation followed by a rotation.

Solution : The transformation matrix for scaling and rotation are given as,

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

Therefore, the overall matrix is

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

i) Equating the transformation matrix for reflection in the line $y = x$ and SR transformation matrix we get,

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

$$S_x \cdot \cos\theta = S_y \cdot \cos\theta = 0 \quad \therefore \theta = 90^\circ$$

$$S_x \cdot \sin 90 = 1 \quad \therefore S_x = 1 \text{ and}$$

$$-S_y \cdot \sin 90 = 1 \quad \therefore S_y = -1$$

Therefore, with $S_x = 1$, $S_y = -1$ and $\theta = 90^\circ$ we can get the reflection in the line $y = x$ by a scaling operation followed by a rotation.

ii) Equating the transformation matrix for reflection in the line $y = -x$ and SR transformation matrix we get,

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

$$S_x \cdot \cos\theta = S_y \cdot \cos\theta = 0 \quad \therefore \theta = 90^\circ$$

$$S_x \cdot \sin 90 = -1 \quad \therefore S_x = -1$$

$$-S_y \cdot \sin 90 = -1 \quad \therefore S_y = 1$$

Therefore, with $S_x = -1$, $S_y = 1$ and $\theta = 90^\circ$ we can get the reflection in the line $y = -x$ by a scaling operation followed by a rotation.

Example 6.7.4 Show that a rotation about the origin can be done by performing three shearing transformations.

Solution : The shearing transformation matrix in both x and y direction is given as,

$$Sh = \begin{bmatrix} 1 & b \\ a & 1 \end{bmatrix} \text{ where } a \text{ represents } x \text{ shear and } b \text{ represents } y \text{ shear}$$

The overall transformation matrix for three shearing transformations can be given as,

$$\begin{aligned} Sh_3 &= \begin{bmatrix} 1 & b_1 \\ a_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & b_2 \\ a_2 & 1 \end{bmatrix} \begin{bmatrix} 1 & b_3 \\ a_3 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1+a_2b_1 & b_1+b_2 \\ a_1+a_2 & a_1b_2+1 \end{bmatrix} \begin{bmatrix} 1 & b_3 \\ a_3 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1+a_2b_1+a_3b_1+a_3b_2 & b_3+a_2b_1b_3+b_1+b_2 \\ a_1+a_2+a_1a_3b_2+a_3 & a_1b_3+a_2b_3+a_1b_2+1 \end{bmatrix} \end{aligned}$$

If this matrix is to represent rotation then element on major diagonal should be equal and element on minor diagonal should be equal with opposite sign.

Therefore, we have

$$1+a_2b_1+a_3b_1+a_3b_2 = a_1b_3+a_2b_3+a_1b_2+1 \quad \dots (1)$$

$$b_3+a_2b_1b_3+b_1+b_2 = -a_1-a_2-a_1a_3b_2-a_3 \quad \dots (2)$$

To simplify this we assume $a_1 = a_3$ substituting these values we get,

$$1+a_2b_1+a_1b_1+a_1b_2 = a_1b_3+a_2b_3+a_1b_2+1$$

$$\therefore a_2b_1+a_1b_1 = a_1b_3+a_2b_3$$

$$\therefore b_1(a_1+a_2) = b_3(a_1+a_2)$$

$$\therefore b_1 = b_3$$

Substituting $a_1 = a_3$, $b_1 = b_3$ and $b_2 = -a_2$ in equation (2) we get,

$$b_1-b_2b_1b_1+b_1+b_2 = -a_1+b_2-a_1a_1b_2-a_1$$

$$\therefore 2b_1+2a_1-b_2b_1^2+a_1^2b_2 = 0$$

$$\therefore 2(a_1+b_1)+b_2(a_1^2-b_1^2) = 0$$

$$\therefore 2(a_1+b_1)+b_2[(a_1+b_1)(a_1-b_1)] = 0$$

Assuming $a_1 - b_1$ i.e., $a_1 = 1 + b_1$ we have,

$$2(1+b_1+b_1)+b_2[(1+b_1+b_1)(1)] = 0$$

$$\therefore 2(1+2b_1)+b_2(1+2b_1) = 0$$

$$\begin{aligned}\therefore b_2(1+2b_1) &= -2(1+2b_1) \\ \therefore b_2 &= -2 \\ \therefore a_2 &= -b_2 \\ &= 2\end{aligned}$$

Substituting $a_1 = 1 + b_1$, $a_2 = -b_2 = 2$, $a_3 = a_1 = 1 + b_1$, and $b_3 = b_1$ we get the overall shear matrix as,

$$Sh_3 = \begin{bmatrix} 1 & b_1 \\ 1+b_1 & 1 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & b_1 \\ 1+b_1 & 1 \end{bmatrix}$$

We can observe that parameter b_1 is varied as the rotation angle is varied. The above overall shear transformation matrix performs the rotation about the origin.

Example 6.7.5 Show that transformation matrix for a reflection about a line $Y = X$ is equivalent to reflection to X -axis followed by counter clockwise rotation of 90° .

SPPU : Dec.-08, Marks 8

Solution : The transformation matrix for reflection about a line $Y = X$ is given as,

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

The transformation matrix for reflection about x -axis and for counter clockwise rotation of 90° are given as,

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \text{ and } \begin{bmatrix} \cos 90^\circ & \sin 90^\circ \\ -\sin 90^\circ & \cos 90^\circ \end{bmatrix}$$

Hence,

$$\begin{aligned}T &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \dots \text{ Proved}\end{aligned}$$

Example 6.7.6 Find out final transformation matrix, when point $P(x, y)$ is to be reflected about a line $y = mx + c$.

Solution : Equation of line :

$$y = mx + c$$

$$\text{Slope} = m \quad y \text{ intercept} = c$$

We can relate slope m to angle θ by equation

$$m = \tan \theta$$

$$\therefore \theta = \tan^{-1} m$$

where θ is inclination of line with respect to x-axis.

Translation matrix can be given as,

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

Rotation matrix to match the given line with x-axis can be obtained as,

$$R_z = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad [\text{Note : Angle of rotation} = -\theta]$$

Reflection matrix about x-axis

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

Inverse transformation matrices,

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

\therefore Final transformation matrix can be obtained as,

$$R_T = T \cdot R_z \cdot M \cdot R_z^{-1} \cdot T^{-1}$$

As we have $\tan \theta = m$, using trigonometric identities we can obtain,

$$\sin \theta = \frac{m}{\sqrt{m^2 + 1}} \quad \cos \theta = \frac{1}{\sqrt{m^2 + 1}}$$

$$\therefore R_T = \begin{bmatrix} \cos 2\theta & \sin 2\theta & 0 \\ \sin 2\theta & -\cos 2\theta & 0 \\ -c \sin 2\theta & c(1 + \cos 2\theta) & 1 \end{bmatrix}$$

By substituting values of $\sin \theta$ and $\cos \theta$ we have,

$$R_T = \begin{bmatrix} \frac{1-m^2}{m^2+1} & \frac{2m}{m^2+1} & 0 \\ \frac{2m}{m^2+1} & \frac{m^2-1}{m^2+1} & 0 \\ \frac{-2cm}{m^2+1} & \frac{2c}{m^2+1} & 1 \end{bmatrix}$$

Example 6.7.7 Derive the appropriate 2D transformation which reflects a figure in point (0.5, 0.5).

Solution : Translating given point to origin

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

Now obtaining reflection of the object about origin

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

Translating point back to original position.

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

The transformation can be given as,

$$R_T = T \cdot M \cdot T^{-1}$$

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

Example 6.7.8 Find out the co-ordinates of a figure bounded by (0, 0), (1, 5), (6, 3) (-3, -4) when reflected along the line whose equation is $y = 2x + 4$ and sheared by 2 units in x direction and 2 units in y direction.

Solution : Equation of the line : $y = 2x + 4$

\therefore Slope = 2 and y intercept = 4

\therefore $\theta = 63.43^\circ$

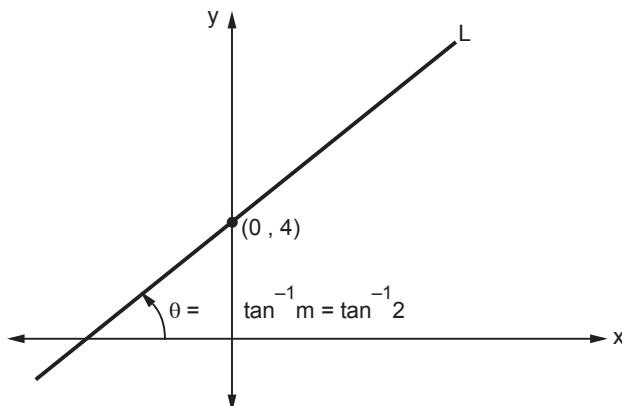


Fig. 6.7.6

Translational matrix can be given as,

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

For matching of given line with x axis we have,

$$R_z = \begin{bmatrix} \cos(-63.43)^\circ & \sin(-63.43)^\circ & 0 \\ -\sin(-63.43)^\circ & \cos(-63.43)^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} 0.4472 & -0.8944 & 0 \\ 0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For reflection about x axis we have,

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

Inverse transformation matrices are

$$R_z^{-1} = \begin{bmatrix} \cos(-63.43)^\circ & -\sin(-63.43)^\circ & 0 \\ +\sin(-63.43)^\circ & \cos(-63.43)^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.4472 & 0.8944 & 0 \\ -0.8944 & 0.4472 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{bmatrix}$$

For shearing along x axis :

$$\begin{aligned} S_x &= \begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

For shearing along y axis

$$\begin{aligned} S_y &= \begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

The resultant transformation matrix can be obtained by

$$R_T = T \cdot R_z \cdot M \cdot R_z^{-1} \cdot T^{-1} \cdot S_x \cdot S_y$$

Final co-ordinates of the given figure can be obtained by

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \cdot R_T$$

Calculations are left for the students as an exercise

Example 6.7.9 Show that 2D reflection through X axis followed by 2-D reflection through the line $Y = -X$ is equivalent to a pure rotation about the origin.

Solution : 2D reflection about X axis

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

2D reflection about $Y = -X$

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

\therefore Resultant transformation matrix

$$\begin{aligned} R_T &= R_x \cdot R' \\ &= \begin{bmatrix} +1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

For pure rotation about origin we have,

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

where θ is angle of rotation

Put $\theta = 90^\circ$

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

$\therefore R_T = R_o$ Hence the result

Example 6.7.10 Show how reflections in the line $y = -x$ and $y = x$ can be performed by a scaling operation followed by rotation.

Solution : We know that,

$$\text{Scaling matrix, } S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and}$$

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

By combining these two matrix we get,

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

By making $\theta = 90^\circ$, $S_x = 1$ and $S_y = -1$ we get,

$$SR = \begin{bmatrix} 1 \times 0 & 1 \times 1 & 0 \\ -(-1) \times 1 & -1 \times 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The resultant matrix is same as the matrix for reflection about line $y = x$.

By making $\theta = 90^\circ$, $S_x = -1$ and $S_y = 1$ we get,

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

The resultant matrix is same as the matrix for reflection about line $y = -x$.

Example 6.7.11 For the triangle ABC A(2, 4), B(4, 6) and C(2, 6) obtained reflection through the line $Y = \frac{1}{2}(x+4)$.

Solution : Equation of line $y = \frac{1}{2}(x+4)$

Slope = $1/2$, y-intercept = 2

$\theta = 26.56$

Translational matrix is,

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

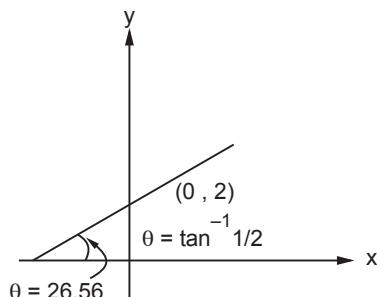


Fig. 6.7.7

For matching with X-axis

$$R_Z = \begin{bmatrix} \cos(-26.56) & \sin(-26.56) & 0 \\ -\sin(-26.56) & \cos(-26.56) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.894 & -0.447 & 0 \\ 0.447 & 0.894 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ for reflection about X-axis}$$

$$R_Z^{-1} = \begin{bmatrix} 0.894 & 0.447 & 0 \\ -0.447 & 0.894 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$RT = T R_Z M R_Z^{-1} T^{-1}$$

Final co-ordinates

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} RT$$

Example 6.7.12 Write pseudo C algorithm to perform shear in two dimensions.

Solution : Float a, b, R_a, R_b, Sh_x, Y_{ref}, Sh_y, x_{ref}, Sh

Procedure : shear (a, b, Sh_x, Y_{ref}, Sh_y, x_{ref}, Sh)

{

```

if (sh == 0) ; shearx
{
    Ra = a + Shx * (b - yref);
    Rb = b ;
}
else; sheary
{
    Ra = a ;
    Rb = Shy * (a - xref) + b ;
}
}
```

Example 6.7.13 Perform X-shear and Y-shear on a triangle having A(2, 1), B(4, 3), C(2, 3).

Consider the constant value b = c = 2.

SPPU : Dec.-08, Marks 8

Solution :

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 4 & 3 & 0 \\ 2 & 3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 5 & 0 \\ 10 & 11 & 0 \\ 8 & 7 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}\therefore A' &= [4, 5] \\ B' &= [10, 11] \\ C' &= [8, 7]\end{aligned}$$

Example 6.7.14 Scale the polygon with co-ordinates A (4, 5), B (8, 10) and C (8, 2) by 2 units in x-direction and 3 units in y-direction. Find the transformed A, B and C points.

SPPU : Dec.-14, Marks 3

Solution :

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} 4 & 5 & 1 \\ 8 & 10 & 1 \\ 8 & 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 15 & 1 \\ 16 & 30 & 1 \\ 16 & 6 & 1 \end{bmatrix}$$

The transformed points are A' = (8, 15), B' = (16, 30) and C' = (16, 6).

Review Questions

- | | |
|---------------------------------------------------------------------------------------------------|----------------------------|
| 1. Write 2D transformation matrices of shearing. | SPPU : May-14, Marks 2 |
| 2. Explain the terms shear and reflection. | SPPU : May-07, 12, Marks 6 |
| 3. Write a short note on 2D shearing transforms. | SPPU : May-08, Marks 5 |
| 4. What is shear transformation ? Explain X-shear and Y-shear. | SPPU : Dec.-12, Marks 5 |
| 5. Write homogeneous transformation matrix for 2-D reflection with respect to origin and 2D year. | SPPU : May-16, Marks 2 |
| 6. Write transformation matrix for 2-D reflection w.r.t. Y-axis. | SPPU : May-19, Marks 2 |

6.8 Inverse Transformation

SPPU : Dec.-06,18, May-07, 08

- When we apply any transformation to point (x, y) we get a new point (x', y'). Sometimes it may require to undo the applied transformation. In such a case we have to get original point (x, y) from the point (x', y'). This can be achieved by inverse transformation.
- The inverse transformation uses the matrix inverse of the transformation matrix to get the original point (x, y). The inverse of a matrix is another matrix such that when the two are multiplied together, we get the identity matrix.
- If the inverse of matrix T is T^{-1} , then

$$T T^{-1} = T^{-1}T = I \quad \dots (6.8.1)$$

where I is the identity matrix with all elements along the major diagonal having value 1, and all other elements having value zero.

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

- The elements for the inverse matrix T^{-1} can be calculated from the elements of T as

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T} \quad \dots (6.8.2)$$

where t_{ij}^{-1} is the element in the i^{th} row and j^{th} column of T^{-1} , and M_{ji} is the $(n - 1)$ by $(n - 1)$ submatrix obtained by deleting the j^{th} row and i^{th} column of the matrix A. The $\det M_{ji}$ and $\det T$ is the determinant of the M_{ji} and T matrices.

- The determinant of a 2×2 matrix is

$$\det \begin{vmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{vmatrix} = t_{11} \cdot t_{22} - t_{12} \cdot t_{21} \quad \dots (6.8.3)$$

- The determinant of a 3×3 matrix is

$$\det T = t_{11} \cdot (t_{22} t_{33} - t_{23} t_{32}) - t_{12} \cdot (t_{21} t_{33} - t_{23} t_{31}) + t_{13} \cdot (t_{21} t_{32} - t_{22} t_{31}) \dots (6.8.4)$$

- In general form, the determinant of T is given by

$$\det T_j = \sum t_{ij} (-1)^{i+j} \det M_{ij} \quad \dots (6.8.5)$$

where M_{ij} is the submatrix formed by deleting row i and column j from matrix T.

- The inverse of the homogeneous co-ordinate transformation matrix can be given as

$$\begin{bmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{bmatrix}^{-1} = \frac{1}{ae - bd} \begin{bmatrix} e & -d & 0 \\ -b & a & 0 \\ bf - ce & cd - af & ae - bd \end{bmatrix}$$

- It is important to note that the elements of inverse matrix T^{-1} can be calculated from the element of T as

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T} \quad \dots (6.8.6)$$

In the above equation the term $\det T$ is in the denominator. Hence, we can obtain an inverse matrix if and only if the determinant of the matrix is nonzero.

Review Question

- Explain inverse transformation and derive the matrix for inverse transformation.

SPPU : Dec.-06, 18, May-07, 08, Marks 8



UNIT - III

7

3D Transformations and Projections

Syllabus

3-D transformations : introduction, 3-D transformations - Translation, scaling, rotation and shear, rotation about an arbitrary axis.

Projections : Parallel (Oblique: Cavalier, Cabinet and orthographic: isometric, diametric, trimetric) and Perspective (Vanishing Points - 1 point, 2 point and 3 point)

Contents

7.1	Introduction	
7.2	3D Geometry	
7.3	Primitives	
7.4	Translation	Dec.-05, 07, 08,
		May-05, 07, 08, · · · Marks 2
7.5	Scaling	May-05, 07, 08, 16,
		Dec.-05, 07, 08, 16 · · · Marks 2
7.6	Rotation	May-06, 07, 11, 16,
		Dec.-16, · · · Marks 8
7.7	Reflection	
7.8	Shears	
7.9	Rotation about Arbitrary Axis	May-06, 08, 11, 12, 14, 17, 19
		Dec.-06, 07, 08, 11, 14, 19 Marks 12
7.10	Reflection with Respect to Given Plane	May-05, 07, Dec.-05, 06, Marks 6
7.11	Concept of Parallel and Perspective Projections	Dec.-08, 10, 18
		May-07, 09, 12, 13, 17, 19 Marks 8
7.12	Types of Parallel Projections	May-05, 06,
		Dec.-05, 06, 08, 15, · · · Marks 8
7.13	Types of Perspective Projections	May-05, 06, 13, 18
		Dec.-05, 06, 08, · · · Marks 8
7.14	Summary of Various Types of Projections	

7.1 Introduction

- Some graphics applications are two-dimensional such as charts and graphics, certain maps and so on. However, to create a realistic picture, scene or model we need to represent it in 3D graphics.
- The creation of realistic pictures is an important task in fields such as simulation, design, entertainment, advertising, research, education, command and control.
- To create a realistic picture we must process the scene or picture through viewing co-ordinate transformations and projection routines that transform three-dimensional viewing co-ordinates onto two-dimensional device co-ordinates.
- We must identify the visible parts of the scene for a selected view and we must apply the surface rendering algorithms to create a realistic scene or picture.
- In this chapter, we begin our study with 3D geometry, and we take an overview of 3D primitives. We then study the different aspects of 3D graphics such as techniques to achieve realism, 3D transformations, projections, viewing transformations and 3D clipping.

7.2 3D Geometry

- In 3D geometry, we need additional, third axis to specify the co-ordinates of any point. Here, the three axes are arranged at right angles to each other and label the width and height axes x and y, respectively, to correspond to the two dimensional case. The third axis, for depth, is named the z-axis.
- There are two types of three dimensional reference system according to the orientation for the co-ordinate axes : **Right handed system** and **left handed system**. The right handed system uses the right hand thumb to point the positive z direction when we imagine the fingers curling from positive x-axis to the positive y-axis (through 90°) grasping the z-axis, as shown in the Fig. 7.2.1.

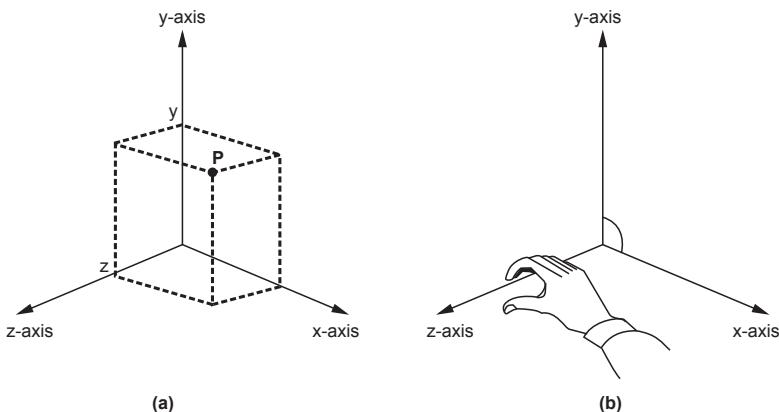


Fig. 7.2.1 Right handed system

- In left handed Cartesian co-ordinate system, the left hand thumb is used to point the positive z direction when we imagine the fingers of the left hand curl from the positive x-axis to the positive y-axis (through 90°) to grasp the z-axis, as shown in the Fig. 7.2.2.

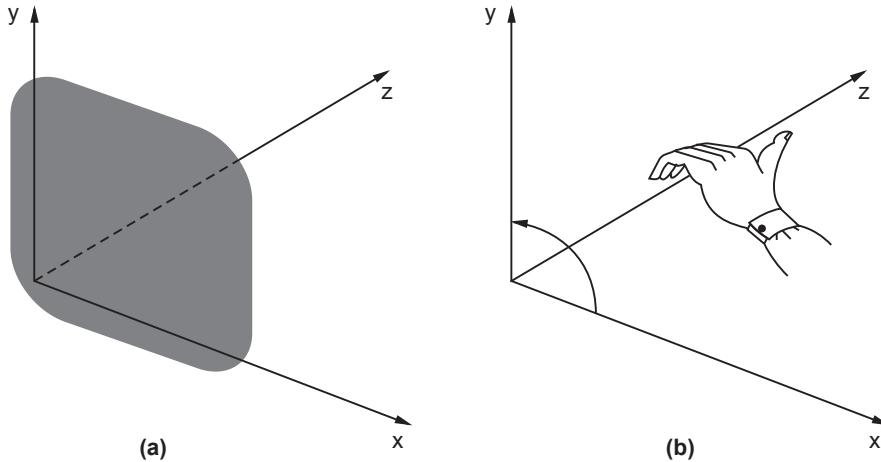


Fig. 7.2.2 Left handed system

- Here, we adopt right handed system for computer graphics. Now, with this co-ordinate system we can specify any point in space by the order triple (x, y, z).
- In 3D geometry, the line is specified by a pair of equations.

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1} \text{ and } \frac{z - z_1}{x - x_1} = \frac{z_2 - z_1}{x_2 - x_1}$$

where (x_1, y_1, z_1) and (x_2, y_2, z_2) are the two points which specify the line.

- A plane is specified by a single equation of the form

$$A_x + B_y + C_z + D = 0$$

- Notice that one of the constants, (for example A , if it is not zero) may be divided out of the equation so that

$$x + B_1 y + C_1 z + D_1 = 0$$

Where, $B_1 = B/A$, $C_1 = C/A$ and $D_1 = D/A$

- From above equation we can say that there are three constants B_1 , C_1 and D_1 are required to specify the plane. The equation for a particular plane may be determined if we know the co-ordinates of three point (not all in a line) which lie within it : (x_1, y_1, z_1) , (x_2, y_2, z_2) and (x_3, y_3, z_3) .
- We can determine the equation in the following manner. Since each point is in the plane, it must satisfy the plane's equation.

$$\begin{aligned}x_1 + B_1 y_1 + C_1 z_1 + D_1 &= 0 \\x_2 + B_1 y_2 + C_1 z_2 + D_1 &= 0 \\x_3 + B_1 y_3 + C_1 z_3 + D_1 &= 0\end{aligned}$$

- Now we have three unknowns and three equations. Therefore, by using simultaneous equations we can get the values of three unknowns.
- Another normalization is

$$A_2 = A/d, \quad B_2 = B/d, \quad C_2 = C/d \quad \text{and} \quad D_2 = D/d$$

Where $d = \sqrt{A^2 + B^2 + C^2}$

- The value of this selection is that the distance between a point (x, y, z) and the plane is given by

$$L = |A_2 x + B_2 y + C_2 z + D_2|$$

- The absolute value indicates the distance and sign of the quantity in the absolute value bars indicates on which side of the plane the point lies.
- Another way of specifying a plane is by a single point in the plane and the direction perpendicular to the plane. A vector perpendicular to plane is called **Normal Vector**, N . Where $[N_x, N_y, N_z]$ are the displacements for the normal vector, and (x_p, y_p, z_p) are the co-ordinates of a point in the plane, as shown in the Fig. 7.2.3.
- If (x, y, z) is to be an arbitrary point in the plane, then $[(x - x_p)(y - y_p)(z - z_p)]$ is a vector parallel to the plane.

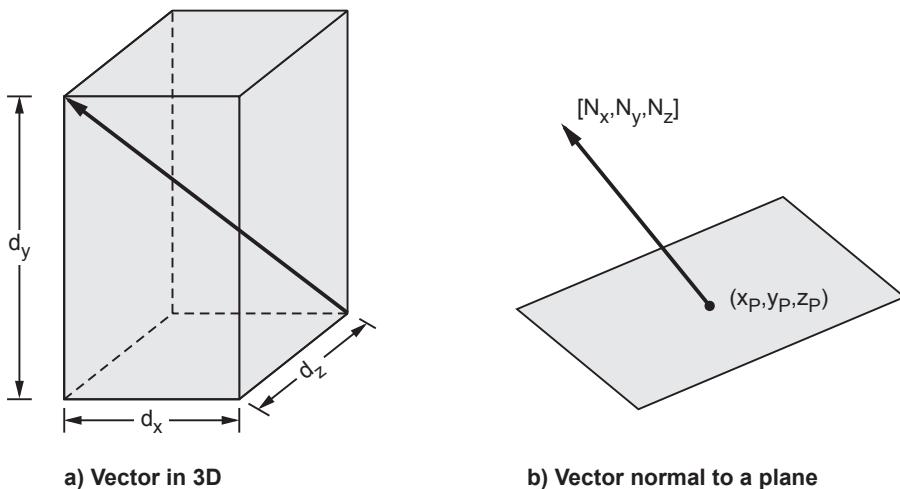


Fig. 7.2.3

- Let us consider, $A = [A_x \ A_y \ A_z]$ and $B = [B_x \ B_y \ B_z]$ are the vectors. The dot product of these two vector is the sum of the products of their corresponding components and is given by

$$A \cdot B = A_x B_x + A_y B_y + A_z B_z$$

- This product is known as **vector dot product**. We can derive an equation for the plane by using vector dot product.
- The result of the dot product is equal to the product of the lengths of the two vectors times the cosine of the angle between them, as shown in the Fig. 7.2.4.
- The angle between any vector parallel to a plane and the normal vector to the plane is $\pi/2$ radians. Since the cosine of $\pi/2$ is 0, we know that the dot product of the normal vector with a vector parallel to the plane is 0.
- We can find a vector parallel to the plane by taking the difference of two points within the plane. Therefore, if

$$N_x(x - x_p) + N_y(y - y_p) + N_z(z - z_p) = 0$$

is true then the vector formed by the difference between (x, y, z) and (x_p, y_p, z_p) must be parallel to the plane.

- We know that (x, y, z) is a point in the plane and since (x_p, y_p, z_p) is in the plane we can get to (x, y, z) by moving along a vector parallel to the plane. This is why the above equation is called an **equation for the plane**.

Review Question

- Explain the 3D co-ordinate systems.

7.3 Primitives

- We have 3D primitives to draw points, lines and plane in three dimensions. Here, we have to provide the three co-ordinate specification instead of two.
- Let us see three-dimensional LINE and MOVE algorithms.

Algorithm 1 : 3D Absolute move

MOVE_ABS_3D (X, Y, Z)

Arguments X, Y, Z are the co-ordinates of point to move the pen to

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position co-ordinates

BEGIN

DF_CUR_X \leftarrow X ;

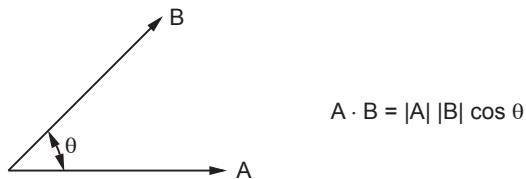


Fig. 7.2.4 Dot product

```

    DF_CUR_Y ← Y ;
    DF_CUR_Z ← Z ;
    DF_ENTER (1) ;
    RETURN ;
END ;

```

Algorithm 2 : 3D Relative move

MOVE_REL_3D(DX, DY, DZ)

Arguments DX, DY, DZ specify the changes to be made in current pen position

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z ; current pen position co-ordinates

BEGIN

```

        DF_CUR_X ← DF_CUR_X + DX ;
        DF_CUR_Y ← DF_CUR_Y + DY ;
        DF_CUR_Z ← DF_CUR_Z + DZ ;
        DF_ENTER (1) ;
        RETURN ;

```

END ;

Algorithm 3 : 3D Absolute line drawing routine

LINE_ABS_3D(X, Y, Z)

Arguments X, Y, Z are the co-ordinates of point to draw the line to

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z ; current pen position co-ordinates

BEGIN

```

        DF_CUR_X ← X ;
        DF_CUR_Y ← Y ;
        DF_CUR_Z ← Z ;
        DF_ENTER (2) ;
        RETURN

```

END;

Algorithm 4 : 3D Relative line drawing routine

LINE_REL_3D (DX, DY, DZ)

Arguments DX, DY, DZ specify the displacement over which a line is to be drawn

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z ; current pen position coordinates

BEGIN

```

        DF_CUR_X ← DF_CUR_X + DX ;
        DF_CUR_Y ← DF_CUR_Y + DY ;
        DF_CUR_Z ← DF_CUR_Z + DZ ;
        DF_ENTER (2) ;
        RETURN ;

```

END ;

Algorithm 5 : 3D Absolute polygon drawing routine

POLYGON_ABS_3D(AX, AY, AZ, N)

Arguments N specifies number of polygon sides AX, AY, AZ arrays of the co-ordinates of the vertices

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z; current pen position co-ordinates

BEGIN

```

IF N < 3 THEN RETURN ERROR ; less number of sides
DF_CUR_X ← AX[N] ;
DF_CUR_Y ← AY[N] ;
DF_CUR_Z ← AZ[N] ;
DF_ENTER (N) ;
FOR J = 1 TO N DO LINE_ABS_3D(AX[I], AY[I], AZ[I]);
RETURN ;

```

END ;

Algorithm 6 : 3D Relative polygon drawing routine

Arguments N specify the number of polygon sides AX, AY, AZ arrays of displacement for the polygon sides.

Global DF_CUR_X, DF_CUR_Y, DF_CUR_Z ; current pen position co-ordinates

BEGIN

```

If N < 3 THEN RETURN ERROR; less number of sides.
    DF_CUR_X ← DF_CUR_X + AX [1] ; [Move to
    DF_CUR_Y ← DF_CUR_Y + AY [1] ; starting
    DF_CUR_Z ← DF_CUR_Z + AZ [1] ; vertex ]
    TEMPX ← DF_CUR_X ; [Save starting
    TEMPY ← DF_CUR_Y ; vertex for
    TEMPZ ← DF_CUR_Z ; closing the polygon]
    DF_ENTER (N) ;
    FOR I=2 TO N DO LINE_REL_3D (AX[I], AY[I], AZ[I]) ;
    LINE_ABS_3D (TEMPX, TEMPY, TEMPZ) ; close the polygon
    RETURN ;

```

END;

Review Question

1. Write a short note on 3D primitives.

7.4 Translation**SPPU : Dec.-05,07,08, May-05,07,08**

- Three dimensional transformation matrix for translation with homogeneous coordinates is as given below. It specifies three coordinates with their own translation factor.

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

$$\therefore P' = P \cdot T$$

$$\begin{aligned} \therefore [x' \ y' \ z' \ 1] &= [x \ y \ z \ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \\ &= [x + t_x \ y + t_y \ z + t_z \ 1] \quad \dots (7.4.1) \end{aligned}$$

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

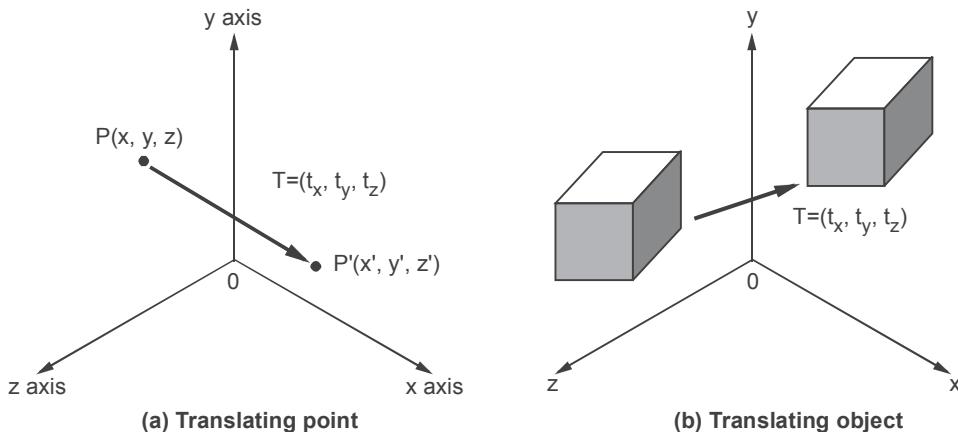


Fig. 7.4.1 3 D translation

Review Question

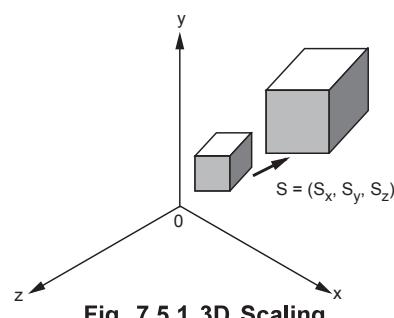
- Obtain the 3D transformation matrices for translation.

SPPU : Dec.-05,07,08, May-05,07,08, Marks 2

7.5 Scaling

SPPU : May-05,07,08,16, Dec.-05,07,08,16

- Three dimensional transformation matrix for scaling with homogeneous co-ordinates is as given below.
- It specifies three co-ordinates with their own scaling factor.



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

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

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

$$= [x \cdot S_x \quad y \cdot S_y \quad z \cdot S_z \quad 1] \quad \dots (7.5.1)$$

- A scaling of an object with respect to a selected fixed position can be represented with the following transformation sequence.
 1. Translate the fixed point to the origin.
 2. Scale the object.
 3. Translate the fixed point back to its original position.

Review Question

1. Obtain 3D transformation matrix for scaling.

SPPU : May-05, 07, 08, 16, Dec.-05, 07, 08, 16, Marks 2

7.6 Rotation

SPPU : May-06, 07, 11, 16, Dec.-16

- Unlike two dimensional rotation, where all transformations are carried out in the xy plane, a three-dimensional rotation can be specified around any line in space. Therefore, for three dimensional rotation we have to specify an axis of rotation about which the object is to be rotated along with the angle of rotation.
- The easiest rotation axes to handle are those that are parallel to the co-ordinate axes. It is possible to combine the co-ordinate axis rotations to specify any general rotation.

Co-ordinate axes rotations

- Three dimensional transformation matrix for each co-ordinate axes rotations with homogeneous co-ordinate are as given below.

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

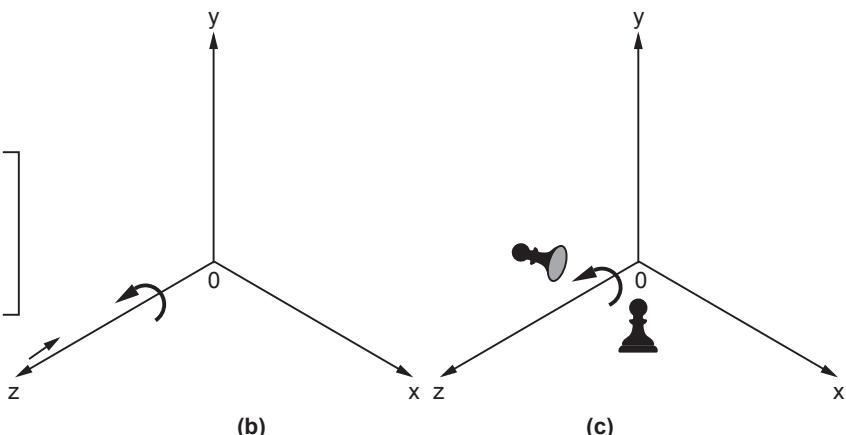


Fig. 7.6.1 Rotation about z axis

- The positive value of angle θ indicates counterclockwise rotation. For clockwise rotation value of angle θ is negative.

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

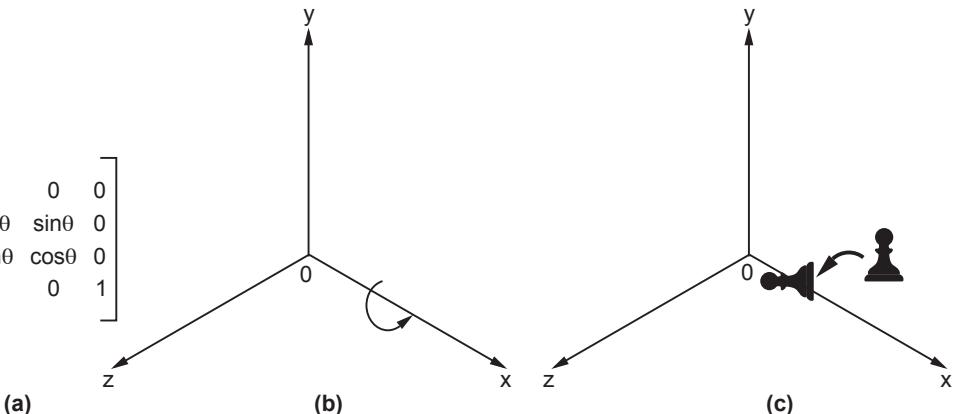


Fig. 7.6.2 Rotation about x axis

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

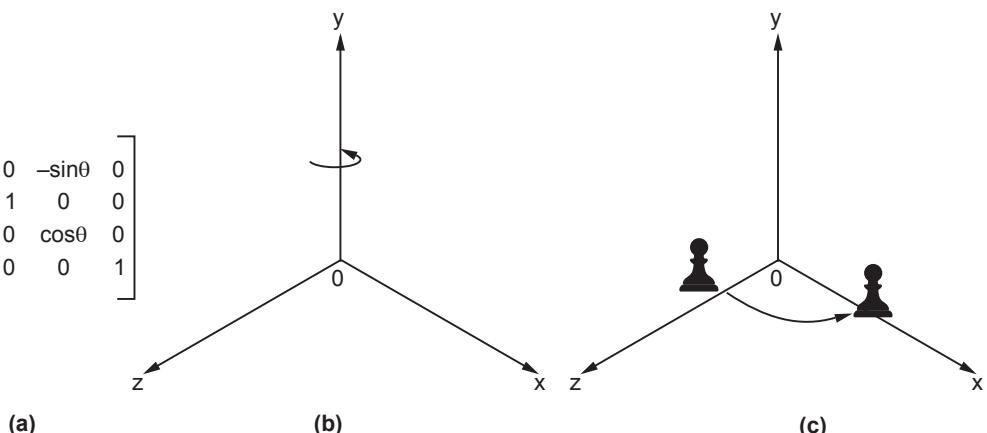


Fig. 7.6.3 Rotation about y axis

Review Questions

1. Obtain the 3D transformation matrix for rotation about z-axis.

SPPU : May-11, Marks 3

2. Obtain 3D transformation matrix for rotation.

SPPU : May-06, 07, Marks 8

3. Explain the rotation about all co-ordinate axis.

4. Give homogenous transformation matrix for 3D rotation with respect to y axis.

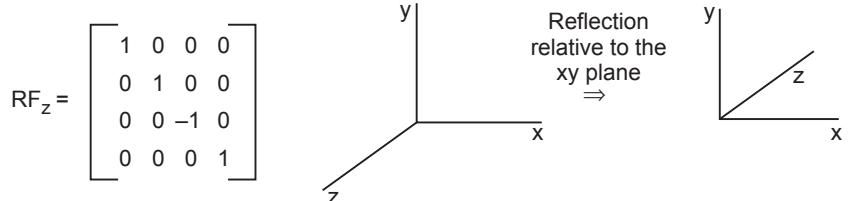
SPPU : May-16, Marks 2

5. Write a matrices for 3D object rotation about x-axis, y-axis, z-axis.

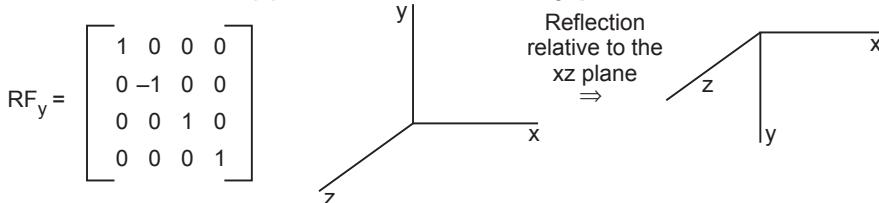
SPPU : Dec.-16, Marks 6

7.7 Reflection

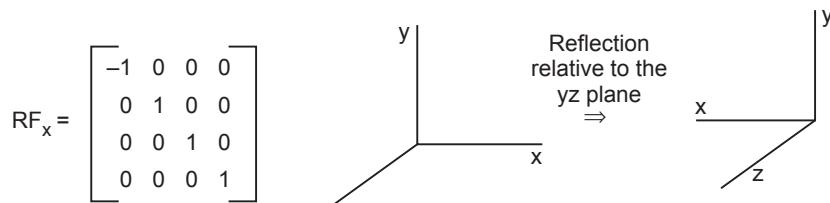
- A three-dimensional reflection can be performed relative to a selected reflection axis or with respect to a selected reflection plane.
- The three-dimensional reflection matrices are set up similarly to those for two dimensions.
- Reflections relative to a given axis equivalent to 180° rotations about that axis.
- Reflections with respect to a plane are equivalent to 180° rotations in four dimensional space.
- The reflections relative to xy, yz and zx planes are as shown in Fig. 7.7.1.



(a) Reflection relative to xy plane



(b) Reflection relative to xz plane



(c) Reflection relative to yz plane
Fig. 7.7.1

Review Question

1. Obtain 3D matrices for reflection relative to plane.

7.8 Shears

- Shearing transformations are used to modify the shape of the object and they are useful in three-dimensional viewing for obtaining general projection transformations.
- The transformation matrix to produce shears relative to x, y and z axes are as shown in the Fig. 7.8.1.

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

Fig. 7.8.1 Shear transformation matrices

Review Question

- Give the shear transformation matrices.

7.9 Rotation about Arbitrary Axis

SPPU : May-06,08,11,12,14,17,19, Dec.-06,07,08,11,14,19

- A rotation matrix for any axis that does not coincide with a coordinate axis can be set up as a composite transformation involving combinations of translations and the coordinate-axes rotations.
- In a special case where an object is to be rotated about an axis that is parallel to one of the coordinate axes we can obtain the resultant coordinates with the following transformation sequence.
 - Translate the object so that the rotation axis coincides with the parallel coordinate axis.
 - Perform the specified rotation about that axis.
 - Translate the object so that the rotation axis is moved back to its original position.
- When an object is to be rotated about an axis that is not parallel to one of the coordinate axes, we have to perform some additional transformations. The sequence of these transformations is given below.
 - Translate the object so that rotation axis specified by unit vector u passes through the coordinate origin. (See Fig. 7.9.1 (a) and (b))
 - Rotate the object so that the axis of rotation coincides with one of the coordinate axes. Usually the z axis is preferred. To coincide the axis of rotation to z axis we have to first perform rotation of unit vector u about x axis to bring it into xz plane and then perform rotation about y axis to coincide it with z axis.
(See Fig. 7.9.1 (c) and (d))

3. Perform the desired rotation θ about the z axis.
4. Apply the inverse rotation about y axis and then about x axis to bring the rotation axis back to its original orientation.
5. Apply the inverse translation to move the rotation axis back to its original position.

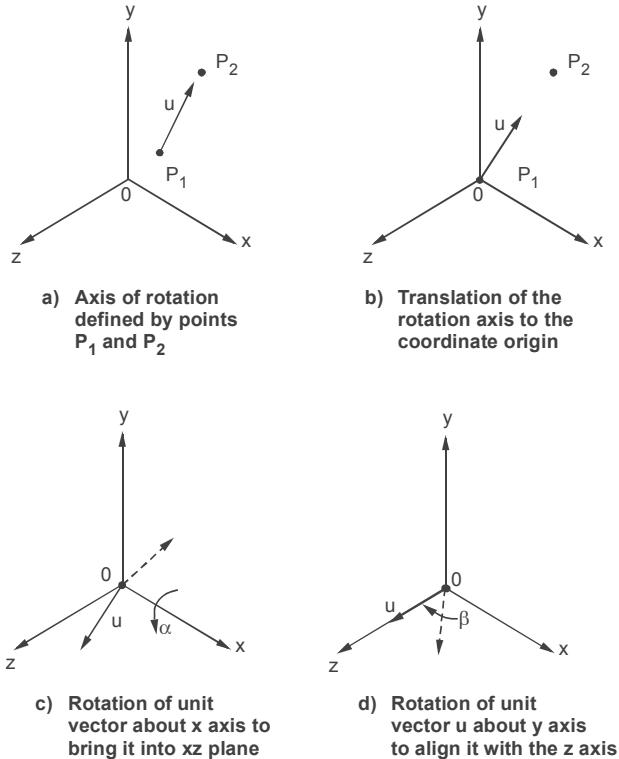


Fig. 7.9.1

- As shown in the Fig. 7.9.1 (a) the rotation axis is defined with two coordinate points P_1 and P_2 and unit vector u is defined along the rotation of axis as

$$u = \frac{V}{|V|} = (a, b, c)$$

where V is the axis vector defined by two points P_1 and P_2 as

$$V = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

- The components a , b , and c of unit vector u are the direction cosines for the rotation axis and they can be defined as

$$a = \frac{x_2 - x_1}{|V|}, \quad b = \frac{y_2 - y_1}{|V|}, \quad c = \frac{z_2 - z_1}{|V|}$$

- As mentioned earlier, the first step in the transformation sequence is to translate the object to pass the rotation axis through the coordinate origin. This can be accomplished by moving point P_1 to the origin. The translation is as given below.

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

- Now we have to perform the rotation of unit vector u about x axis. The rotation of u around the x axis into the xz plane is accomplished by rotating u' ($0, b, c$) through angle α into the z axis and the cosine of the rotation angle α can be determined from the dot product of u' and the unit vector u_z ($0, 0, 1$) along the z axis.

$$\cos \alpha = \frac{\mathbf{u}' \cdot \mathbf{u}_z}{|\mathbf{u}'| |\mathbf{u}_z|} = \frac{c}{d} \quad \text{where } \mathbf{u}' (0, b, c) = b\mathbf{J} + c\mathbf{K} \text{ and}$$

$$\mathbf{u}_z (0, 0, 1) = \mathbf{K}$$

$$= \frac{c}{|\mathbf{u}'| |\mathbf{u}_z|}$$

$$= \frac{c}{|\mathbf{u}'|}$$

$$= \frac{c}{d}$$

Since $|\mathbf{u}_z| = 1$

where d is the magnitude of u' .

$$d = \sqrt{b^2 + c^2}$$

- Similarly, we can determine the sine of α from the cross product of u' and u_z .

$$\mathbf{u}' \times \mathbf{u}_z = u_x |\mathbf{u}'| |\mathbf{u}_z| \sin \alpha \quad \dots (7.9.1)$$

and the Cartesian form for the cross product gives us

$$\mathbf{u}' \times \mathbf{u}_z = \mathbf{u}_x \cdot \mathbf{b} \quad \dots (7.9.2)$$

- Equating the right sides of equations (7.9.1) and (7.9.2) we get

$$u_x |\mathbf{u}'| |\mathbf{u}_z| \sin \alpha = u_x \cdot \mathbf{b}$$

$$\therefore |\mathbf{u}'| |\mathbf{u}_z| \sin \alpha = b$$

$$\therefore \sin \alpha = \frac{b}{|\mathbf{u}'| |\mathbf{u}_z|}$$

$$= \frac{b}{d} \quad \text{since } |\mathbf{u}_z| = 1 \text{ and } |\mathbf{u}'| = d$$

- This can also be verified graphically as shown in Fig. 7.9.2.
- By substituting values of $\cos \alpha$ and $\sin \alpha$ the rotation matrix R_x can be given as

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

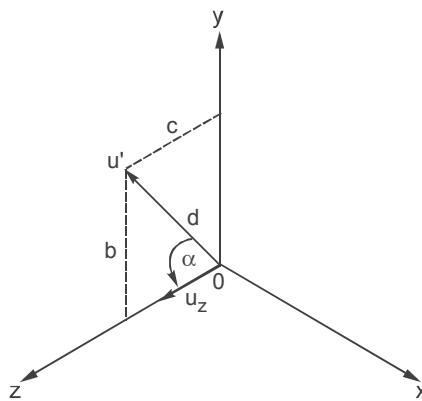


Fig. 7.9.2

- Next we have to perform the rotation of unit vector about y axis. This can be achieved by rotating u'' ($a, 0, d$) through angle β onto the z axis. Using similar equations we can determine $\cos \beta$ and $\sin \beta$ as follows.

- We have angle of rotation = $-\beta$

$$\therefore \cos(-\beta) = \cos \beta = \frac{\mathbf{u''} \cdot \mathbf{u}_z}{|\mathbf{u''}| |\mathbf{u}_z|} \text{ where } \mathbf{u''} = a\mathbf{I} + d\mathbf{K} \text{ and}$$

$$\begin{aligned} \mathbf{u}_z &= \mathbf{K} \\ &= \frac{d}{|\mathbf{u''}| |\mathbf{u}_z|} \\ &= \frac{d}{|\mathbf{u''}|} && \because |\mathbf{u}_z| = 1 \\ &= \frac{d}{\sqrt{a^2 + d^2}} \end{aligned}$$

- Consider cross product of u'' and u_z

$$\begin{aligned} \mathbf{u''} \times \mathbf{u}_z &= \mathbf{u}_y |\mathbf{u''}| |\mathbf{u}_z| \sin(-\beta) \\ &= -\mathbf{u}_y |\mathbf{u''}| |\mathbf{u}_z| \sin \beta && \because \sin(-\theta) = -\sin \theta \end{aligned}$$

- Cartesian form of cross product gives us

$$\mathbf{u''} \times \mathbf{u}_z = \mathbf{u}_y (+a)$$

- Equating above equations,

$$-|\mathbf{u''}| |\mathbf{u}_z| \sin \beta = a$$

$$\begin{aligned} \therefore \sin \beta &= \frac{-a}{|\mathbf{u''}| |\mathbf{u}_z|} \\ &= \frac{-a}{|\mathbf{u''}|} && \because |\mathbf{u}_z| = 1 \\ &= \frac{-a}{\sqrt{a^2 + d^2}} \end{aligned}$$

but we have, $d = \sqrt{b^2 + c^2}$

$$\therefore \cos \beta = \frac{d}{\sqrt{a^2 + d^2}} = \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}}$$

$$\text{and } \sin \beta = \frac{-a}{\sqrt{a^2 + d^2}} = \frac{-a}{\sqrt{a^2 + b^2 + c^2}}$$

- By substituting values of $\cos \beta$ and $\sin \beta$ in the rotation matrix R_y can be given as

$$R_y = \begin{bmatrix} \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}} & 0 & \frac{+a}{\sqrt{a^2 + b^2 + c^2}} & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}} & 0 \\ \frac{\sqrt{b^2 + c^2}}{\sqrt{a^2 + b^2 + c^2}} & 0 & 0 & 1 \end{bmatrix}$$

Let $\lambda = \sqrt{b^2 + c^2}$ and $|V| = \sqrt{a^2 + b^2 + c^2}$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{\lambda} & \frac{+b}{\lambda} & 0 \\ 0 & \frac{-b}{\lambda} & \frac{c}{\lambda} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{+a}{|V|} & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & \frac{\lambda}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

\therefore Resultant rotation matrix $R_{xy} = R_x \cdot R_y$

$$R_{xy} = \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{|V|\lambda} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{|V|\lambda} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From equation 6.8.6 of section 6.8 we have

$$t_{ij}^{-1} = \frac{(-1)^{i+j} \det M_{ji}}{\det T}$$

Using above equation we get inverse of R_{xy} as

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{|V|\lambda} & \frac{-ac}{|V|\lambda} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse of translation matrix can be given as

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

- With transformation matrices T and R_{xy} we can align the rotation axis with the positive z axis. Now the specified rotation with angle θ can be achieved by rotation transformation as given below.

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

- To complete the required rotation about the given axis, we have to transform the rotation axis back to its original position. This can be achieved by applying the inverse transformations T^{-1} and R_{xy}^{-1} . The overall transformation matrix for rotation about an arbitrary axis then can be expressed as the concatenation of five individual transformations.

$$R(\theta) = T \cdot R_{xy} \cdot R_z \cdot R_{xy}^{-1} \cdot T^{-1}$$

$$\text{i.e. } R(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{bmatrix} \begin{bmatrix} \frac{\lambda}{|V|} & 0 & \frac{a}{|V|} & 0 \\ \frac{-ab}{\lambda|V|} & \frac{c}{\lambda} & \frac{b}{|V|} & 0 \\ \frac{-ac}{\lambda|V|} & \frac{-b}{\lambda} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\lambda}{|V|} & \frac{-ab}{\lambda|V|} & \frac{-ac}{\lambda|V|} & 0 \\ 0 & \frac{c}{\lambda} & \frac{-b}{\lambda} & 0 \\ \frac{a}{|V|} & \frac{b}{|V|} & \frac{c}{|V|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{bmatrix}$$

Review Questions

- Derive the transformation matrix for rotation about any arbitrary axis.

SPPU : May-11, 12,17, Marks 10 Dec.-11,19, Marks 12

- Explain the rotation about any arbitrary axis.

SPPU : May-06,08,12,14, Dec.-06, 07, 08,19, Marks 8

3. Explain the concept of 3D rotation about an arbitrary axis with an example.

SPPU : Dec.-14, Marks 7

4. Write transformation matrix for : 3-D rotation about x-axis.

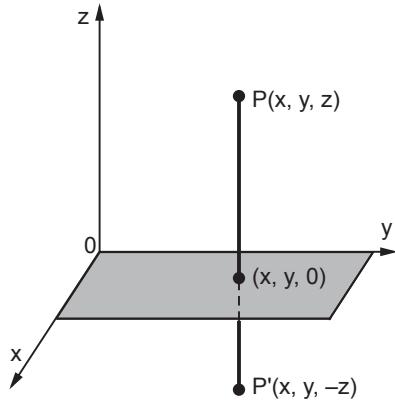
SPPU : May-19, Marks 2

7.10 Reflection with Respect to Given Plane SPPU : May-05,07, Dec.-05,06

7.10.1 Reflection with respect to xy Plane

- Consider point $P(x, y, z)$. The reflection of this point with respect to xy plane is given by point $P'(x, y, -z)$, as shown in Fig. 7.10.1.
- Corresponding to this reflection the transformation matrix can be given as

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



7.10.2 Reflection with respect to Any Plane

Fig. 7.10.1

- Often it is necessary to reflect an object through a plane other than $x = 0$ (yz plane), $y = 0$ (xz plane) or $z = 0$ (xy plane). Procedure to achieve such a reflection (reflection with respect to any plane) can be given as follows :
- Translate a known point P_o , that lies in the reflection plane to the origin of the co-ordinate system.
 - Rotate the normal vector to the reflection plane at the origin until it is coincident with +ve z axis, this makes the reflection plane $z = 0$ co-ordinate plane i.e. xy plane.
 - Reflect the object through $z = 0$ (xy plane) co-ordinate plane.
 - Perform the inverse transformation to those given above to achieve the result.

- Let $P_o (x_o, y_o, z_o)$ be the given known point. Translate this point to the origin by using corresponding translation matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_o & -y_o & -z_o & 1 \end{bmatrix}$$

- Let the normal vector

$$N = n_1 I + n_2 J + n_3 K$$

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

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

- As we want to match this vector with z axis, (so that the plane of reflection will be parallel to xy plane), we will use the same procedure as used in rotation.

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

- As seen earlier for reflection about xy plane we have

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

- Now for inverse transformation we have,

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_o & y_o & z_o & 1 \end{bmatrix}$$

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

\therefore Resultant transformation matrix can be given as

$$R_T = T \cdot R_{xy} \cdot M \cdot R_{xy}^{-1} \cdot T^{-1}$$

Example 7.10.1 Find the matrix for mirror reflection with respect to the plane passing through the origin and having a normal vector whose direction is $M = I + J + K$

Solution : Here, $P_o (0, 0, 0)$ and the plane passes through the origin hence translation matrix is not necessary.

The normal vector $N = I + J + K$

$$\therefore n_1 = 1, n_2 = 1, n_3 = 1$$

$$|N| = \sqrt{3} \quad \text{and} \quad \lambda = \sqrt{2}$$

$$\therefore R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{\sqrt{6}}{0} & \frac{\sqrt{2}}{0} & \frac{\sqrt{3}}{0} & 1 \end{bmatrix}$$

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{+1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

\therefore The reflection matrix is given by

$$R_T = R_{xy} \cdot M \cdot R_{xy}^{-1}$$

$$\therefore R_T = \begin{bmatrix} 1/3 & -2/3 & -2/3 & 0 \\ -2/3 & 1/3 & -2/3 & 0 \\ -2/3 & -2/3 & +1/3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Review Question

- Obtain the 3D-transformation matrices for reflection about an arbitrary plane.

SPPU : May-05, 07, Dec.-05, 06, Marks 6

7.11 Concept of Parallel and Perspective Projections

SPPU : Dec.-08,10,18, May-07,09,12,13,17,19

- After converting the description of objects from world co-ordinates to viewing co-ordinates, we can project the three dimensional objects onto the two dimensional view plane. There are two basic ways of projecting objects onto the view plane : **Parallel projection** and **Perspective projection**.

7.11.1 Parallel Projection

- In parallel projection, z co-ordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane.
- The point of intersection is the **projection of the vertex**.
- We connect the projected vertices by line segments which correspond to connections on the original object.
- As shown in the Fig. 7.11.1, a parallel projection preserves relative proportions of objects but does not produce the realistic views.

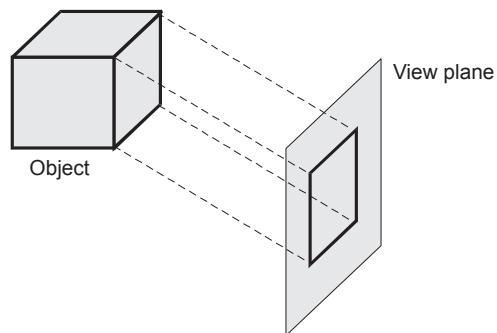


Fig. 7.11.1 Parallel projection of an object to the view plane

7.11.2 Perspective Projection

- The perspective projection, on the other hand, produces realistic views but does not preserve relative proportions.
- In perspective projection, the lines of projection are not parallel. Instead, they all converge at a single point called the **center of projection or projection reference point**.
- The object positions are transformed to the view plane along these converged projection lines and the projected view of an object is determined by calculating the intersection of the converged projection lines with the view plane, as shown in the Fig. 7.11.2.

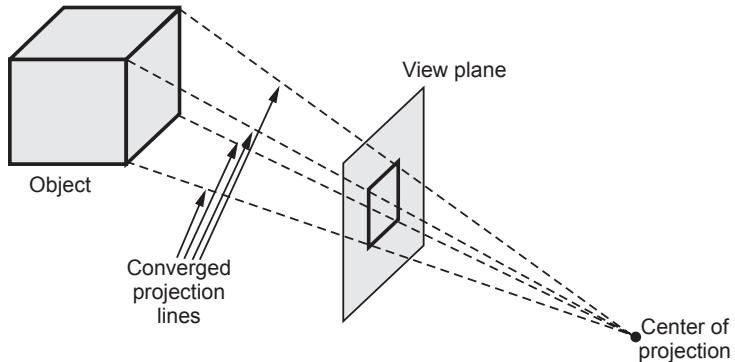


Fig. 7.11.2 Perspective projection of an object to the view plane

Review Questions

- Write a short note on projections.
- Explain parallel projection ?

SPPU : May-12, Marks 8

SPPU : Dec.-08, 10,18, May-07, Marks 5

3. Explain perspective projection ?

SPPU : Dec.-10, 08, May-07, Marks 5

4. What are the types of projections and brief about each type of projections.

SPPU : May-17, Marks 6

4. Write a note on Parallel and Perspective projection.

SPPU : May-13, Marks 8

5. Explain ways of projection 3D objects onto 2D screen in detail.

SPPU : May-09, Marks 8

6. What are the types of projection and write in brief about each type of projections.

SPPU : May-19, Marks 6

7.12 Types of Parallel Projections

SPPU : May-05,06, Dec.-05,06,08,15

- Parallel projections are basically categorized into two types, depending on the relation between the direction of projection and the normal to the view plane.
- When the direction of the projection is normal (perpendicular) to the view plane, we have an **orthographic parallel projection**. Otherwise, we have an **oblique parallel projection**. Fig. 7.12.1 illustrates the two types of parallel projection.

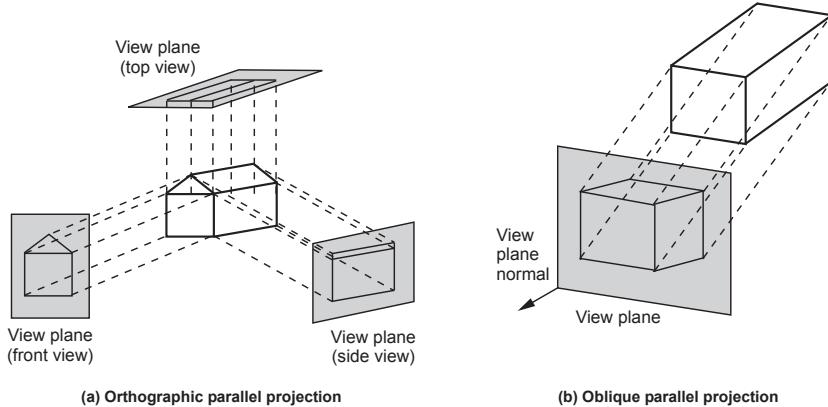


Fig. 7.12.1

7.12.1 Orthographic Projection

- Orthographic Projections are classified as axonometric orthographic projection and multiview orthographic projection.

7.12.1.1 Axonometric Orthographic Projection

- As shown in Fig. 7.12.1 (a), the most common types of orthographic projections are the front projection, top projection and side projection. In all these, the projection plane (view plane) is perpendicular to the principle axis. These projections are often used in engineering drawing to depict machine parts, assemblies, buildings and so on.
- The orthographic projection can display more than one face of an object. Such an orthographic projection is called **axonometric** orthographic projection.

- It uses projection planes (view planes) that are not normal to a principle axis.
- They resemble the perspective projection in this way, but differ in that the foreshortening is uniform rather than being related to the distance from the center of projection.
- Parallelism of lines is preserved but angles are not.
- The most commonly used axonometric orthographic projection is the **isometric** projection.

7.12.1.2 Types of Axonometric Projection

- This is used to overcome the limitation of single orthographic projection which is unable to illustrate the general three-dimensional shape of an object.
- The construction of an axonometric projection is done by using rotation and translation to manipulate the object such that at least three adjacent faces are shown. The result is then projected at infinity onto one of the co-ordinate planes (usually $z=0$ plane) from the centre of projection.
- An axonometric projection shows its true shape only when a face is parallel to the plane of projection. Here, the parallel lines are equally foreshortened.
- The **foreshortening factor** is the ratio of the projected length of line to its true length.
- **Axonometric projections of three types :**
 - **Isometric** : All three principle axes are foreshortened equally.
 - **Dimetric** : Two principle axes are foreshortened equally.
 - **Trimetric** : All three principle axes are foreshortened unequally.
- The Fig. 7.12.2 shows views of isometric, dimetric and trimetric projections of a cube.

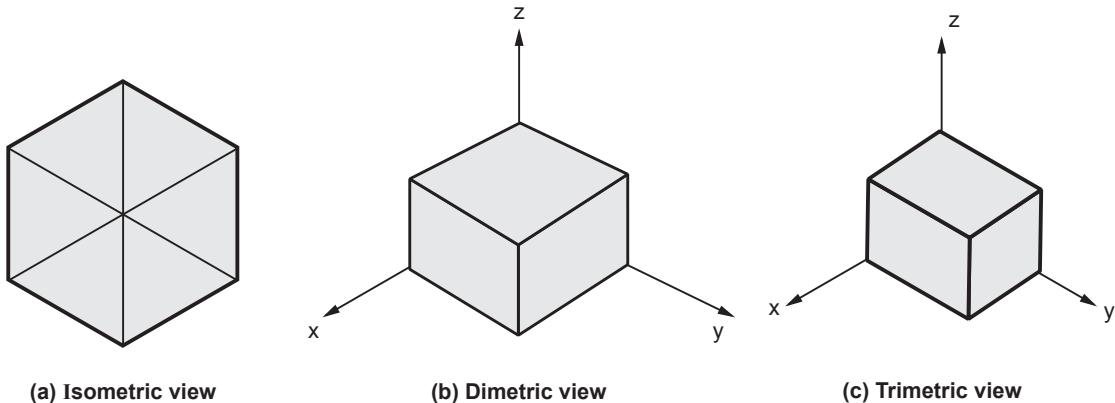


Fig. 7.12.2

1. Trimetric projection :

- It is formed by arbitrary rotations in arbitrary order about any or all of the co-ordinate axes, followed by parallel projection onto the $z=0$ plane. For each projected principle axis (x , y and z), the foreshortening ratios are different. For any specific trimetric projection, the foreshortening ratios are obtained by applying the concatenated transformation matrix to the unit vectors along the principle axes, i.e.

$$\begin{aligned}[U][T] &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} [T] \\ &= \begin{bmatrix} X_x^* & Y_x^* & 0 & 1 \\ X_y^* & Y_y^* & 0 & 1 \\ X_z^* & Y_z^* & 0 & 1 \end{bmatrix}\end{aligned}$$

- Where $[U]$ is the matrix of unit vectors along the untransformed x , y and z axes, respectively and $[T]$ is the concatenated trimetric projection matrix. The foreshortening factors along the projected principle axes are :

$$f_x = \sqrt{(X_x^*)^2 + (Y_x^*)^2}$$

$$f_y = \sqrt{(X_y^*)^2 + (Y_y^*)^2}$$

$$f_z = \sqrt{(X_z^*)^2 + (Y_z^*)^2}$$

Example 7.12.1 Find the resultant co-ordinates and foreshortening factors for trimetric projection on a cube with one corner removed with $\phi = 30^\circ$ rotation about y -axis, followed by a $\theta = 45^\circ$ rotation about x -axis.

Solution : The position vectors of a cube with one corner removed are :

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

The concatenated trimetric projection onto $z = 0$ place is,

$$\begin{aligned}
 [T] &= [R_y][R_x][R_z] \\
 &= \begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \cos\phi & \sin\phi \sin\theta & 0 & 0 \\ 0 & \cos\theta & 0 & 0 \\ \sin\phi & -\cos\phi \sin\theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{2}}{4} & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{1}{2} & -\frac{\sqrt{6}}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Thus, the transformed position vectors are :

$$[X^*] = [X][T] = \begin{bmatrix} 0.5 & -0.612 & 0 & 1 \\ 1.366 & -0.259 & 0 & 1 \\ 1.366 & 0.095 & 0 & 1 \\ 0.933 & 0.272 & 0 & 1 \\ 0.5 & 0.095 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0.866 & 0.354 & 0 & 1 \\ 0.866 & 1.061 & 0 & 1 \\ 0 & 0.707 & 0 & 1 \\ 1.116 & 0.754 & 0 & 1 \end{bmatrix}$$

The foreshortening ratios are :

$$\begin{aligned}
 [U][T] &= \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{2}}{4} & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{1}{2} & -\frac{\sqrt{6}}{4} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{\sqrt{2}}{4} & 0 & 1 \\ 0 & \frac{\sqrt{2}}{2} & 0 & 1 \\ \frac{1}{2} & -\frac{\sqrt{6}}{4} & 0 & 1 \end{bmatrix}
 \end{aligned}$$

And,

$$f_x = \sqrt{\left(\frac{\sqrt{3}}{2}\right)^2 + \left(\frac{\sqrt{2}}{4}\right)^2} = 0.935$$

$$f_y = \frac{\sqrt{2}}{2} = 0.707$$

$$f_z = \sqrt{\left(\frac{1}{2}\right)^2 + \left(-\frac{\sqrt{6}}{4}\right)^2} = 0.791$$

2. Dimetric projection :

- In dimetric projection two foreshortening factors are equal and third arbitrary. It is constructed by a rotation about a y-axis through an angle ϕ followed by a rotation about the x-axis through an angle θ and projection at infinity onto the $z = 0$ plane.
- Considering rotations with ϕ and θ , the mathematical interpretation is :

$$[T] = [R_y][R_x][R_z]$$

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

$$[T] = \begin{bmatrix} \cos\phi & \sin\phi \sin\theta & 0 & 0 \\ 0 & \cos\theta & 0 & 0 \\ \sin\phi & -\cos\phi \sin\theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The unit vectors on the x, y and z principle axis transform to :

$$[U^*] = [U][T] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} \cos\phi & \sin\phi \sin\theta & 0 & 0 \\ 0 & \cos\theta & 0 & 0 \\ \sin\phi & -\cos\phi \sin\theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

- The foreshortening factors are :

$$f_x = \sqrt{\cos^2\phi + \sin^2\phi \sin^2\theta}$$

$$f_y = \sqrt{\cos^2\theta} = \cos\theta$$

$$f_z = \sqrt{\sin^2\phi + \cos^2\phi \sin^2\theta}$$

3. Isometric Projection :

- The problem is that a dimetric projection allows two of the three transformed principle axes to be measured with the same scale factor.
- Measurements along the third transformed principle axis require a different scale factor. If accurate scaling of the dimensions of the projected object is required, it leads to error.
- The solution to the above problem in isometric projection is that all three foreshortening factors are kept equal.
- The projection plane normal makes equal angles with each principle axis. If the projection-plane normal is (d_x, d_y, d_z) , then we require that $|d_x| = |d_y| = |d_z|$.
- The angle that the projected x-axis makes with the horizontal is important in manual construction of isometric projections. Transforming the unit vector along the x-axis using the isometric projection matrix yields.

$$[U_x^*] = [1 \ 0 \ 0 \ 1] = \begin{bmatrix} \cos\phi & \sin\phi \sin\theta & 0 & 0 \\ 0 & \cos\theta & 0 & 0 \\ \sin\phi & -\cos\phi \sin\theta & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= [\cos\phi \ \sin\phi \ \sin\theta \ 0 \ 1]$$

- The angle between projected z-axis and horizontal is :

$$\tan \alpha \frac{y_x^*}{x_x^*} = \frac{\sin\phi \sin\theta}{\cos\phi} = \pm \sin\theta$$

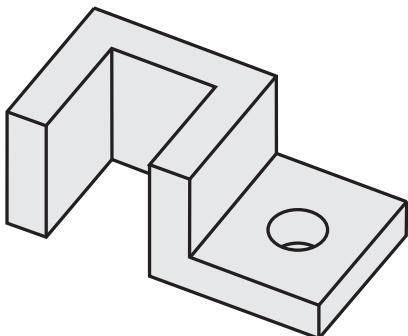
- Since $\sin\phi = \cos\phi$ for $\phi = 45^\circ$,

$$\alpha = \tan^{-1}(\pm \sin 35.26439^\circ) = \pm 30^\circ$$

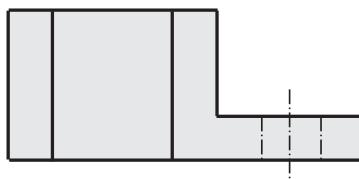
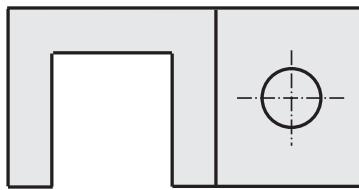
7.12.1.3 Multiview Orthographic Projection

- Most drawings produced and used in industry are **multiview drawings**. Multiview drawings are used to provide accurate three-dimensional object information on two-dimensional media, a means of communicating all of the information necessary to transform an idea or concept into reality.
- Multiview drawings usually require several orthographic projections to define the shape of a three-dimensional object.
- Each orthographic view is a two-dimensional drawing showing only two of the three dimensions of the three-dimensional object.
- Consequently, no individual view contains sufficient information to completely define the shape of the three-dimensional object. All orthographic views must be looked at together to comprehend the shape of the three-dimensional object.

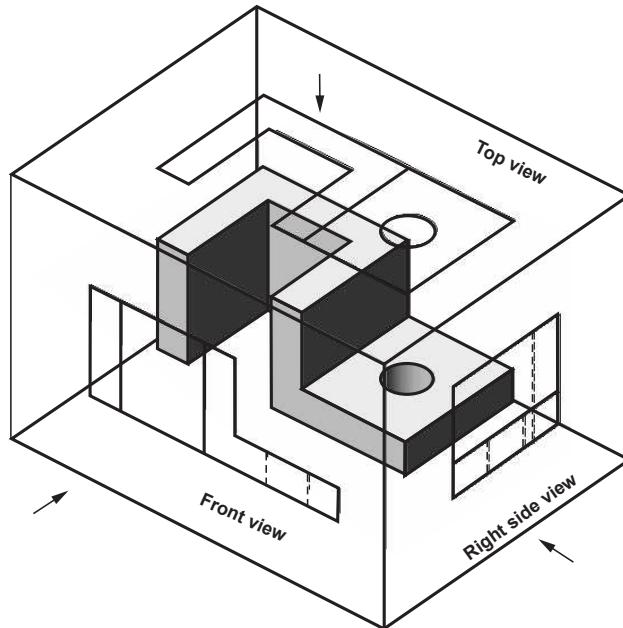
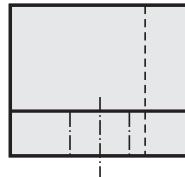
- The arrangement and relationship between the views are therefore very important in multiview drawings.



(a) 3D object



(b) Individual multiviews



(C) Multiview orthographic projection
Fig. 7.12.3

7.12.2 Oblique Projection

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the projection plane.
- Fig. 7.12.4 (b) shows the oblique projection.

- Notice that the view plane normal and the direction of projection are not the same.
- The oblique projections are further classified as the **cavalier** and **cabinet** projections.
- For the cavalier projection, the direction of projection makes a 45° angle with the view plane. As a result, the projection of a line perpendicular to the view plane has the same length as the line itself; that is, there is no foreshortening. Fig. 7.12.4 shows cavalier projections of a unit cube with $\alpha = 45^\circ$ and $\alpha = 30^\circ$.
- When the direction of projection makes an angle of $\arctan(2) = 63.4^\circ$ with the view plane, the resulting view is called a **cabinet projection**. For this angle, lines perpendicular to the viewing surface are projected at one-half their actual length.

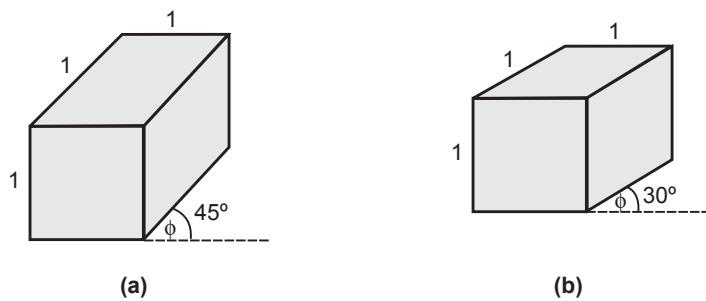


Fig. 7.12.4 Cavalier projections of the unit cube

- Cabinet projections appear more realistic than cavalier projections because of this reduction in the length of perpendiculars.
- Fig. 7.12.5 shows the examples of cabinet projections for a unit cube.

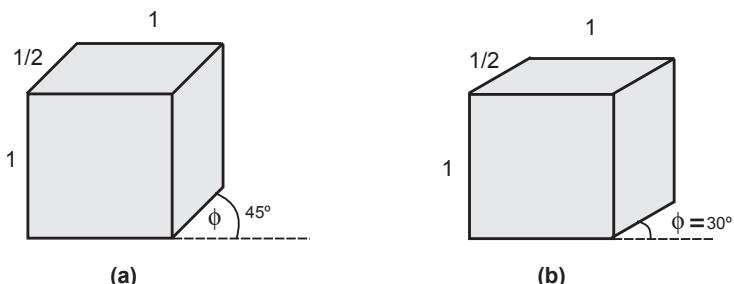


Fig. 7.12.5 Cabinet projections of the unit cube

Review Questions

- Compare parallel projections. SPPU : May-05, Marks 4
- Explain classification of parallel projection in detail. SPPU : May-06, Dec.-05, 06, 08, Marks 8
- Discuss applications of parallel projection.
- Explain parallel and perspective projection with example. SPPU : Dec.-15, Marks 4

7.13 Types of Perspective Projections SPPU : May-05,06,13,18, Dec.-05,06,08

- The perspective projection of any set of parallel lines that are not parallel to the projection plane converge to a **vanishing point**. The vanishing point for any set of lines that are parallel to one of the three principle axes of an object is referred to as a **principle vanishing point** or **axis vanishing point**. There are at most three such points, corresponding to the number of principle axes cut by the projection plane. The perspective projection is classified according to number of principle vanishing points in a projection : one-point, two-points or three-points projections. Fig. 7.13.1 shows the appearance of one-point, two-points and three point perspective projections of a cube.

7.13.1 One-Point Perspective Projection

- This projection is based on single vanishing point, hence it is also known as **single-point perspective**.
- Here, the object is placed so that one of its surfaces is parallel to the plane of projection. This is shown in the Fig. 7.13.1 (b).

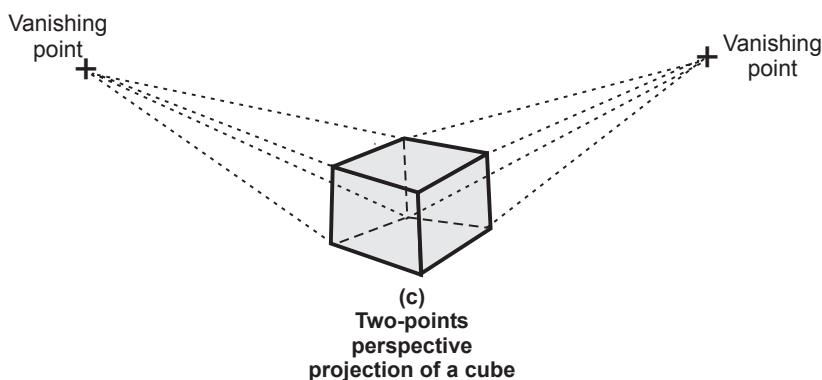
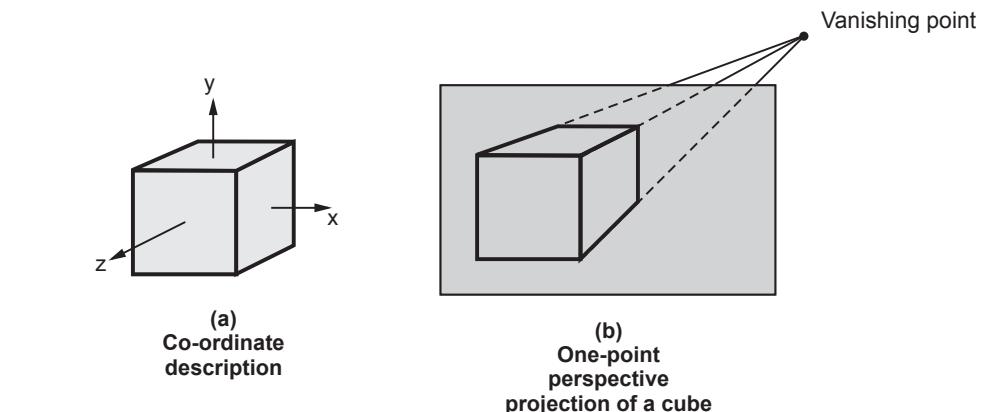
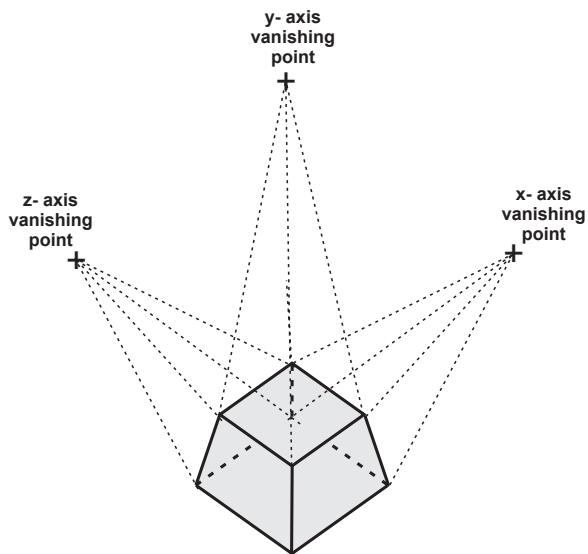


Fig. 7.13.1

- One point perspective projections are simple to produce and find many practical uses in engineering, architecture, and in computer graphics.
- One of the features of the one-point perspective is that if an object has cylindrical or circular forms and these are placed parallel to the plane of projection, then the forms are represented as circles or circular arcs in the perspective. This can be an advantage, considering that circles and circular arcs are easier to draw than ellipses or other conics.
- A special form of the one-point perspective projection takes place when the vanishing point is placed centrally within the figure. This type of projection is called a **tunnel perspective** or **tunnel projection**.
- Because of the particular positioning of the object in the coordinate axes, the depth cues in a tunnel projection are not very obvious. Fig. 7.13.2 shows the tunnel projection of a cube.
- It is typically used to view roads, rail tracks or buildings so that the front is directly facing the viewer.



(d) Three-point perspective of a cube
Fig. 7.13.1 Perspective projections

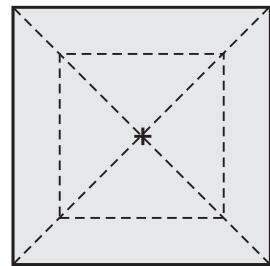


Fig. 7.13.2 Tunnel projection of a cube

7.13.2 Two-Point Perspective Projection

- In this projection, two surfaces of the object have vanishing points. In the case of a cube this is achieved if the object is rotated along its y-axis, so that lines along that axis remain parallel to the viewing plane, but those along the two other axes have vanishing points.
- Fig. 7.13.1 (c) shows a two-point perspective of a cube.
- Two-point perspective can be used to draw the same objects as one-point perspective, rotated : looking at the corner of a house, or looking at two forked road shrink into the distance, for example. One point represents one set of parallel lines, the other point represents the other. Looking at a house from the corner, one wall would recede towards one vanishing point, the other wall would recede towards the opposite vanishing point.

7.13.3 Three-Point Perspective Projection

- A three-point perspective is achieved by positioning the object so that none of its axes are parallel to the plane of projection. Although the visual depth cues in a three-point perspective are stronger than in the two-point perspective, the resulting geometrical deformations are sometimes disturbing to the viewer.
- Fig. 7.13.1 (d) is a three-point perspective projection of a cube.
- Three-point perspective is usually used for buildings seen from above (or below). In addition to the two vanishing points from before, one for each wall, there is now one for how those walls recede into the ground. This third vanishing point will be below the ground. Looking up at a tall building is another common example of the third vanishing point. This time the third vanishing point is high in space.

Review Questions

- Give examples one for each case of 3D objects having i) Never a vanishing point , ii) At most one vanishing point , iii) At most two vanishing point, iv) At most three vanishing points.

SPPU : May-13, Marks 8

- Give the classification of perspective projection. **SPPU : May-05, 06, Dec.-06, 08, Marks 4**
- What is the concept of vanishing point in perspective projection ? **SPPU : Dec.-05, Marks 8**
- Explain perspective projection and its types in brief. **SPPU : May-18, Marks 3**

7.14 Summary of Various Types of Projections

The Fig. 7.14.1 summarizes the logical relationship among the various types of projections.

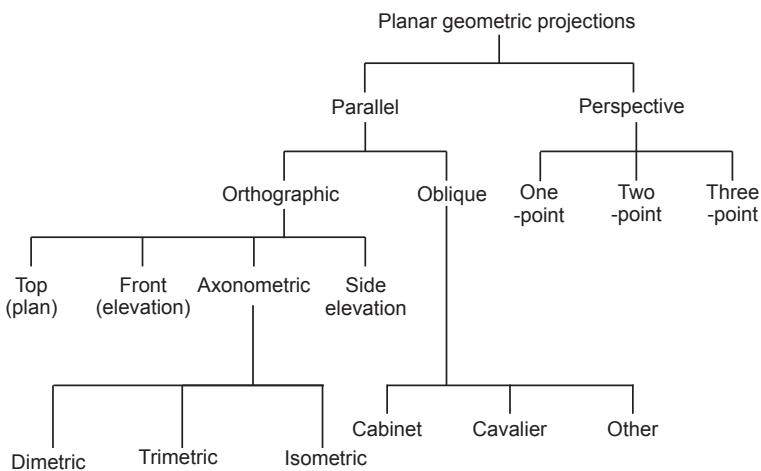
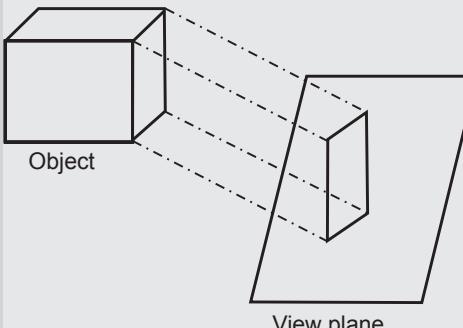
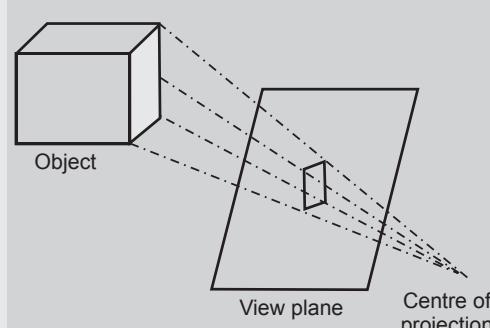


Fig. 7.14.1

Comparison between Parallel and Perspective Projections

Sr. No.	Parallel Projection	Perspective Projection
1.	 <p>Object</p> <p>View plane</p>	 <p>Object</p> <p>View plane</p> <p>Centre of projection</p>
2.	<p>It is a projection where co-ordinates of the given object are transformed by forming parallel lines to a view plane.</p>	<p>It is a projection where co-ordinates of the given object are transformed by forming converging lines also known as a center of projection to a view plane.</p>
3.	<p>Distance of the object from the center of projection is infinite.</p>	<p>Distance of the object from the center of projection is finite.</p>
4.	<p>It produces accurate view of various sides of the object to the view plane.</p>	<p>It does not produce accurate view of various sides of the object to the view plane; object size varies based on the distance from the view plane after transformation.</p>
5.	<p>It does not provide a realistic representation of an object.</p>	<p>It gives a realistic representation of an object.</p>
6.	<p>It preserves relative proportion of an object.</p>	<p>It does not preserve relative proportion of an object.</p>
7.	<p>Types :</p> <ol style="list-style-type: none"> 1. Orthographic projection 2. Oblique projection. 	<p>Types :</p> <ol style="list-style-type: none"> 1. One point perspective projection 2. Two point perspective projection 3. Three point perspective projection
8.	<p>Parallel projections are best suitable for architectural drawings, where measurements are necessary.</p>	<p>Perspective projections are suitable for creating realistic views.</p>

Review Questions

1. Give the classification of various types of projections.
2. Compare parallel and perspective projection.

Solved Examples

Example 7.1 A 3D cube dimensions (length, breadth, and height) 2 units each is placed in a 3D anti-clockwise axis system such that one of its vertex "A" is at the origin. (i.e. (0, 0, 0)) and vertex "F" in 3D space. Apply necessary transformations such that vertex F becomes the origin. Give complete mathematical formulation. Draw initial and final state of the cube.

SPPU : May-11, Marks 18

Solution : To bring vertex F to origin we need translation,

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

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 2 & 1 \\ 2 & 0 & 2 & 1 \\ 2 & 0 & 0 & 1 \\ 0 & 2 & 2 & 1 \\ 2 & 2 & 2 & 1 \\ 2 & 2 & 0 & 1 \\ 0 & 2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -2 & -2 & -2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \end{bmatrix} = \begin{bmatrix} -2 & -2 & -2 & 1 \\ -2 & -2 & 0 & 1 \\ 0 & -2 & 0 & 1 \\ 0 & -2 & -2 & 1 \\ -2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -2 & 1 \\ -2 & 0 & -2 & 1 \end{bmatrix}$$

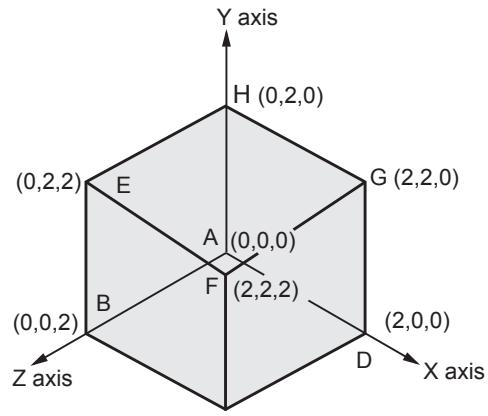


Fig. 7.1 Initial state of cube

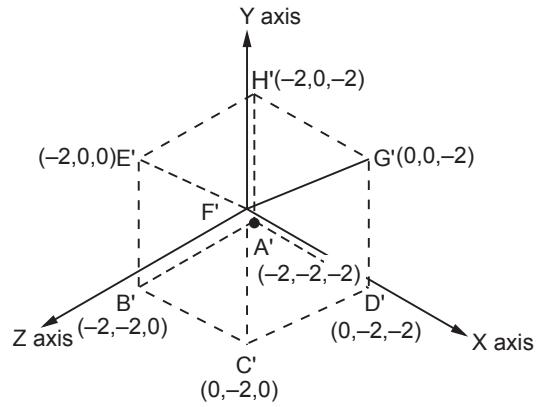


Fig. 7.2 Final state

Example 7.2 Obtain transformation matrix for rotation about the line joining the points (0, 0, 0) and (1, 1, 1) with the angle of rotation 45° in counter-clockwise sense.

Solution : In this case the line passes through the origin, so the translation is not required. Therefore, R_T can be given

$$R_T = R_{xy} \cdot R \cdot R_{xy}^{-1}$$

By usual notations,

$$\begin{aligned}\lambda &= \sqrt{1+1} \\ &= \sqrt{2} \\ |\mathbf{V}| &= \sqrt{1+1+1} \\ &= \sqrt{3}\end{aligned}$$

Here, $a = 1$, $b = 1$, $c = 1$. By using derived rotation matrices for R_{xy} , R and R_{xy}^{-1} we have

$$R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{\sqrt{6}}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{6}} & \frac{\sqrt{3}}{\sqrt{6}} & 0 \\ \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{6}} & 0 \\ \frac{\sqrt{6}}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{6}} & \frac{\sqrt{3}}{\sqrt{6}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{0}{\sqrt{2}} & \frac{0}{\sqrt{2}} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & -1 & -1 & 0 \\ \frac{\sqrt{3}}{\sqrt{6}} & \frac{\sqrt{6}}{\sqrt{6}} & \frac{\sqrt{6}}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}\therefore R_T &= R_{xy} \cdot R \cdot R_{xy}^{-1} \\ &= \begin{bmatrix} 0.80473 & 0.5058 & -0.3106 & 0 \\ -0.3106 & 0.80473 & 0.5058 & 0 \\ 0.5058 & -0.3106 & 0.80473 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

Example 7.3 A triangle is defined by 3 vertices $A(0, 2, 1)$, $B(2, 3, 0)$, $C(1, 2, 1)$. Find the final co-ordinates after it is rotated by 45° around a line joining the points $(1, 1, 1)$ and $(0, 0, 0)$.

Solution : The required transformation matrix for this example is already obtained in previous example.

$$\therefore \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot R_T$$

$$\begin{aligned}
 &= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.804 & 0.505 & -0.3106 & 0 \\ -0.3106 & 0.804 & 0.505 & 0 \\ 0.5058 & -0.3106 & 0.804 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} -0.116 & 1.297 & -1.814 & 0 \\ 0.676 & 3.422 & 0.893 & 0 \\ 0.687 & 1.802 & 0.998 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Therefore final co-ordinates are : A' (- 0.116, 1.297, - 1.814)

B' (0.676, 3.422, 0.893)

C' (0.687, 1.802, 0.998)

Example 7.4 A triangle is defined by 3 vertices A (0, 2, 1) B (2, 3, 0), C (1, 2, 1). Find the final co-ordinates after it is rotated by 45° around a line joining the points (2, 2, 2) and (1, 1, 1).

Solution : Here the given axis of rotation is not at the origin, therefore translation matrix is required. The translation matrix can be given as :

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

Therefore the inverse of translation matrix can be obtained as

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

It can be seen that after translating the given axis to the origin we get line points as (1, 1, 1) and (0, 0, 0) which are same as the line points considered in the previous example. The rotation angle (45°) in this example also matches with that in the previous example. Therefore we can use resultant matrix of the previous problem, in deriving the transformation matrix for the given problem. The transformation matrix for this problem can be given as

$$R_T = T \cdot R_{xy} \cdot R_z R_{xy}^{-1} \cdot T^{-1}$$

Substituting resultant matrix from the previous problem we have,

$$\begin{aligned}
 R_T &= \begin{bmatrix} 0.804 & 0.505 & -0.310 & 0 \\ -0.310 & 0.804 & 0.505 & 0 \\ 0.505 & -0.310 & 0.804 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\
 &\quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0.804 & 0.505 & -0.310 & 0 \\ -0.310 & 0.804 & 0.505 & 0 \\ 0.505 & -0.310 & 0.804 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \\
 &\quad \begin{bmatrix} 0.804 & 0.505 & -0.310 & 0 \\ -0.310 & 0.804 & 0.505 & 0 \\ 0.505 & -0.310 & 0.804 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 \begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} &= \begin{bmatrix} A \\ B \\ C \end{bmatrix} \cdot R_T \\
 &= \begin{bmatrix} 0 & 2 & 1 & 0 \\ 2 & 3 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.804 & 0.505 & -0.310 & 0 \\ -0.310 & 0.804 & 0.505 & 0 \\ 0.505 & -0.310 & 0.804 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned}$$

Calculation part is left for the student as an exercise.

Example 7.5 A cube defined by 8 vertices

$$A(0, 0, 0), \quad B(2, 0, 0), \quad C(2, 2, 0), \quad D(0, 2, 0)$$

$$E(0, 0, 2), \quad F(2, 0, 2), \quad G(2, 2, 2), \quad H(0, 2, 2)$$

Find the final co-ordinates after it is rotated by 45° around a line joining the points $(2, 0, 0)$ and $(0, 2, 2)$.

Solution : Let P $(2, 0, 0)$ and Q $(0, 2, 2)$

Step 1 : Shifting the arbitrary axis to origin by using translational matrix.

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

Step 2 : Finding inverse translation matrix

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

Step 3 : Finding matrix for coinciding the given axis with the z axis

Here, $a = 2$, $b = -2$, $c = -2$

by usual notations,

$$\lambda = \sqrt{b^2 + c^2}$$

$$|V| = \sqrt{a^2 + b^2 + c^2}$$

$$\therefore \lambda = \sqrt{4+4} = \sqrt{8} = 2\sqrt{2}$$

$$|V| = \sqrt{2^2 + (-2)^2 + (-2)^2} = \sqrt{12} = 2\sqrt{3}$$

By using derived rotation matrix for R_{xy}

$$R_{xy} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{6}} & -1 & -1 & 0 \\ \frac{1}{\sqrt{6}} & \frac{\sqrt{2}}{\sqrt{3}} & \frac{\sqrt{3}}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4 : Finding R_{xy}^{-1}

By using derived rotation matrix for R_{xy}^{-1}

$$R_{xy}^{-1} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{\sqrt{3}}{\sqrt{2}} & \frac{\sqrt{3}}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 5 : Matrix for rotation about z-axis. Here, $\theta = 45^\circ$

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

\therefore Resultant transformation matrix can be given as

$$R_T = T \cdot R_{xy} \cdot R \cdot R_{xy}^{-1} \cdot T^{-1}$$

From the resultant matrix the final co-ordinates of the cube can be obtained as

$$\begin{bmatrix} A' \\ B' \\ C' \\ D' \\ E' \\ F' \\ G' \\ H' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{bmatrix} \cdot R_T$$

Calculation part is left for the student as an exercise.



Notes

UNIT - IV

8

Colour Models

Syllabus

Properties of Light, CIE chromaticity Diagram, RGB, HSV, CMY.

Contents

8.1	<i>Introduction</i>	
8.2	<i>Properties of Light</i>	<i>May-18, Dec.-17, 19, Marks 3</i>
8.3	<i>CIE Chromaticity Diagram</i>	<i>May-09, 12, 13, 17, 18 Dec.-18 Marks 4</i>
8.4	<i>RGB</i>	<i>Dec.-12, 15, 16, May-16, 17, 19 Marks 8</i>
8.5	<i>HSV</i>	<i>May-05, 06, 16, 17, 18, 19, Dec.-10, Marks 4</i>
8.6	<i>CMY</i>	<i>May-06, 08, Dec.-10, 19 Marks 6</i>

8.1 Introduction

- A colour model is a specification of a 3D colour co-ordinate system and a visible subset in the co-ordinate system within which all colours in a particular colour range lie. For example, RGB colour model is the unit cube subset of the 3D cartesian co-ordinate system.
- The colour model allows to give convenient specification of colours in the specific colour range or gamut.
- There are three hardware oriented colour models : RGB, used for colour CRT monitors, YIQ used for the broadcast TV colour system and CMY (Cyan, Magenta, Yellow) used for some colour printing devices.
- However, these models are not easy to use because they does not relate directly to intuitive colour notions of hue, saturation and brightness. Therefore, another class of colour model has been developed. These include HSV, HLS and HVC models.

In this section, we are going to study RGB, HSV, CMY, CMYK colour models in detail.

8.2 Properties of Light

SPPU : May-18, Dec.-17, 19

- A light source produced by a sun or electric bulb emits all frequencies within the visible range to give white light.
- When this light is incident upon an object, some frequencies are absorbed and some are reflected by the object.
- The combination of reflected frequencies decides the colour of the object.
- If the lower frequencies are predominant in the reflected frequencies, the object colour is red. In this case, we can say that the perceived light has a dominant frequency at the red end of the spectrum.
- **Dominant frequency** : When white light is incident upon an object, some frequencies are reflected and some are absorbed by the object. The combination of frequencies present in the reflected light determines what we perceive as the colour of the object. If low frequencies are predominant in the reflected light, the object is described as red. In this case, we say the perceived light has a dominant frequency (or dominant wavelength) at the red end of the spectrum. The dominant frequency is also called the **hue**, or simply the colour, of the light.
- Apart from the frequency, there are two more properties which describe various characteristics of light. These are : brightness and saturation (purity).
- The **brightness** refers to the intensity of the perceived light.
- The **saturation** describes the purity of the colour.
- **Purity** : The purity is the perceived characteristic of the light. It describes how washed out or how "pure" the colour of the light appears.

- Pastels and pale colours are described as less pure or less saturated.
- When the two properties purity and dominant frequency are used collectively to describe the colour characteristics, are referred to as **chromaticity**.
- Two different colour light sources with suitably chosen intensities can be used to produce a range of other colour.
- When two colour sources are combined to produce white colour, they are referred to as **complementary colours**.
- Red and cyan, green and magenta, blue and yellow are complementary colour pairs.
- Usually, the colour model use combination of three colours to produce wide range of colours, called the **colour gamut** for that model.
- The basic colours used to produce colour gamut in particular model are called **primary colours**.

Review Questions

1. Discuss the properties of light.
2. What is colour gamut ?
3. Define dominant frequency.
4. Define purity.

SPPU : May-18, Dec.-19, Marks 3

SPPU : Dec.-17, Mark 1

8.3 CIE Chromaticity Diagram

SPPU : May-09, 12, 13, 17, 18, Dec.-18

- Matching and therefore defining a coloured light with a combination of three fixed primary colours is desirable approach to specify colour.
- In 1931, the Commission Internationale de l'Eclairage (CIE) defined three standard primaries, called X, Y and Z to replace red, green and blue.
- Here, X, Y and Z represent vectors in a three-dimensional, additive colour space.

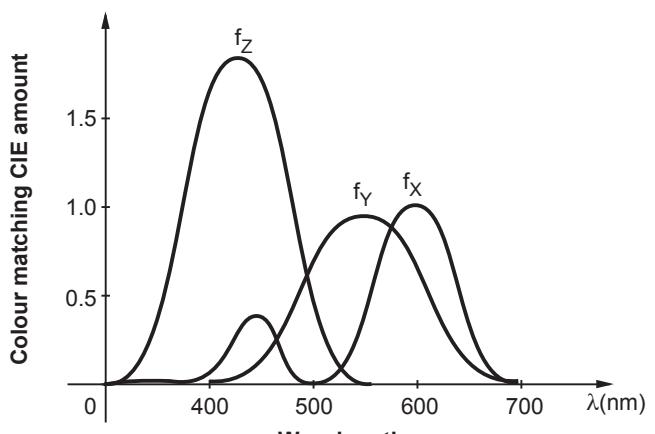


Fig. 8.3.1 Amounts of CIE primaries needed to display spectral colours

- The three standard primaries are imaginary colours. They are defined mathematically with positive colour-matching functions, as shown in Fig. 8.3.1. They specify the amount of each primary needed to describe any spectral colour.
- The advantage of using CIE primaries is that they eliminate matching of negative colour values and other problems associated with selecting a set of real primaries.
- Any colour (C_λ) using CIE primaries can be expressed as,

$$C_\lambda = X \mathbf{X} + Y \mathbf{Y} + Z \mathbf{Z}$$

where X , Y and Z are the amounts of the standard primaries needed to match C_λ and \mathbf{X} , \mathbf{Y} and \mathbf{Z} represent vectors in a three-dimensional, additive colour space.

- With above expression we can define chromaticity values by normalizing against luminance ($X + Y + Z$). The normalizing amounts can be given as,

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z}$$

- Notice that $x + y + z = 1$. That is, x , y and z are on the $(X + Y + z = 1)$ plane. The complete description of colour is typically given with the three values x , y and z . The remaining values can be calculated as follows :

$$z = 1 - x - y, \quad X = \frac{x}{y} Y, \quad Z = \frac{z}{y} Y$$

- Chromaticity values depend only on dominant wavelength and saturation and are independent of the amount of luminous energy.
- By plotting x and y for all visible colours, we obtain the CIE chromaticity diagram shown in Fig. 8.3.2, which is the projection onto the (X, Y) plane of the $(X + Y + Z = 1)$ plane.
- The interior and boundary of the tongue-shaped region represent all visible chromaticity values.
- The points on the boundary are the **pure** colours in the electromagnetic spectrum, labelled according to wavelength in nanometre from the red end to the violet end of the spectrum.
- A standard white light, is formally defined by a light source illuminant C , marked by the center dot.

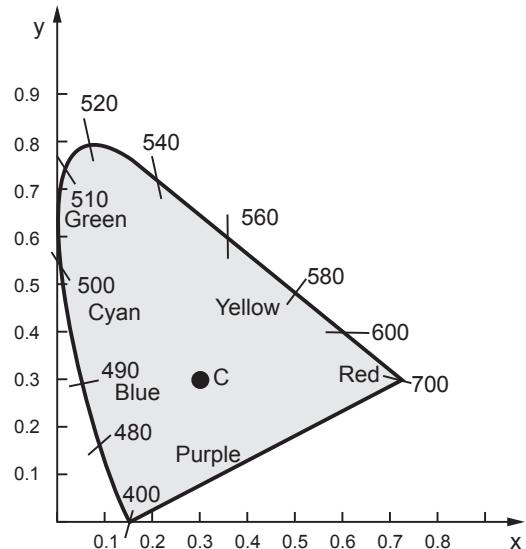


Fig. 8.3.2

- The line joining the red and violet spectral points is called the purple line, which is not the part of the spectrum.
- The CIE chromaticity diagram is useful in many ways :
 - It allows us to measure the dominant wavelength and the purity of any colour by matching the colour with a mixture of the three CIE primaries.
 - It identifies the complementary colours.
 - It allows to define colour gamuts or colour ranges, that show the effect of adding colours together.
- The Fig. 8.3.3 represents the complementary colours on the chromaticity diagram.
- The straight line joining colours represented by points D and E passes through point C (represents white light).
- This means that when we mix proper amounts of the two colours D and E in Fig. 8.3.3, we can obtain white light. Therefore, colours D and E are complementary colours and with point C on the chromaticity diagram we can identify the complement colour of the known colour.
- Colour gamuts are represented on the chromaticity diagram as a straight line or as a polygon.
- Any two colours say A and B can be added to produce any colour along their connecting line by mixing their appropriate amounts.
- The colour gamut for three points, such as D, E and F in Fig. 8.3.4, is a triangle with three colour points as vertices.
- The triangle DEF in Fig. 8.3.4, shows that three primaries can only generate colours inside or on the bounding edges of the triangle.
- The chromaticity diagram is also useful to determine the dominant wavelength of a colour.

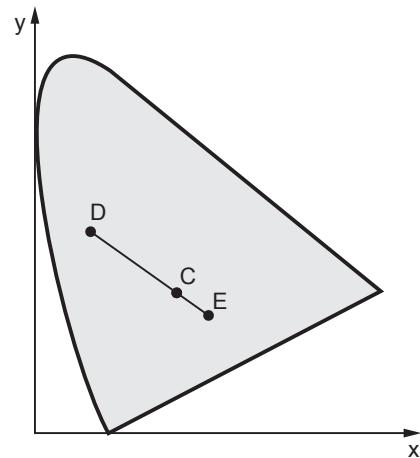


Fig. 8.3.3 Complementary colours on chromaticity diagram

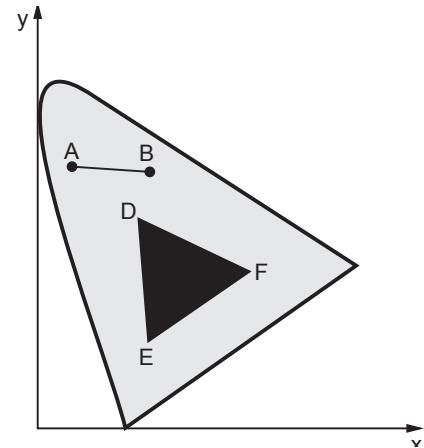


Fig. 8.3.4 Definition of colour gamuts on the chromaticity diagram

- For colour point D in the Fig. 8.3.5, we can draw a straight line from C through D to intersect the spectral curve at point E.
- The colour D can then be represented as a combination of white light C and the spectral colour E. Thus, the dominant wavelength of D is E.
- This method for determining dominant wavelength will not work for colour points that are between C and the purple line because the purple line is not a part of spectrum.

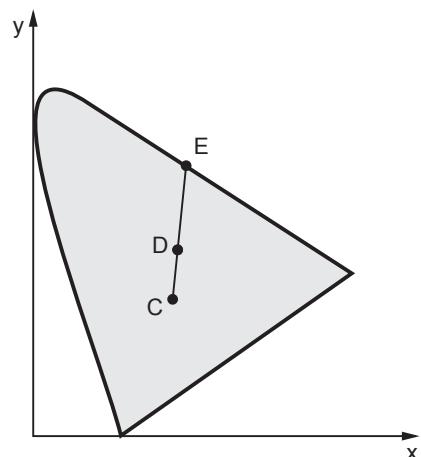


Fig. 8.3.5 Determination of dominant wavelength on the chromaticity diagram

Review Questions

- Define chromaticity, complementary colours, colour gamut and primary colours.
- Explain the usefulness of CIE chromaticity diagram with examples.
- Explain CIE chromaticity diagram.

SPPU : May-09,12,13,17,18,Dec.-18, Marks 4

8.4 RGB

SPPU : Dec.-12, 15,16, May-16,17,19

- The red, green and blue (RGB) colour model used in colour CRT monitors and colour raster graphics employ a cartesian co-ordinate system.
- In this model, the individual contribution of red, green and blue are added together to get the resultant colour.
- We can represent this colour model with the unit cube defined on R, G and B axes, as shown in the Fig. 8.4.1.
- The vertex of the cube on the axes represent the primary colours and the remaining vertices represent the complementary colour for each of the primary colours.
- The main diagonal of the cube, with equal amounts of each primary represents the gray B

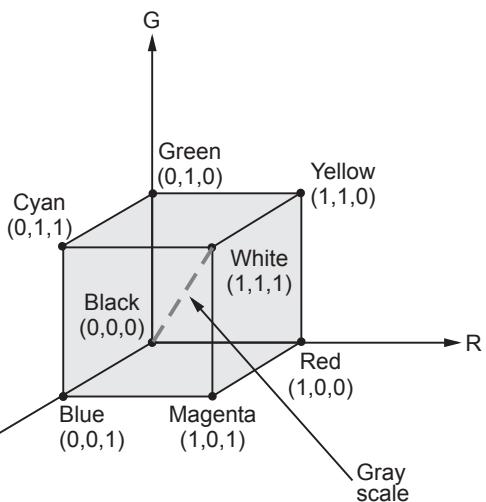


Fig. 8.4.1 The RGB cube

levels. The end at the origin of the diagonal represents black (0, 0, 0) and other end represents white (1, 1, 1).

- Each colour point within the bounds of the cube is represented as the triple (R, G, B), where value for R, G, B are assigned in the range from 0 to 1.
- As mentioned earlier, it is an additive model.
- Intensities of the primary colours are added to get the resultant colour. Thus, the resultant colour C_λ is expressed in RGB component as,

$$C_\lambda = RR + GG + BB$$

- The RGB chromaticity co-ordinates for the CIE RGB colour model as given as R (0.735, 0.265), G (0.274, 0.717), B (0.167, 0.009). The Fig. 8.4.2 shows the colour gamut for the CIE standard RGB primaries.

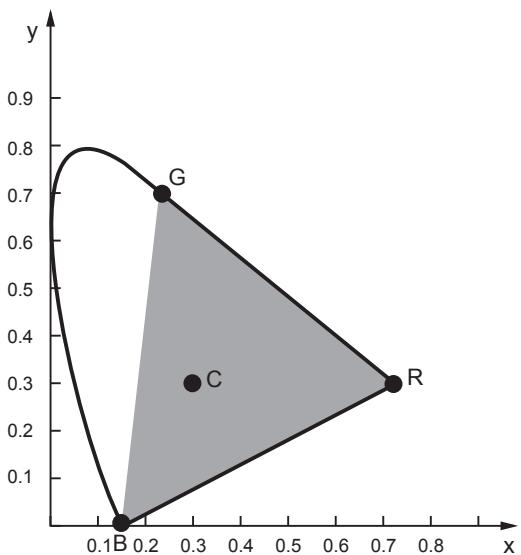


Fig. 8.4.2 Colour gamut for CIE standard RGB primaries

Review Question

1. Explain RGB colour model.

SPPU : Dec.-12,15,16, May-16,17,19, Marks 8

8.5 HSV

SPPU : May-05,06,16,17,18,19, Dec.-10

- Another model based on intuitive colour parameters is the HLS colour model used by Tektronix.
- The three colour parameters in this model are : hue (H), lightness (L) and saturation (S). It is represented by double hexcone, as shown in the Fig. 8.5.1.
- The hue specifies the angle around the vertical axis of the double hexcone.
- In this model, H = 0° corresponds to blue.
- The remaining colours are specified around the perimeter of the hexcone in the same order as in the HSV model.
- Magenta is at 60°, red is at 120° and yellow is located at H = 180°.
- Complementary colours are 180° apart on the double hexcone.
- The vertical axis in this model represents the lightness, L.

- At $L = 0$, we have black and at $L = 1$, we have white.
- In between value of L we have gray levels.
- The saturation parameters S varies from 0 to 1 and it specifies relative purity of a colour.
- At $S = 0$, we have the gray scale and at $S = 1$ and $L = 0.5$, we have maximum saturated (pure) hue.
- As S decrease, the hue saturation decreases i.e. hue becomes less pure.
- In HLS model a hue can be selected by selecting hue angle H and the desired shade, tint or tone can be obtained by adjusting L and S .
- The colours can be made lighter by increasing L and can be made darker by decreasing L .
- The colours can be moved towards gray by decreasing S .

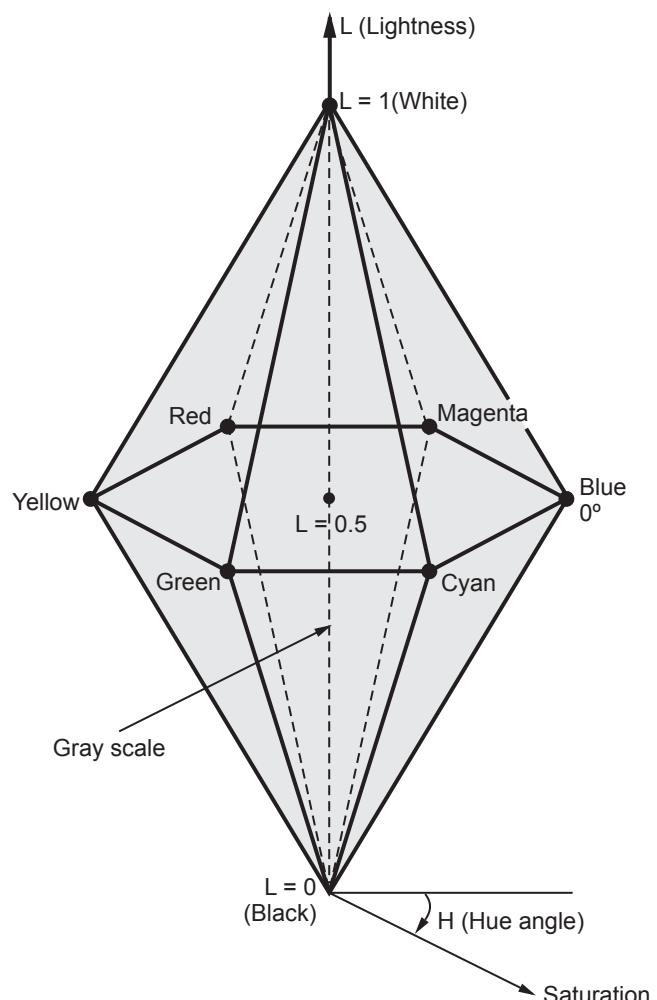


Fig. 8.5.1 Double-hexcone HLS colour model

Review Question

1. Explain HSV colour model.

SPPU : May-05, 06, 17, 18, 19, Dec.-10, Marks 4

8.6 CMY

SPPU : May-06, 08, Dec.-10, 19

- In this model cyan, magenta and yellow colours are used as a primary colours. This model is used for describing colour output to hard-copy devices.
- Unlike video monitor, which produce a colour pattern by combining light from the screen phosphors, hard-copy devices such as plotters produce a colour picture by coating a paper with colour pigments.

- The subset of the cartesian co-ordinate system for the CMY model is the same as that for RGB except that white (full light) instead of black (no light) is at the origin. Colours are specified by what is removed or subtracted from white light, rather than by what is added to blackness.
- Cyan can be formed by adding green and blue light. Therefore, when white light is reflected from cyan coloured ink, the reflected light does not have red component. That is, red light is absorbed or subtracted, by the ink.
- Magenta ink subtracts the green component from incident light and yellow subtracts the blue component. Therefore, cyan, magenta and yellow are said to be complements of red, green and blue respectively.
- Fig. 8.6.1 shows the cube representation for CMY model.
- As shown in the Fig. 8.6.1, point (1,1,1) represents black, because all components of the incident light are subtracted.
- The point (0, 0, 0), the origin represents white light.
- The main diagonal represents equal amount of primary colours, thus the gray colours.
- A combination of cyan and yellow produces green light, because the red and blue components of the incident light are absorbed.
- Other colour combinations are obtained by a similar subtractive process.
- It is possible to get CMY representation from RGB representation as follows,

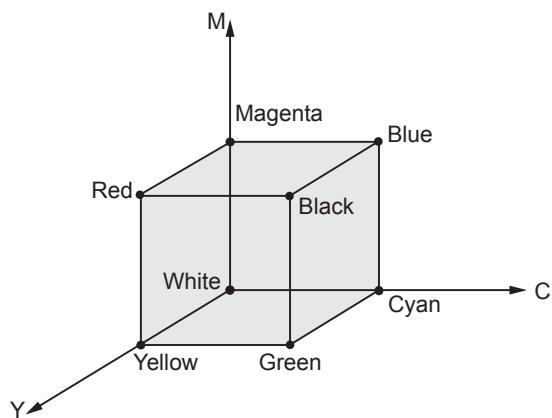


Fig. 8.6.1 The CMY cube

- The unit column vector is the RGB representation for white and the CMY representation for black. The conversion from RGB to CMY is then can be given as,

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

Review Questions

1. Explain CMY colour model.
2. Compare RGB and CMY colour model.

SPPU : May-06, Dec.-10,19, Marks 4**SPPU : May-08, Marks 6**

UNIT - IV

9

Illumination Models and Shading Algorithms

Syllabus

Illumination Models : Ambient Light, Diffuse reflection, Specular Reflection, and the Phong model, Combined diffuse and Specular reflections with multiple light sources, warn model,

Shading Algorithms : Halftone, Gouraud and Phong Shading.

Contents

9.1	<i>Introduction</i>	
9.2	<i>Light Sources</i>	May-19, Dec.-19, Marks 2
9.3	<i>Ambient Light</i>	May-19, Dec.-19, Mark 1
9.4	<i>Diffuse Illumination and Reflection</i>	Dec.-07,10,14,15,16,17,18, May-11,12,13,16,19, Marks 8
9.5	<i>Specular Reflection</i>	May-13, Marks 3
9.6	<i>Phong Model</i>	Dec.-05, 08,17,18, May-17,18, Marks 4
9.7	<i>Combined Diffuse and Specular Reflections with Multiple Light Sources</i>	
9.8	<i>Warn Model</i>	May-17, Marks 4
9.9	<i>Shading Algorithms</i>	Dec.-06,08,09,11,12,14,17,18,19, May-07,08,10,11,12,13,14,15, 17,18,19, Marks 8

9.1 Introduction

As a simple definition, rendering is the process of producing realistic images or pictures. Producing realistic images involves both physics and psychology. Light, i.e., electromagnetic energy, reaches the eye after interacting with the physical environment. The human brain interprets this information for visualization of the object. Therefore, to produce realistic images we should know which surface of the object appears brighter or darker and which surface appears lighter. Many times we have to show the shadows of the objects appear in the picture.

9.2 Light Sources

SPPU : May-19, Dec.-19

We see reflected light from the surfaces of the object. The total reflected light is the sum of the contributions from light sources and other reflecting surfaces in the scene. This is illustrated in Fig. 9.2.1. Thus, a surface that is not directly exposed to a light source may still be visible if nearby objects are illuminated. There are light-emitting sources such as electric bulb and light reflecting surfaces such as the walls of a room.

The term light source is used for an object that is emitting radiant energy, such as a Light bulb or the sun. A luminous object, in general, can be both a light source and a light reflector. For example, a plastic globe with a light bulb inside both emits and reflects light from the surface of the globe. Emitted light from the globe may then illuminate other objects in the vicinity.

Point Source Illumination Model

It is a simplest model for a light emitter. Here, rays from the source follow radially diverging paths from the source position. This is illustrated in Fig. 9.2.2. This light-source model is a reasonable approximation for sources whose dimensions are small compared to the size of objects in the scene or for sources, such as the sun, that are sufficiently far from the scene.

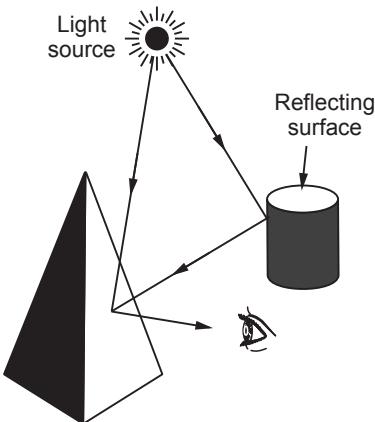


Fig. 9.2.1 Light from light source and reflecting surface

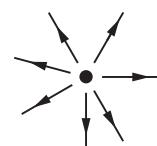


Fig. 9.2.2 Ray paths from a point light source

Distributed Light Source

When the area of the source is not small compared to the surfaces in the scene, the illumination effects cannot be approximated realistically with a point source. In such cases, light source such as the long fluorescent light is more accurately modeled as a distributed light source. This is illustrated in Fig. 9.2.3.

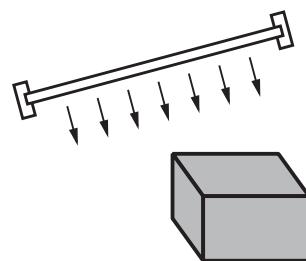


Fig. 9.2.3 Distributed light source

Diffuse Reflection

We know that shiny materials reflect more of the incident light, and dull surfaces absorb more of the incident light. Whereas rough or grainy surfaces tend to scatter the reflected light in all directions. This scattered light is called **diffuse reflection**. This is illustrated in Fig. 9.2.4.

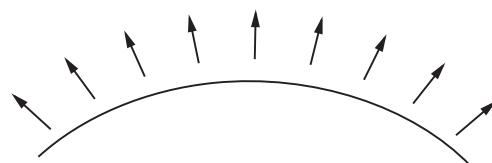


Fig. 9.2.4 Diffuse reflection

Specular Reflection

Light sources create highlights, or bright spots, called **specular reflection**. This highlighting effect is more pronounced on shiny surfaces than on dull surfaces. This is illustrated in Fig. 9.2.5.

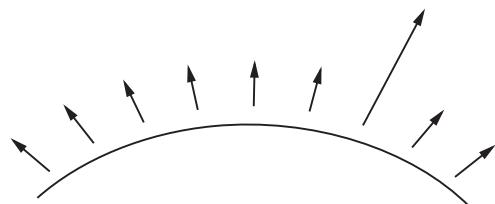


Fig. 9.2.5 Specular reflection superimposed on diffuse reflection

Review Question

1. Explain various light sources.
2. Explain diffuse reflection and specular reflection.

SPPU : May-19, Dec.-19, Mark 1

SPPU : May-19, Dec.-19, Marks 2

9.3 Ambient Light

SPPU : May-19, Dec.-19

A surface of the object that is not exposed directly to a light source still will be visible if nearby objects are illuminated. A simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the **ambient light**, or **background light** is to set a general level of brightness for a scene. This level is defined by the parameter I_a and each surface is then illuminated with this constant value. With such illumination, the reflected light is a constant for each surface, independent of the viewing direction and the spatial orientation of the surface. However, the intensity of the reflected light for each surface, depends on the optical properties of the surface, that is; how much of the incident light energy is to be reflected and how much absorbed.

Review Question

1. What do you mean by ambient light ?

SPPU : May-19, Dec.-19, Mark 1

9.4 Diffuse Illumination and Reflection

SPPU : Dec.-07,10,14,15,16,17,18, May-11,12,13,16,19

- An object illumination is as important as its surface properties in computing its intensity. The object may be illuminated by light which does not come from any particular source but which comes from all directions. When such illumination is uniform from all directions, the illumination is called **diffuse illumination**.
- Usually, diffuse illumination is a background light which is reflected from walls, floor and ceiling.
- When we assume that going up, down, right and left is of same amount then we can say that the reflections are constant over each surface of the object and they are independent of the viewing direction. Such a reflection is called **diffuse reflection**.
- In practice, when object is illuminated, some part of light energy is absorbed by the surface of the object, while the rest is reflected.
- The ratio of the light reflected from the surface to the total incoming light to the surface is called **coefficient of reflection** or the **reflectivity**. It is denoted by R.
- The value of R varies from 0 to 1. It is closer to 1 for white surface and closer to 0 for black surface. This is because white surface reflects nearly all incident light whereas black surface absorbs most of the incident light. Reflection coefficient for gray shades is in between 0 to 1.
- In case of colour object, reflection coefficient are various for different colour surfaces.

Lambert's Law

- We have seen that, the diffuse reflections from the surface are scattered with equal intensity in all directions, independent of the viewing direction. Such surfaces are sometimes referred to as **ideal diffuse reflectors**. They are also called **Lambertian reflector**, since radiated light energy from any point on the surface is governed by **Lambert's cosine law**.

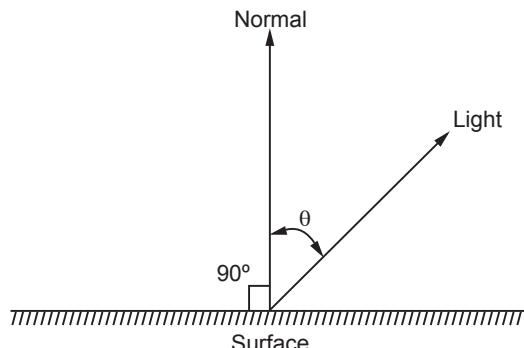


Fig. 9.4.1 The direction of light is measured from the surface normal

- This law states that the reflection of light from a perfectly diffusing surface varies as the cosine of the angle between the normal to the surface and the direction of the reflected ray. This is illustrated in Fig. 9.4.1.
- Thus if the incident light from the source is perpendicular to the surface at a perpendicular point, that point is fully illuminated.
- On the other hand, as the angle of illumination moves away from the surface normal, the brightness of the point drops off; but the points appear to be squeezed closer together and the net effect is that the brightness of the surface is unchanged. This is illustrated in Fig. 9.4.2.
- In other words, we can say that the reduction in brightness due to cosine of angle gets cancelled by increase in the number of light-emitting points within the area of view.
- A similar cancellation effect can be observed as the surface is moved farther from the view point. As we move farther from the view port, the light coming from the surface spreads over a large area. This area increases by the square of distance, thus the amount of light reaching the eye decreases by the same factor.
- This factor is compensated by the size of the object. When object is moved farther from the viewport, it appears smaller. Therefore, even though there is less light, it is applied to a smaller area on the retina and hence the brightness of the surface remains unchanged.
- The expression for the brightness of an object illuminated by diffuse ambient or background light can be given as,

$$I_{\text{ambdiff}} = k_a I_a$$

where I_a is the intensity of the ambient light or background light, k_a is the ambient reflection coefficient and I_{ambdiff} is the intensity of diffuse reflection at any point on the surface which is exposed only to ambient light.

- Using above equation it is possible, to create light or dark scenes or gray shaded objects. But in this simple model, every plane on a particular object will be shaded equally. The real shaded object does not look like this. For more realistic shading model we also have to consider the point sources of illumination.

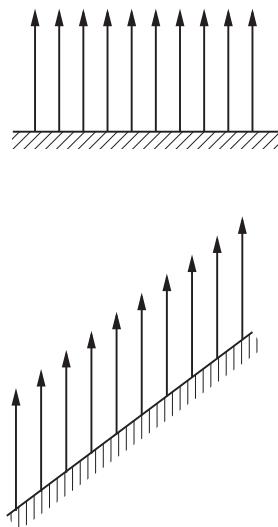


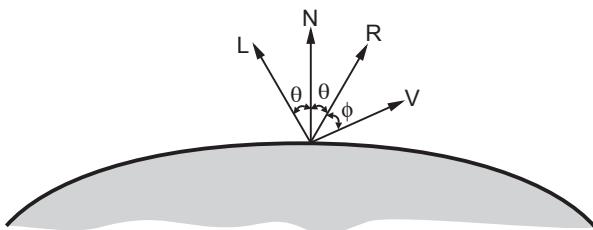
Fig. 9.4.2 Surface brightness

Review Questions

1. *Describe diffuse illumination.* **SPPU : Dec.-07, 10, 15, May-12, 13, 16, Marks 4**
2. *Define : diffuse illumination, diffuse reflection and coefficient of reflection.*
3. *Explain the Lambert's cosine law ? What is its significance ?* **SPPU : May-19, Marks 3**
4. *What is diffused reflection ? Give the illumination model that incorporate this reflection.*
5. *Write short note on diffuse illumination.* **SPPU : May-11, Marks 8**
6. *Explain reflectivity.* **SPPU : Dec.-14, Marks 2**
7. *Explain and compare point source illumination and diffused illumination.*

SPPU : Dec.-16,17,18, Marks 3**9.5 Specular Reflection****SPPU : May-13**

- When we illuminate a shiny surface such as polished metal or an apple with a bright light, we observe highlight or bright spot on the shiny surface. This phenomenon of reflection of incident light in a concentrated region around the specular-reflection angle is called **specular-reflection**.
- Due to specular-reflection, at the highlight, the surface appears to be not in its original colour, but white, the colour of incident light.
- The Fig. 9.5.1 shows the specular-reflection direction at a point on the illuminated surface. The specular-reflection angle equals the angle of the incident light, with the two angles measured on opposite sides of the unit normal surface vector N.
- As shown in the Fig. 9.5.1, R is the unit vector in the direction of ideal specular-reflection, L is the unit vector directed towards the point light source and V is the unit vector pointing to the viewer from the surface position.
- The angle ϕ between vector R and vector V is called **viewing angle**. For an ideal reflector (perfect mirror), incident light is reflected only in the specular-reflection direction. In such case, we can see reflected light only when vector V and R coincide, i.e., $\phi = 0$.

**Fig. 9.5.1 Specular-reflection****Review Question**

1. *Explain the following in detail : Specular reflection.*

SPPU : May-13, Marks 3

9.6 Phong Model

SPPU : Dec.-05, 08,17,18, May-17,18

- Phong Bui-Tuong developed a popular illumination model for nonperfect reflectors.
- It assumes that maximum specular-reflection occurs when ϕ is zero and falls off sharply as ϕ increases.
- This rapid fall-off is approximated by $\cos^n \phi$, where n is the specular-reflection parameter determined by the type of surface. The values of n typically vary from 1 to several hundred, depending on the surface material. The larger values (say, 100 or more) of n are used for very shiny surface and smaller values are used for dull surfaces. For a perfect reflector, n is infinite. For rough surface, such as chalk, n would be near to 1.
- Fig. 9.6.1 and Fig. 9.6.2 show the effect of n on the angular range of specular-reflection.

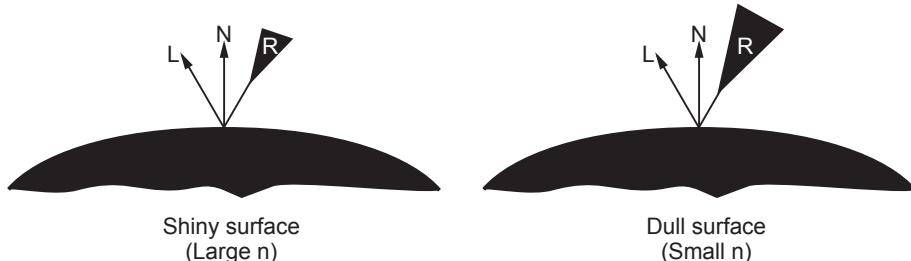


Fig. 9.6.1 Effect of n on the angular range of specular-reflection

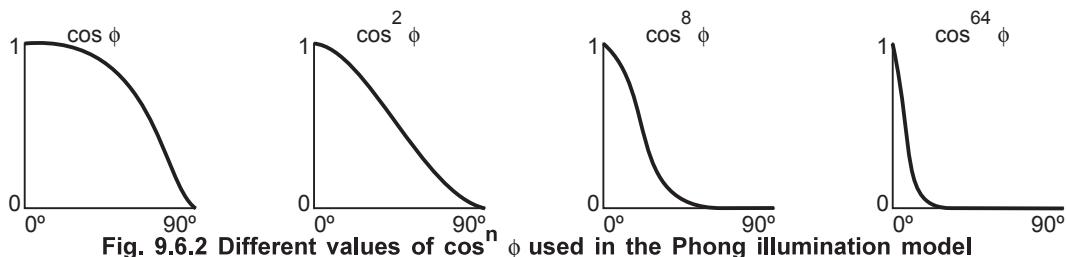


Fig. 9.6.2 Different values of $\cos^n \phi$ used in the Phong illumination model

- The amount of incident light specularly reflected depends on the angle of incidence θ , material properties of surface, polarization and colour of the incident light. The model is approximated for monochromatic specular intensity variations using a specular-reflection coefficient, $W(\theta)$, for each surface. We can write the equation for Phong specular-reflection model as,

$$I_{\text{spec}} = W(\theta) I_l \cos^n \phi$$

where I_l is the intensity of the light source and ϕ is the angle between viewing vector and specular reflection vector R .

- $W(\theta)$ is typically set to a constant k_s , the material's specular-reflection coefficient, which ranges from between 0 to 1. The value of k_s is selected experimentally to produce aesthetically pleasing results. Note that V and R are the unit vectors in the viewing and specular-reflection directions, respectively. Therefore, we can calculate the value of $\cos \phi$ with the dot product $V \cdot R$. Considering above changes we can rewrite the equation for intensity of the specular-reflection as,

$$I_{\text{spec}} = k_s I_l (V \cdot R)^n$$

- The vector R in the above equation can be calculated in terms of vector L and N . This calculation requires mirroring L about N . As shown in Fig. 9.6.3, this can be accomplished with some simple geometry. Since N and L are normalized, the projection of L onto N is $N \cos \theta$. Note that $R = N \cos \theta + S$, where $|S|$ is $\sin \theta$. But, by vector subtraction and congruent triangles, S is just $N \cos \theta - L$. Therefore,

$$\begin{aligned} R &= N \cos \theta + N \cos \theta - L \\ &= 2 N \cos \theta - L \end{aligned}$$

- Substituting $N \cdot L$ for $\cos \theta$ we have,

$$R = 2N(N \cdot L) - L$$

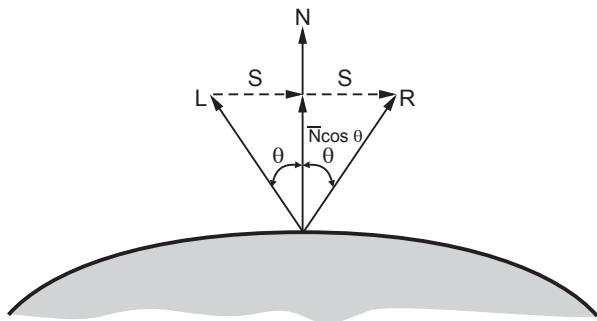


Fig. 9.6.3 Calculating the reflection vector

Review Question

1. Explain the Phong model.

SPPU : Dec.-05, 08,17,18, May-17,18, Marks 4

9.7 Combined Diffuse and Specular Reflections with Multiple Light Sources

- For a single point light source, the combined diffuse and specular reflections from any point on the illuminated surface is given as,

$$\begin{aligned} I &= I_{\text{diff}} + I_{\text{spec}} \\ &= k_a I_a + k_d I_l (N \cdot L) + k_s I_l (N \cdot H)^n \end{aligned}$$

- For a multiple point light source the above equation can be modified as,

$$I = k_a I_a + \sum_{i=1}^M I_{l_i} [k_d (N \cdot L_i) + k_s (N \cdot H_i)^n]$$

- Therefore, in case of multiple point light sources the light reflected at any surface point is given by summing the contributions from the individual sources.

Review Question

1. Derive the illumination model with combine diffuse and specular reflections.

9.8 Warn Model

SPPU : May-17

So far, we have modelled light sources as points radiating uniformly in all directions. The Warn model improves this by letting a light source be aimed in a certain direction. The intensity of light then varies with the direction from the source. The Warn model provides a method for simulating studio lighting effects by controlling light intensity in different directions.

In the Warn model, this directionality is given by $\cos^n \beta$ where β is the angle from the central (most intense) direction. The larger the value of n , the more concentrated the light is in a certain direction. Here, the light sources are modeled as points on a reflecting surface, using the Phong model for the surface point. Then the intensity in different directions is controlled by selecting values for the Phong exponent.

Warn model can be used to simulate barn doors and spot lighting.

Review Question

1. Write a note on Warn model.

SPPU : May-17, Marks 4

9.9 Shading Algorithms

SPPU : Dec.-06,08,09,11,12,14,17,18,19, May-07,08,10,11,12,13,14,15,17,18,19

- From the previous discussion, it is clear that we can shade any surface by calculating the surface normal at each visible point and applying the desired illumination model at that point.
- Unfortunately, this shading method is expensive. In this section, we discuss more efficient shading methods for surfaces defined by polygons.

9.9.1 Constant-Intensity Shading

- The fast and simplest method for shading polygon is constant shading, also known as **faceted shading** or **flat shading**.
- In this method, illumination model is applied only once for each polygon to determine single intensity value. The entire polygon is then displayed with the single intensity value.
- This method is valid for the following assumptions :

1. The light source is at infinity, so $N \cdot L$ is constant across the polygon face.
 2. The viewer is at infinity, so $V \cdot R$ is constant over the surface.
 3. The polygon represents the actual surface being modeled and is not an approximation to a curved surface.
- If either of the first two assumptions are not true still we can use constant intensity shading approach; however, we require some method to determine a single value for each of L and V vectors.

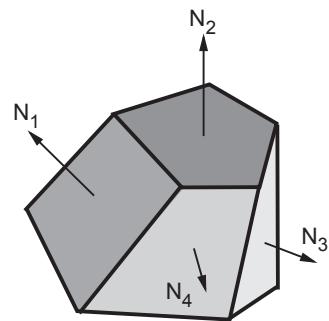


Fig. 9.9.1 Polygons and their surface normals

9.9.2 Gouraud Shading

- In this method, the intensity interpolation technique developed by Gouraud is used, hence the name.
- The polygon surface is displayed by linearly interpolating intensity values across the surface. Here, intensity values for each polygon are matched with the values of adjacent polygons along the common edges. This eliminates the intensity discontinuities that can occur in flat shading.
- By performing following calculations we can display polygon surface with Gouraud shading.
 1. Determine the average unit normal vector at each polygon vertex.
 2. Apply an illumination model to each polygon vertex to determine the vertex intensity.
 3. Linearly interpolate the vertex intensities over the surface of the polygon.
- We can obtain a normal vector at each polygon vertex by averaging the surface normals of all polygons sharing that vertex. This is illustrated in Fig. 9.9.2.
- As shown in the Fig. 9.9.2, there are three surface normals N_1 , N_2 and N_3 of polygon sharing vertex V . Therefore, normal vector at vertex V is given as

$$N_V = \frac{N_1 + N_2 + N_3}{|N_1 + N_2 + N_3|}$$

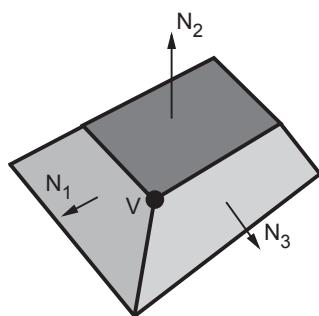


Fig. 9.9.2 Calculation of normal vector at polygon vertex V

- In general, for any vertex position V , we can obtain the unit vertex normal by equation,

$$N_V = \frac{\sum_{i=1}^n N_i}{\sqrt{\sum_{i=1}^n N_i}}$$

where n is the number of surface normals of polygons sharing that vertex.

- The next step in Gouraud shading is to find vertex intensities. Once we have the vertex normals, their vertex intensities can be determined by applying illumination model to each polygon vertex.
- Finally, each polygon is shaded by linear interpolating of vertex intensities along each edge and then between edges along each scan line. This is illustrated in Fig. 9.9.3.

- For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints. For example, in Fig. 9.9.3, the polygon edge with endpoint vertices 1 and 2 is intersected by the scan line at point 'a'. The intensity at point 'a' can be interpolated from intensities I_1 and I_2 as,

$$I_a = \frac{y_a - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_a}{y_1 - y_2} I_2$$

- Similarly, we can interpolate the intensity value for right intersection (point b) from intensity values I_2 and I_3 as,

$$I_b = \frac{y_a - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_a}{y_3 - y_2} I_2$$

- Once the intensities of intersection points a and b are calculated for a scan line, the intensity of an interior point (such as P in Fig. 9.9.3) can be determined as,

$$I_p = \frac{x_b - x_p}{x_b - x_a} I_a + \frac{x_p - x_a}{x_b - x_a} I_b$$

- During the scan conversion process, usually incremental calculations are used to obtain the successive edge intensity values between the scan lines and to obtain successive intensity along a scan line. This eliminates the repetitive calculations.
- If the intensity at edge position (x, y) is interpolated as,

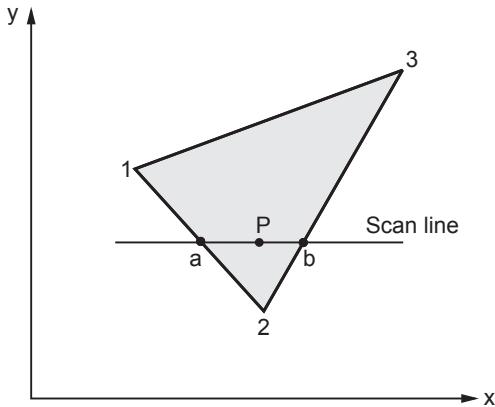


Fig. 9.9.3

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

then we can obtain the intensity along this edge for the next scan line, $y - 1$ as (See Fig. 9.9.4),

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

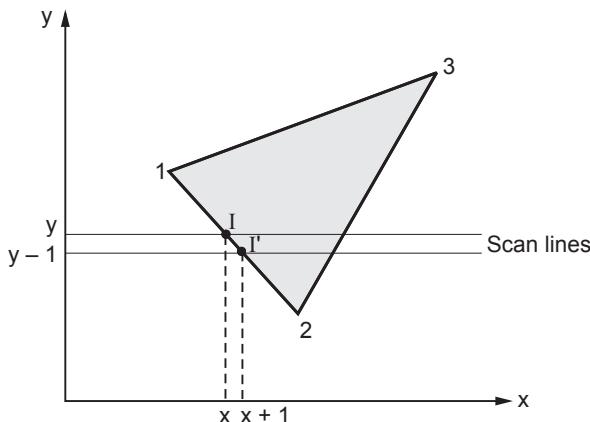


Fig. 9.9.4 Calculation of incremental interpolation of intensity values along a polygon edge for successive scan lines

- Similarly, we can obtain intensities at successive horizontal pixel positions along each scan line (See Fig. 9.9.5) as,

$$I' = I + \frac{I_b - I_a}{x_b - x_a}$$

Advantages

- It removes the intensity discontinuities exists in constant shading model.
- It can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.

Disadvantages

- Highlights on the surface are sometimes displayed with anomalous shapes.
- The linear intensity interpolation can result bright or dark intensity streaks to appear on the surface. These bright or dark intensity streaks, are called **Mach**

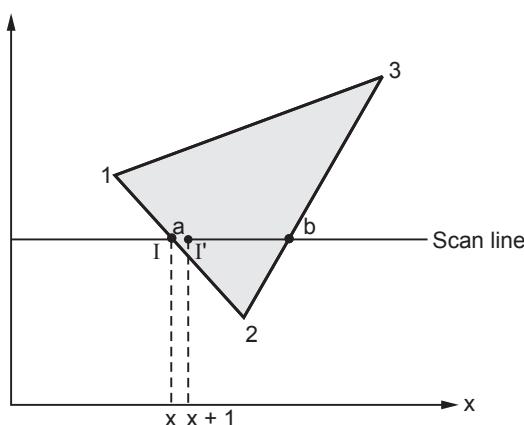


Fig. 9.9.5 Calculation of incremental interpolation of intensity values

bands. The mach band effect can be reduced by breaking the surface into a greater number of smaller polygons.

3. Sharp drop of intensity values on the polygon surface can not be displayed.

9.9.3 Phong Shading

- Phong shading, also known as **normal-vector interpolation shading**, interpolates the surface normal vector N , instead of the intensity.
- By performing following steps we can display polygon surface using Phong shading.
 1. Determine the average unit normal vector at each polygon vertex.
 2. Linearly interpolate the vertex normals over the surface of the polygon.
 3. Apply an illumination model along each scan line to determine projected pixel intensities for the surface points.
- The first steps in the Phong shading is same as first step in the Gouraud shading.
- In the second step the vertex normals are linearly interpolated over the surface of the polygon. This is illustrated in Fig. 9.9.6. As shown in the Fig. 9.9.6, the normal vector N for the scan line intersection point along the edge between vertices 1 and 2 can be obtained by vertically interpolating between edge endpoint normals :

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

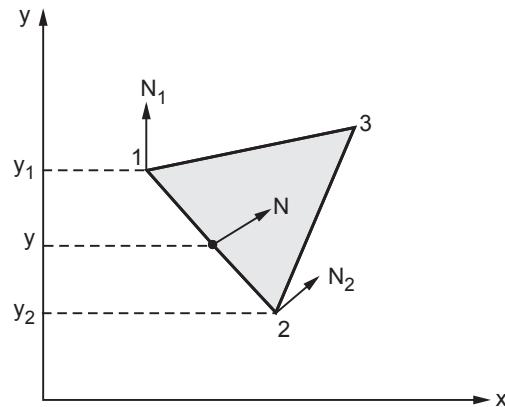


Fig. 9.9.6 Calculation of interpolation of surface normals along a polygon edge

- Like, Gouraud shading, here also we can use incremental methods to evaluate normals between scan lines and along each individual scan line.
- Once the surface normals are evaluated, the surface intensity at that point is determined by applying the illumination model.

Advantages

1. It displays more realistic highlights on a surface. (See Fig. 9.9.7 (d) on next page).
2. It greatly reduces the Mach-band effect.
3. It gives more accurate results.

Disadvantage

1. It requires more calculations and greatly increases the cost of shading steeply.

Fig. 9.9.7 shows the improvement in display of polygon surface using Phong shading over Gouraud shading.

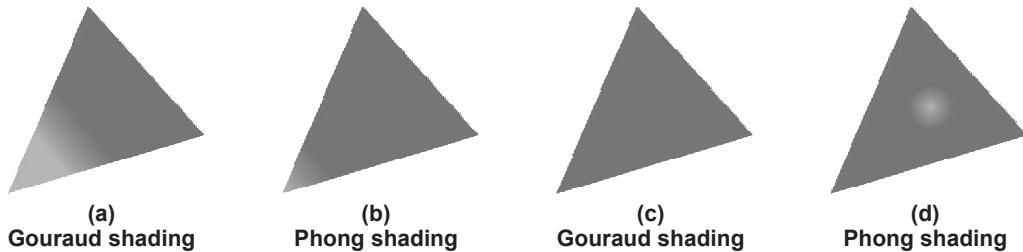


Fig. 9.9.7

Method of Speeding Up Phong Shading Technique

- Phong shading is applied by determining the average unit normal vector at each polygon vertex and then linearly interpolating the vertex normals over the surface of the polygon. Then apply an illumination model along each scan line to calculate projected pixel intensities for the surface points.
- Phong shading can be speeded up by the intensity calculations using a **Taylor - Series expansion** and **triangular surface patches**.
- Since phong shading interpolates normal vectors from vertex normals, we can express the surface normal N at any point (x, y) over a triangle as,

$$N = A_x + B_y + C$$

where A, B, C are determined from three vertex equations .

- $N_k = Ax_k + By_k + C \quad k = 1, 2, 3$ with (x_k, y_k) denoting a vertex positions. Discarding the reflectivity and attenuation parameters, the calculations for light source diffuse reflection from a surface point (x, y) as

$$\begin{aligned} I_{\text{diff}}(x, y) &= \frac{L \cdot N}{|L| |N|} = \frac{L \cdot (A_x + B_y + C)}{|L| |A_x + B_y + C|} \\ &= \frac{(L \cdot A)x + (L \cdot B)y + L \cdot C}{|L| |A_x + B_y + C|} \end{aligned}$$

- Now the expression can be rewritten in the form as,

$$I_{\text{diff}}(x, y) = \frac{ax + by + c}{(dx^2 + exy + fy^2 + gx + hy + i)^{1/2}}$$

where parameters a, b, c and d are used to represent the various dot products. We can express the denominator as a Taylor - series expansion and retain terms up to second degree in x and y.

$$I_{\text{diff}}(x, y) = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0$$

where each T_k is a functions of parameter a, b, c and so forth.

- Using forward differences, we can evaluate above equation with only two additions for each pixel position (x, y) once the initial forward difference parameters have been evaluated. Thus the fast phong shading technique reduces the calculations and speed up the process.

Comparison between Gouraud and Phong's shading

Sr. No.	Gouraud shading	Phong's shading
1.	In this shading model the polygon surface is displayed by linearly interpolating intensity values across the surface.	In this shading model, the surface normal vector N is interpolated, instead of intensity.
2.	It suffers from mach-band effect.	It greatly reduces the mach-band effect.
3.	It requires less calculations than Phong's shading.	It requires more calculations, increasing the cost of shading.

9.9.4 Halftone Shading

- Many displays and hardcopy devices are bilevel. They can only produce two intensity levels. In such displays or hardcopy devices, we can create an apparent increase in the number of available intensities. This is achieved by incorporating multiple pixels positions into the display of each intensity value.
- When we view a very small area from a sufficiently large viewing distance, our eyes average fine details within the small area and record only the overall intensity of the area. This phenomenon of apparent increase in the number of available intensities by considering combine intensity of multiple pixels is known as **halftoning**.
- The halftoning is commonly used in printing black and white photographs in newspapers, magazines and books. The pictures produced by halftoning process are called **halftones**.
- In computer graphics, halftone reproductions are approximated using rectangular pixel regions, say 2×2 pixels or 3×3 pixels. These regions are called **halftone patterns** or **pixel patterns**. Fig. 9.9.8 shows the halftone patterns to create number of intensity levels.

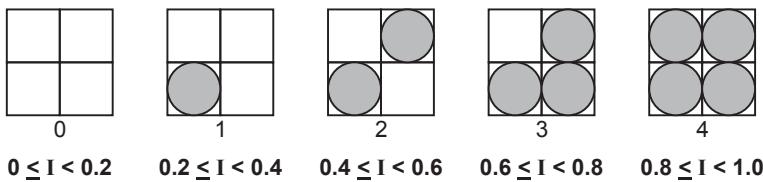


Fig. 9.9.8 (a) 2×2 pixel patterns for creating five intensity levels

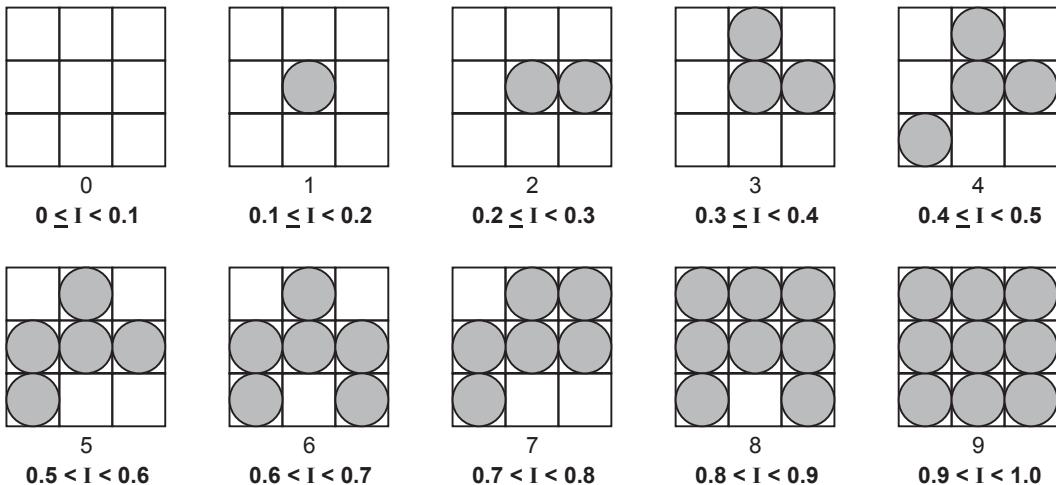


Fig. 9.9.8 (b) 3×3 pixel patterns for creating ten intensity levels

Review Questions

1. Compare Gouraud and Phong's method of shading. SPPU : Dec.-06, Marks 8
2. Explain Gouraud shading. SPPU : May-07, 12,18, Dec.-08, 11, 12,17,18, Marks 4
3. Explain Phong's shading algorithm. SPPU : May-07, 10, 15, Dec.-17, Marks 4
4. What is shading ? What steps are required to shade an object using Gouraud shading? SPPU : May-08, Dec.-09, Marks 6
5. Explain and compare shading algorithms. SPPU : May-14, Dec.-14, Marks 6
6. Write short note on shading algorithm. SPPU : May-11,Dec.-14, Marks 8
7. Explain various surface shading algorithm. SPPU : May-13, Marks 8
8. Enlist and explain shading algorithms with their disadvantages. SPPU : May-17, Marks 7
9. Write a short note on half tone. SPPU : May-17, Marks 3
10. Enlist and explain in detail any two shading algorithms. SPPU : May-18, Marks 7
11. Enlist and explain any two shading algorithms. SPPU : May-19, Marks 6
12. Explain difference between Gouraud shading and Phong shading. SPPU : Dec.-19, Marks 6



UNIT - IV

10

Hidden Surfaces

Syllabus

Introduction, Back face detection and removal, Algorithms: Depth buffer (z), Depth sorts (Painter), Area subdivision (Warnock).

Contents

10.1 <i>Introduction</i>	Dec.-06, 08, 11,
.....	May-07, 08, 11, 12, Marks 8
10.2 <i>Back Face Detection and Removal Algorithm</i>	May-05,06,08,12,13,18,
.....	Dec.-07,11,12,16,17,18,19, · Marks 8
10.3 <i>Depth Buffer (Z) Algorithm</i>	Dec.-05,06,08,11,12,14,16,18,
.....	May-07,10,11,14,16,
.....	17,18,19, · · · · · Marks 8
10.4 <i>Depth Sorts (Painter) Algorithm</i>	Dec.-07,08,11,14,15,16,18,
.....	May-09,10,11,12,16,17,18, · Marks 8
10.5 <i>Area Subdivision (Warnock) Algorithm</i>	Dec.-05,07,10,
.....	May-06,07,15,16,17,18,19, · Marks 8

10.1 Introduction**SPPU : Dec.-06,08,11, May-07,08,11,12**

- In a given set of 3D objects and viewing specification, we wish to determine which lines or surfaces of the objects are visible, so that we can display only the visible lines or surfaces. This process is known as **hidden surfaces** or **hidden line elimination** or **visible surface determination**.
- The hidden line or hidden surface algorithm determines the lines, edges, surfaces or volumes that are visible or invisible to an observer located at a specific point in space.
- These algorithms are broadly classified according to whether they deal with object definitions directly or with their projected images. These two approaches are called **object-space methods** and **image-space methods**, respectively.

Object-Space Method

- Object-space method is implemented in the **physical co-ordinate system** in which objects are described.
- It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible. Object-space methods are generally used in **line-display algorithms**.

Image-Space Method

- Image space method is implemented in the **screen co-ordinate system** in which the objects are viewed.
- In an image-space algorithm, visibility is decided point by point at each pixel position on the view plane.
- Most **hidden line/surface algorithms** use image-space methods.

Review Questions1. *Why are hidden surface algorithms needed ?***SPPU : Dec.-06, 08, 11, May-07, 08, 12, Marks 2**2. *Explain, how hidden lines and surfaces are removed ?***SPPU : May-11, Marks 8****10.2 Back Face Detection and Removal Algorithm****SPPU : May-05,06,08,12,13,18, Dec.-07,11,12,16,17,18,19**

- We know that a polygon has two surfaces, a front and a back, just as a piece of paper does. We might picture our polygons with one side painted light and the other painted dark. But the question is "how to find which surface is light or dark?".
- When we are looking at the light surface, the polygon will appear to be drawn with counter clockwise pen motions and when we are looking at the dark surface

the polygon will appear to be drawn with clockwise pen motions, as shown in the Fig. 10.2.1.

- Let us assume that all solid objects are to be constructed out of polygons in such a way that only the light surfaces are open to the air; the dark faces meet the material inside the object. This means that when we look at an object face from the outside, it will appear to be drawn counterclockwise, as shown in the Fig. 10.2.2.
- If a polygon is visible, the light surface should face towards us and the dark surface should face away from us. Therefore, if the direction of the light face is pointing towards the viewer, the face is visible (a front face), otherwise, the face is hidden (a back face) and should be removed.
- The direction of the light face can be identified by examining the result $N \cdot V > 0$ where

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

V : A vector in the viewing direction from the eye (or "camera") position
(Refer Fig. 10.2.3)

- We know that, the dot product of two vectors, gives the product of the lengths of the two vectors times the cosine of the angle between them. This cosine factor is important to us because if the vectors are in the same direction ($0 \leq \theta < \pi/2$), then the cosine is positive and the overall dot product is positive. However, if the directions are opposite ($\pi/2 < \theta \leq \pi$), then the cosine and the overall dot product is negative (Refer Fig. 10.2.4).

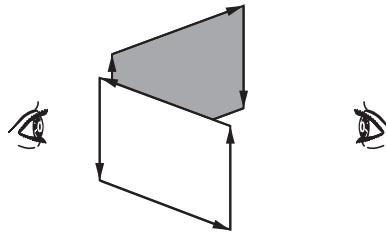


Fig. 10.2.1 Drawing directions

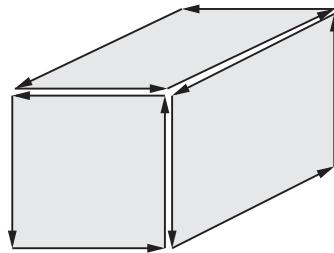


Fig. 10.2.2 Exterior surfaces are coloured light and drawn counter clockwise

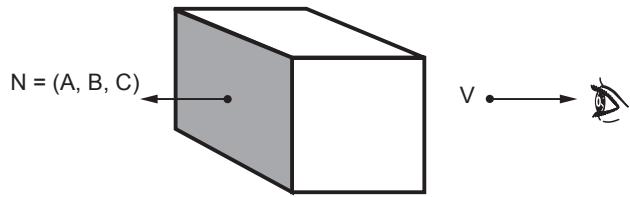


Fig. 10.2.3

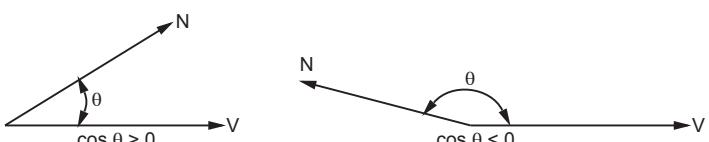


Fig. 10.2.4 Cosine angles between two vectors

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

$$V \cdot N = V_z C$$
- So that we only have to consider the sign of C , the z component of the normal vector N . Now, if the z component is positive, then the polygon faces towards the viewer, if negative, it faces away.

Review Question

1. Explain backface removal algorithm.

SPPU : May-05,06,08,12,13,18, Dec.-07,11,12,16,17,18,19, Marks 8

10.3 Depth Buffer (Z) Algorithm

SPPU : Dec.-05,06,08,11,12,14,16,18, May-07,10,11,14,16,17,18,19

- One of the simplest and commonly used image space approach to eliminate hidden surfaces is the **Z-buffer** or **depth buffer** algorithm. It is developed by Catmull.
- This algorithm **compares surface depths** at each pixel position on the projection plane.
- The surface depth is measured from the view plane along the z axis of a viewing system.
- When object description is converted to projection co-ordinates (x, y, z), each pixel position on the view plane is specified by x and y co-ordinate, and z value gives the depth information. Thus object depths can be compared by comparing the z - values.
- The Z-buffer algorithm is usually implemented in the normalized co-ordinates, so that z values range from 0 at the back clipping plane to 1 at the front clipping plane.
- The implementation requires another buffer memory called **Z-buffer** along with the frame buffer memory required for raster display devices.
- A Z-buffer is used to store depth values for each (x, y) position as surfaces are processed and the frame buffer stores the intensity values for each position.
- At the beginning Z-buffer is initialized to zero, representing the z -value at the back clipping plane and the frame buffer is initialized to the background colour. Each surface listed in the display file is then processed, one scan line at a time,

calculating the depth (z-value) at each (x, y) pixel position. The calculated depth value is compared to the value previously stored in the Z-buffer at that position.

- If the calculated depth values is greater than the value stored in the Z-buffer, the new depth value is stored and the surface intensity at that position is determined and placed in the same xy location in the frame buffer.
- For example, in Fig. 10.3.1 among three surfaces, surface S_1 has the smallest depth at view position (x, y) and hence highest z value. So it is visible at that position.

Z-buffer Algorithm

1. Initialize the Z-buffer and frame buffer so that for all buffer positions

$$\text{Z-buffer } (x, y) = 0 \text{ and frame-buffer } (x, y) = I_{\text{background}}$$

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

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

If $z > \text{Z-buffer } (x, y)$, then set

$$\text{Z-buffer } (x, y) = z, \text{ frame-buffer } (x, y) = I_{\text{surface}} (x, y)$$

3. Stop

- Note that, $I_{\text{background}}$ is the value for the background intensity and I_{surface} is the projected intensity value for the surface at pixel position (x, y) . After processing of all surfaces, the Z-buffer contains depth values for the visible surfaces and the frame buffer contains the corresponding intensity values for those surfaces.

- To calculate z-values, the plane equation

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

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

- Therefore, we can write

$$z = \frac{-Ax - By - D}{C}$$

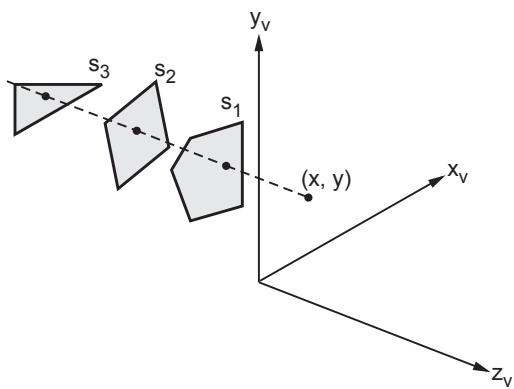


Fig. 10.3.1

- Note, if at (x, y) the above equation evaluates to z_1 , then at $(x + \Delta x, y)$ the value of z , is

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

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

Advantages

1. It is easy to implement.
2. It can be implemented in hardware to overcome the speed problem.
3. Since the algorithm processes objects one at a time, the total number of polygons in a picture can be arbitrarily large.

Disadvantages

1. It requires an additional buffer and hence the large memory.
2. It is a time consuming process as it requires comparison for each pixel instead of for the entire polygon.

Review Questions

1. How does the Z-buffer algorithm determine which surfaces are hidden ?

SPPU : Dec.-05, 06, 08; May-07,19, Marks 8

2. Explain Z-buffer algorithm and its applications.

SPPU : May-10,11,16,17,18 Dec.-11,12,16,18, Marks 8

3. Enlist hidden face removal alorithm and explain any two.

SPPU : May-14, Dec.-14, Marks 7

10.4 Depth Sorts (Painter) Algorithm

SPPU : Dec.-07,08,11,14,15,16,18, May-09,10,11,12,16,17,18

- The basic idea of the painter's algorithm developed by Newell and Sancha, is to paint the polygons into the frame buffer in order of decreasing distance from the viewpoint. This process involves following basic functions.
- 1. Sorting of polygons in order of decreasing depth.
- 2. Resolving any ambiguities. This may cause when the polygon's z extents overlap, i.e., splitting polygons if necessary.
- 3. Scan conversion of polygons in order, starting with the polygon of greatest depth.
- The algorithm gets its name from the manner in which an oil painting is created. The artist begins with the background. He then adds the most distant object and

then the nearer object and so forth. There is no need to erase portions of background; the artist simply paints on top of them. The new paint covers the old so that only the newest layer of paint is visible. This is illustrated in Fig. 10.4.1.

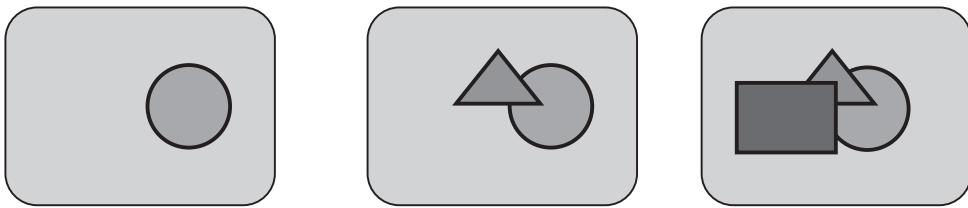


Fig. 10.4.1 Painter's algorithm

- Using the similar technique, we first sort the polygons according to their distance from the view point. The intensity values for the farthest polygon are then entered into the frame buffer. Taking each polygon in succeeding polygon in turn (in decreasing depth order), polygon intensities are painted on the frame buffer over the intensities of the previously processed polygons. This process is continued as long as no overlaps occur.
 - If depth overlap is detected by any point in the sorted list, we have to make some additional comparisons to determine whether any of the polygon should be reordered.
 - We can check whether any polygon Q does not obscure polygon P by performing following steps :
1. The z-extents of P and Q do not overlap, i.e. $z_Q \text{ max} < z_P \text{ min}$.
(See Fig. 10.4.2 (a) on next page).
 2. The y-extents of P and Q do not overlap (See Fig. 10.4.2 (b) on next page.)
 3. The x-extents of P and Q do not overlap.
 4. Polygon P lying entirely on the opposite side of Q's plane from the view port.
(See Fig. 10.4.2 (c) on next page)
 5. Polygon Q lying entirely on the same side of P's plane as the viewport.
(See Fig. 10.4.2 (d) on next page).
 6. The projections of the polygons P and Q onto the xy screen do not overlap.
 - If all these five tests fail, we assume for the moment that P actually obscures Q and therefore test whether Q can be scan-converted before P. Here, we have to repeat tests 4 and 5 for Q. If these tests also fail then we can say that there is no order in which P and Q can be scan converted correctly and we have to split either P or Q into two polygons. The idea behind the splitting is that the split polygons may not obscure other polygon.

Algorithm

1. Sort all polygons in order of decreasing depth.
2. Determine all polygons Q (preceding P) in the polygon list whose z-extents overlap that of P.

3. Perform test 2 through 6 for each Q

- If every Q passes the tests, scan convert the polygon P.
- If test fails for some Q, swap P and Q in the list and make the indication that Q is swapped. If Q has already been swapped, use the plane containing polygon P to divide polygon Q into two polygons, Q_1 and Q_2 . Replace Q with Q_1 and Q_2 . Repeat step 3.

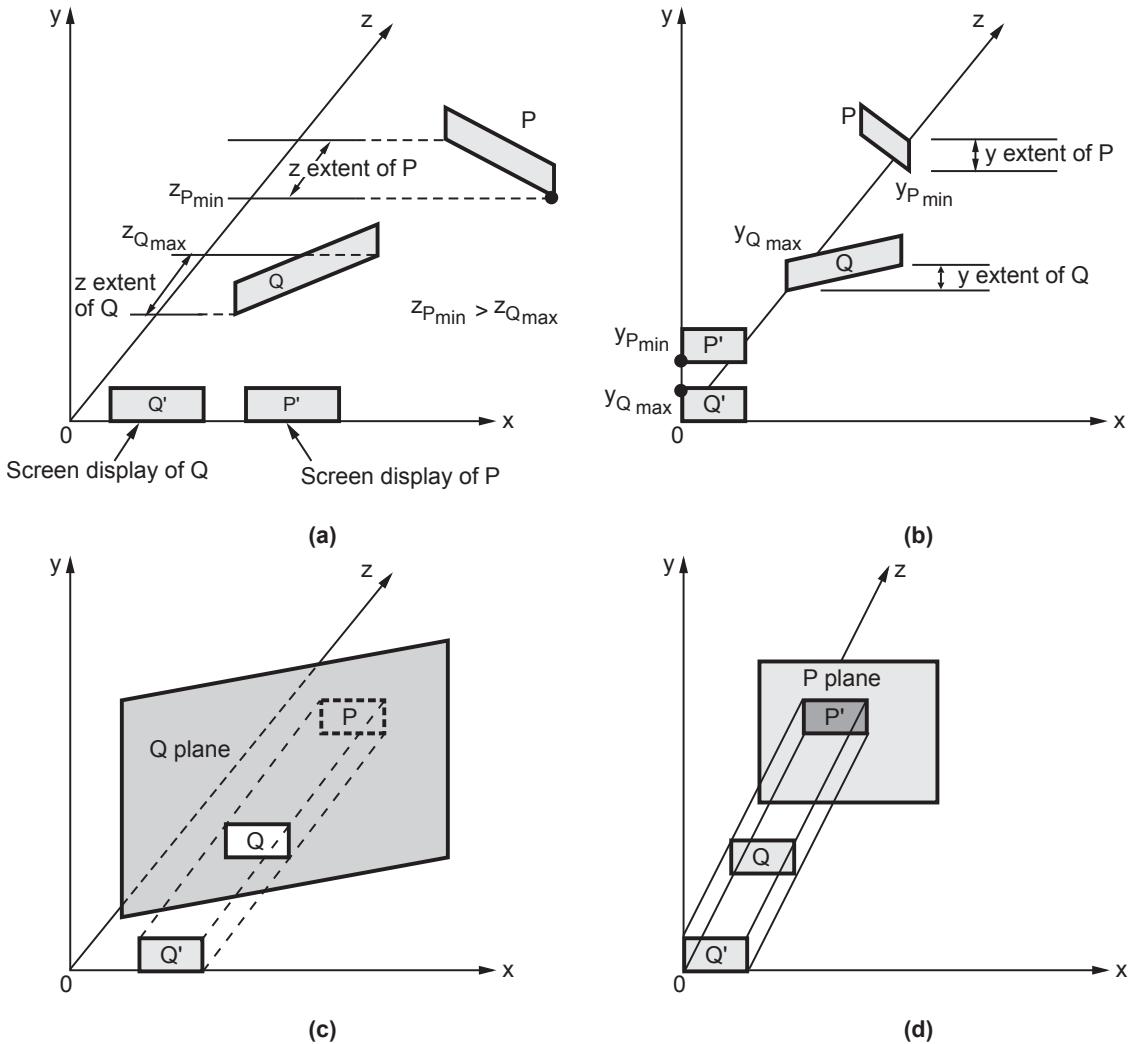


Fig. 10.4.2

Advantages of Painter's Algorithm

- Simple
- Easy transparency

Disadvantages

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

Review Questions

1. Explain Painter's algorithm and its applications.

SPPU : Dec.-07,08,11,15,16,18, May-09,10,11,12,16,17,18, Marks 8

2. Why is Painter's algorithm a priority algorithm ?

SPPU : Dec.-08, 14, May-09, Marks 4

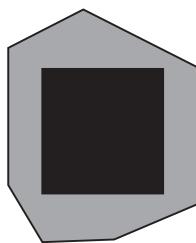
10.5 Area Subdivision (Warnock) Algorithm

SPPU : Dec.-05,07,10, May-06,07,15,16,17,18,19

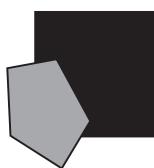
- An interesting approach to the hidden-surface problem was developed by Warnock.
- He developed area subdivision algorithm which subdivides each area into four equal squares.
- At each stage in the recursive-subdivision process, the relationship between projection of each polygon and the area of interest is checked for four possible relationships :

 1. Surrounding Polygon - One that completely encloses the (shaded) area of interest (See Fig. 10.5.1 (a)).
 2. Overlapping or Intersecting Polygon - One that is partly inside and partly outside the area (See Fig. 10.5.1 (b)).
 3. Inside or Contained Polygon - One that is completely inside the area (See Fig. 10.5.1 (c)).
 4. Outside or Disjoint Polygon - One that is completely outside the area (See Fig. 10.5.1 (d)).

- After checking four relationships we can handle each relationship as follows :



(a) Surrounding



(b) Overlapping



(c) Inside or Contained



(d) Outside or Disjoint

Fig. 10.5.1 Possible relationships with polygon surfaces and the area of interest

1. If all the polygons are disjoint from the area, then the background colour is displayed in the area.
 2. If there is only one intersecting or only one contained polygon, then the area is first filled with the background colour and then the part of the polygon contained in the area is filled with colour of polygon.
 3. If there is a single surrounding polygon, but no intersecting or contained polygons, then the area is filled with the colour of the surrounding polygon.
 4. If there are more than one polygon intersecting, contained in, or surrounding the area then we have to do some more processing.
- See Fig. 10.5.2 on next page. In Fig. 10.5.2 (a), the four intersections of surrounding polygon are all closer to the viewpoint than any of the other intersections. Therefore, the entire area is filled with the colour of the surrounding polygon.

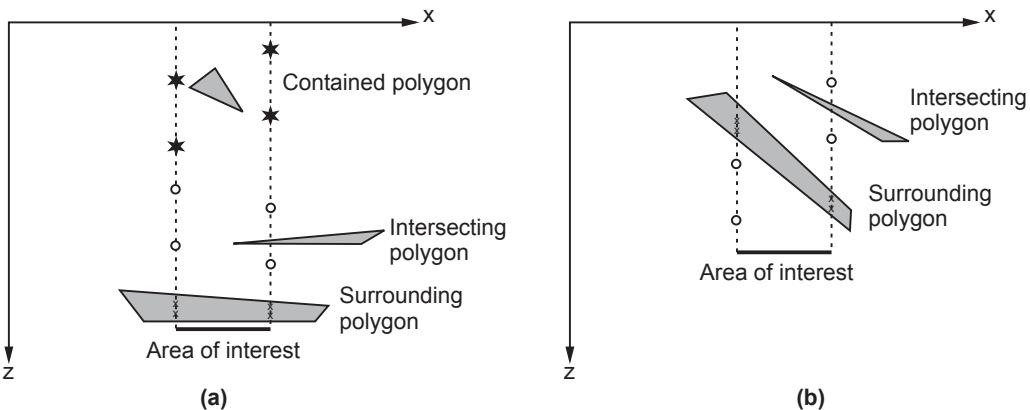


Fig. 10.5.2

- However, Fig. 10.5.2 (b) shows that surrounding polygon is not completely in front of the intersecting polygon. In such case, we cannot make any decision and hence Warnock's algorithm subdivides the area to simplify the problem. This is illustrated in Fig. 10.5.3. As shown in the Fig. 10.5.3 (a) we cannot make any decision about which polygon is in front of the other. But after dividing area of interest polygon 1 is ahead of the polygon 2 in left area and polygon 2 is ahead of polygon 1 in the right area. Now we can fill these two areas with corresponding colours of the polygons.
- The Warnock's algorithm stops subdivision of area only when the problem is simplified or when area is only a single pixel.

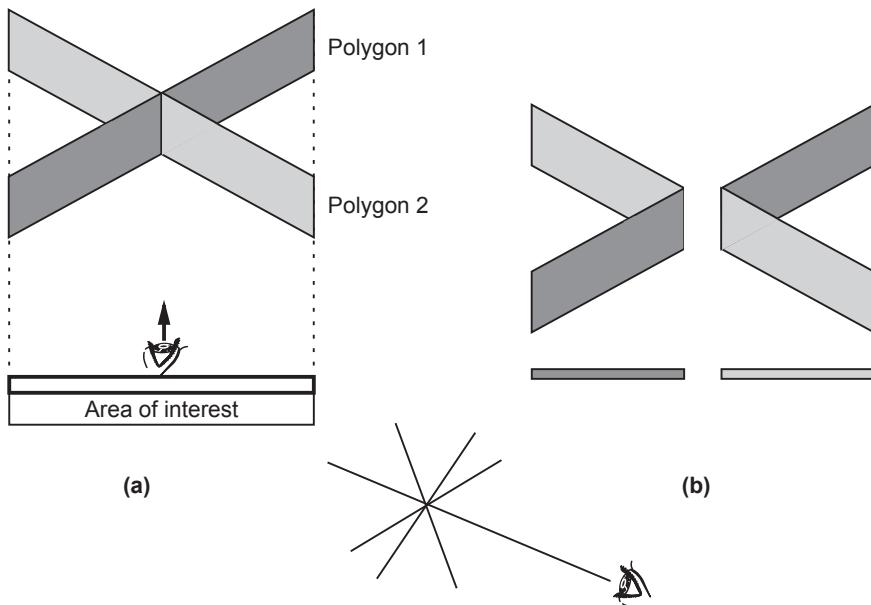


Fig. 10.5.3

Algorithm

1. Initialize the area to be the whole screen.
2. Create the list of polygons by sorting them with their z-values of vertices. Don't include disjoint polygons in the list because they are not visible.
3. Find the relationship of each polygon.
4. Perform the visibility decision test.
 - a) If all the polygons are disjoint from the area, then fill area with background colour.
 - b) If there is only one intersecting or only one contained polygon then first fill entire area with background colour and then fill the part of the polygon contained in the area with the colour of polygon.
 - c) If there is a single surrounding polygon, but no intersecting or contained polygons, then fill the area with the colour of the surrounding polygon.
 - d) If surrounding polygon is closer to the viewpoint than all other polygons, so that all other polygons are hidden by it, fill the area with the colour of the surrounding polygon.
 - e) If the **area is the pixel** (x, y) and neither a, b, c, nor d applies, compute the z co-ordinate at pixel (x, y) of all polygons in the list. The pixel is then set to colour of the polygon which is closer to the viewpoint.
5. If none of the above tests are true then subdivide the area and go to step 2.

Advantages

1. It follows the divide-and-conquer strategy, therefore, parallel computers can be used to speed up the process.
2. Extra memory buffer is not required.

Review Questions

1. Explain Warnock's algorithm. **SPPU : Dec.-05,07,10, May-06,07,16,17,18,19, Marks 8**

2. Why this algorithm is also called as area subdivision algorithm ? **SPPU : Dec.-10, Marks 4**

3. What are the advantages of Warnock's algorithm ? **SPPU : May-15,18, Marks 3**



UNIT - V

11

Curves and Fractals

Syllabus

Curves : Introduction, Interpolation and Approximation, Blending function, 8-Spline curve, Bezier curve,

Fractals : Introduction, Classification, Fractal generation: snowflake, Triadic curve, Hilbert curve, Applications.

Contents

11.1	<i>Introduction</i>	
11.2	<i>Curve Generation</i>	<i>Dec.-05,07,10,11, May-06,10, Marks 10</i>
11.3	<i>Interpolation, Approximation and Blending Function</i>	<i>Dec.-05,10,11,12 May-07,08,09, Marks 8</i>
11.4	<i>Interpolating Polygons</i>	<i>Dec.-08, Marks 4</i>
11.5	<i>Spline Representations</i>	<i>Dec.-10,18, May-11,12, Marks 8</i>
11.6	<i>Bezier Curves</i>	<i>May-05,06,07,10,11,13,14,17,18,19, Dec.-05,08,12,14,15,18 Marks 10</i>
11.7	<i>B-Spline Curve</i>	<i>May-05,06,07,08,10,11,13,15,17,18,19, Dec.-05,06,10,11,12,16,17 Marks 8</i>
11.8	<i>Fractals</i>	<i>May-05,06,08,09,13,14,15,17,18,19, Dec.-05,07,08,10,12,15,16,17,18,19 Marks 8</i>

11.1 Introduction

- We have seen line, circle and polygon generation algorithms, we learnt how to represent and manipulate line segments and polygons and we have also seen transformations and clipping of them.
- However, many real world objects are inherently smooth and involve curves to represent them.
- Some natural objects are neither perfectly flat nor smoothly curved but often have rough, jagged contours.
- In this chapter, we will see the methods for generating curved lines and curved surfaces.

11.2 Curve Generation

SPPU : Dec.-05,07,10,11, May-06,10

- We can use two approaches to draw curved lines. One approach is to use a curve generation algorithm such as DDA. In this approach a true curve is created.
- In the second approach the curve is approximated by a number of small straight line segments. This can be achieved with the help of **interpolation techniques**.

11.2.1 Circular Arc Generation using DDA Algorithm

- Digital differential analyzer algorithm uses the differential equation of the curve.
- The differential equations for simple curve such as circle is fairly easy to solve.
- Let us see the DDA algorithm for generating circular arcs. The equation for an arc in the angle parameters can be given as

$$\begin{aligned} x &= R \cos\theta + x_0 \\ y &= R \sin\theta + y_0 \end{aligned} \quad \dots (11.2.1)$$

where (x_0, y_0) is the centre of curvature, and R is the radius of arc. (See Fig. 11.2.1).

- Differentiating equation (11.2.1) we get,

$$dx = -R \sin\theta d\theta$$

$$dy = R \cos\theta d\theta \quad \dots (11.2.2)$$

- From equation (11.2.1) we can solve for $R \cos\theta$ and $R \sin\theta$ as follows.

$$x = R \cos\theta + x_0$$

$$\therefore R \cos\theta = x - x_0 \quad \text{and}$$

$$R \sin\theta = y - y_0 \quad \dots (11.2.3)$$

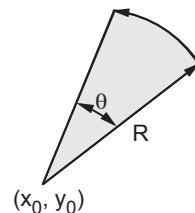


Fig. 11.2.1

- Substituting values of $R \cos \theta$ and $R \sin \theta$ from equation (11.2.3) in equation (11.2.2) we get,

$$\begin{aligned} dx &= -(y - y_0) d\theta \text{ and} \\ dy &= (x - x_0) d\theta \end{aligned} \quad \dots (11.2.4)$$

- The values of dx and dy indicate the increment in x and y respectively, to be added in the current point on the arc to get the next point on the arc. Therefore, we can write

$$\begin{aligned} x_2 &= x_1 + dx = x_1 - (y_1 - y_0) d\theta \\ y_2 &= y_1 + dy = y_1 + (x_2 - x_0) d\theta \end{aligned} \quad \dots (11.2.5)$$

- The equation (11.2.5) forms the basis for arc generation algorithm. From equation (11.2.5) we can see that the next point on the arc is the function of $d\theta$. To have a smooth curve, the neighbouring points on the arc should be close to each other. To achieve this, the value of $d\theta$ should be small enough not to leave gaps in the arc. Usually, the value of $d\theta$ can be determined from the following equation.

$$d\theta = \text{Min}(0.01, 1/(3.2 \times (|x - x_0| + |y - y_0|)))$$

Algorithm

- Read the centre of a curvature, say (x_0, y_0) .
- Read the arc angle, say θ .
- Read the starting point of the arc, say (x, y) .
- Calculate $d\theta$.
- $d\theta = \text{Min}(0.01, 1 / (3.2 \times (|x - x_0| + |y - y_0|)))$
- Initialize Angle = 0.
- While (Angle < θ)
 - do
 - { Plot (x, y)
 - $x = x - (y - y_0) \times d\theta$
 - $y = y + (x - x_0) \times d\theta$
 - Angle = Angle + $d\theta$
- Stop.

Problems in True-Curve Generation Approach

- To specify a curve, we need more information than just its end points.

2. It is difficult to apply transformations. For example, a circle when scaled in only one direction becomes an ellipse. If our algorithm supports only circular arc generation then ability to scale pictures is limited.
3. New clipping algorithm is required to clip arcs.
4. The curve generation algorithms for curves other than circular or elliptical such as airplane wings or cars or human faces, are complex.

Review Questions

1. Explain curve generation methods with example.

SPPU : Dec.-05, Marks 8

2. What is true curve generation ? Write a Pseudo code to implement DDA arc generation.

SPPU : May-06, Dec.-07, 11, Marks 10

3. Explain the term control points in curve drawing.

SPPU : May-10, Marks 4

4. Write short note on True curve generation.

SPPU : Dec.-10, Marks 2

11.3 Interpolation, Approximation and Blending Function

SPPU : Dec.-05,10,11,12, May-07,08,09

- In the last section we have seen limitations of true curve generation approach.
- Furthermore in practice we have to deal with some complex curves for which no direct mathematical function is available. Such curves can be drawn using approximation methods.
- If we have set of sample points which lie on the required curve, then we can draw the required curve by filling portions of the curve with the pieces of known curves which pass through nearby sample points.
- The gap between the sample points can be filled by finding the co-ordinates of the points along the known **approximating curve** and connecting these points with line segments as shown in the Fig. 11.3.1.

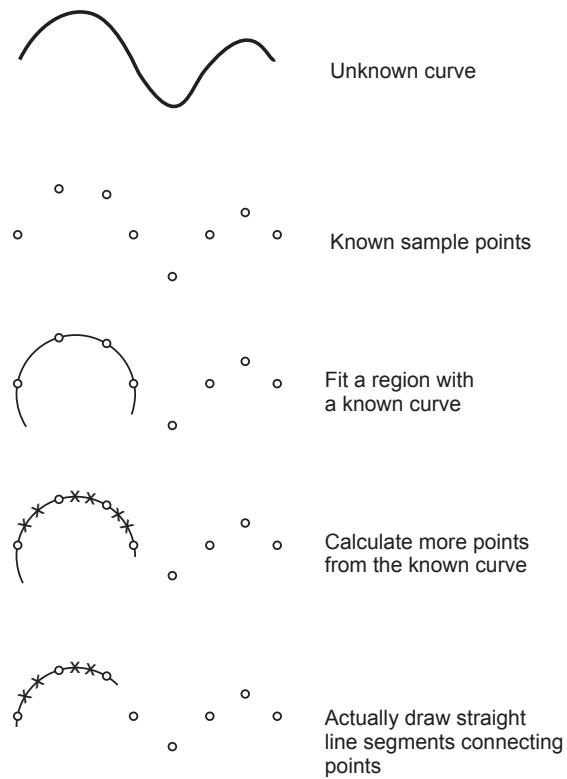


Fig. 11.3.1 The interpolation process

- The main task in this process is to find the suitable mathematical expression for the known curve.
- There are polynomial, trigonometric, exponential and other classes of functions that can be used to approximate the curve. Usually polynomial functions in the parametric form are preferred. The polynomial functions in the parametric form can be given as

$$x = f_x(u)$$

$$y = f_y(u)$$

$$z = f_z(u)$$

- We can realize from above equations that the difference between 2 and 3 dimensions is just the addition of the third equation for z.
- Furthermore the parametric form treats all the three dimensions equally and allows multiple values (several values of y or z for a given x value). Due to these multiple values curves can double back or even cross themselves, as shown in Fig. 11.3.2.

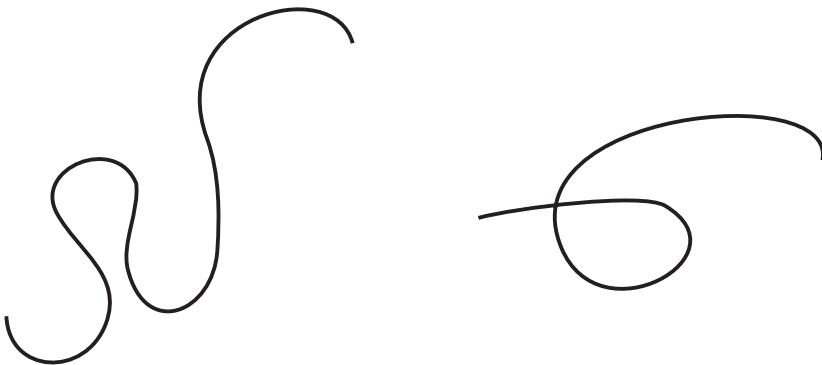


Fig. 11.3.2 Representation of curves with double back or crossing themselves

- We have seen that, we have to draw the curve by determining the intermediate points between the known sample points. This can be achieved using interpolation techniques. Let us see the interpolation process.
- Suppose we want a polynomial curve that will pass through n sample points. $(x_1, y_1, z_1), (x_2, y_2, z_2), \dots, (x_n, y_n, z_n)$
- We will construct the function as the sum of terms, one term for each sample point. These functions can be given as

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

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

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

- The function $B_i(u)$ is called '**blending function**'. For each value of parameter u , the blending function determines how much the i^{th} sample point affects the position of the curve.
- In other words we can say that each sample points tries to pull the curve in its direction and the function $B_i(u)$ gives the strength of the pull.
- If for some value of u , $B_i(u) = 1$ for unique value of i (i.e. $B_i(u) = 0$ for other values of i) then i^{th} sample point has complete control of the curve and the curve will pass through i^{th} sample point.
- For different value of u , some other sample point may have complete control of the curve. In such case the curve will pass through that point as well.
- In general, the blending functions give control of the curve to each of the sample points in turn for different values of u .
- Let us assume that the first sample point (x_1, y_1, z_1) has complete control when $u = -1$, the second when $u = 0$, the third when $u = 1$, and so on. i.e.
 - When $u = -1 \Rightarrow B_1(u) = 1$ and 0 for $u = 0, 1, 2, \dots, n-2$
 - When $u = 0 \Rightarrow B_2(u) = 1$ and 0 for $u = -1, 1, \dots, n-2$
 - $\vdots \quad \vdots \quad \vdots$
 - When $u = (n-2) \Rightarrow B_n(u) = 1$ and 0 for $u = -1, 0, \dots, (n-1)$
- To get $B_1(u) = 1$ at $u = -1$ and 0 for $u = 0, 1, 2, \dots, n-2$, the expression for $B_1(u)$ can be given as

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

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

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

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

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

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

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

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

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

$$x = x_1 B_1(u) + x_2 B_2(u) + x_3 B_3(u) + x_4 B_4(u)$$

$$y = y_1 B_1(u) + y_2 B_2(u) + y_3 B_3(u) + y_4 B_4(u)$$

$$z = z_1 B_1(u) + z_2 B_2(u) + z_3 B_3(u) + z_4 B_4(u)$$

- It is possible to get intermediate points between two sampling points by taking values of u between the values of u related to the two sample points under consideration. For example, we can find the intermediate points between second and third sample points for which values of u are 0 and 1, respectively; by taking values of u between 0 and 1. This is shown in Fig. 11.3.3.

- The subsequent intermediate points can be obtained by repeating the same procedure. Finally the points obtained by this procedure are joined by small straight line segments to get the approximated curve.
- Initially, sample points (1, 2, 3, 4) are considered and intermediate points between (2, 3) are obtained. Then sample point at one end is discarded and sample point at the other end is added to get new sample points (2, 3, 4, 5). Now the curve between sample points (3, 4) is approximated. The subsequent intermediate points can be obtained by repeating the same procedure. The initial and final portions of the curve require special treatment. For the first four points (1, 2, 3, 4) we have to draw region between points (1, 2) with u values between -1 and 0.
- Similarly the blending function for very last step of the curve should be evaluated with u values between 1 and 2.

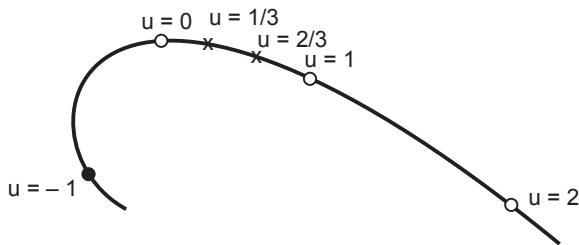


Fig. 11.3.3 Determining intermediate points for approximation of curve

Interpolating Algorithm

1. Get the sample points.
2. Get intermediate values of u to determine intermediate points.
3. Calculate blending function values for middle section of the curve.
4. Calculate blending function values for first section of the curve.
5. Calculate blending function values for the last section of the curve.
6. Multiply the sample points by blending functions to give points on approximation curve.
7. Connect the neighbouring points using straight line segments.
8. Stop.

Review Questions

1. Explain interpolation for curve generation.

SPPU : Dec.-05, May-07, Marks 8

2. What is interpolation ? Explain Lagrange interpolation method.

SPPU : May-08, 09, Dec.-11,12, Marks 8

3. Write short note on interpolating algorithm.

SPPU : Dec.-10, 12, Marks 6

11.4 Interpolating Polygons

SPPU : Dec.-08

- Blending functions can also be used to round the sides of polygon.
- Smoothing of each side of the polygon is done by replacing it with several small line segments.
- We start out with a polygon that has only a few sides and end up with a polygon which has many more sides and appears smoother due to the interpolation.
- This is illustrated in Fig. 11.4.1.

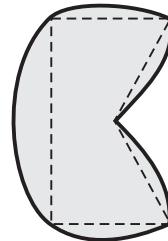


Fig. 11.4.1 Smoothing of a polygon

Algorithm

1. Read the original number of polygon sides, say N .
2. Read the vertices of original polygon.
3. Read the blending function values.
4. Read the count for the number of small line segments per original polygon side.
5. Smooth one side of original polygon by stepping through four blending functions.

6. Repeat step 5 according to number of small line segments required per original polygon side.
7. Repeat step 5 and 6 for each side of the original polygon.
8. Draw the polygon using small line segments.
9. Stop.

Review Question

1. Write a note on interpolating polygons.

SPPU : Dec.-08, Marks 4

11.5 Spline Representations

SPPU : Dec.-10,18, May-11,12

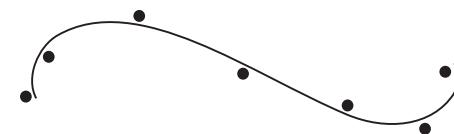
- To produce a smooth curve through a designated set of points, a flexible strip called **spline** is used.
- Such a spline curve can be mathematically described with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections.

11.5.1 Interpolation and Approximation

- We can specify a spline curve by giving a set of co-ordinate positions, called **control points**, which indicates the general shape of the curve.
- When polynomial sections are fitted so that the curve passes through all control points, as shown in the Fig. 11.5.1 (a), the resulting curve is said to **interpolate** the set of control points.
- On the other hand, when the polynomials are fitted to the path which is not necessarily passing through all control points, the resulting curve is said to approximate the set of control points. This is illustrated in the Fig. 11.5.1 (b).



(a) Interpolation spline



(b) Approximation spline

Fig. 11.5.1

Convex Hull

- The convex polygon boundary that encloses a set of control points is called the **convex hull**. One way to envision the shape of a convex hull is to imagine a rubber band stretched around the positions of the control points so that each control point is either on the perimeter of the hull or inside it. Convex hulls provide a measure for the deviation of a curve or surface from the region

bounding the control points. Some splines are bounded by the convex hull, thus ensuring that the polynomials smoothly follow the control points without erratic oscillations. Also, the polygon region inside the convex hull is useful in some algorithms as a clipping region.

11.5.2 Geometric and Parametric Continuity

- To ensure a smooth transition from one section of a piecewise parametric curve to the next, we can impose various continuity conditions at the connection points. We see **parametric continuity** and **geometric continuity** conditions.
- In geometric continuity we require parametric derivatives of two sections to be proportional to each other at their common boundary instead of equal to each other.
- Parametric continuity is set by matching the parametric derivatives of adjoining two curve sections at their common boundary.
- In **zero order** parametric continuity, given as C^0 , it means simply the curve meet and same is for zero order geometric continuity.
- In **first order** parametric continuity called as C^1 means that first parametric derivatives of the coordinate functions for two successive curve sections are equal at the joining point and geometric first order continuity means the parametric first derivative are proportional at the intersection of two successive sections.
- Second order** parametric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are same at the intersection and for second order geometric continuity or C^2 continuity means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under C^2 continuity curvature of the two curve sections match at the joining positions.

Two curves

$$r(t) = (t^2 - 2t, t)$$

$$n(t) = (t^2 + 1, t + 1)$$

C^1 and G^1 are continuous at $r(1) = n(0)$

$$\text{Derivative } r(t) = 2t - 2, 1$$

$$r(1) = 2 - 2, 1 = 0, 1$$

$$\text{Derivative } n(t) = 2t, 1$$

$$n(0) = 0, 1$$

$\therefore r(1) = n(0)$, two curves are continuous.



(a) Zero order continuity



(b) First order continuity



(c) Second order continuity

Example 11.5.1 Show that two curves $n(t) = (t^2 + 2t - 2, t^2)$ and $r(t) = (t^2 + 2t + 1, t + 1)$ are both C^0 and G^0 continuous where they join at $n(1) = r(0)$. Do they meet C^1 and G^1 continuity.

Solution : $n(t) = (t^2 + 2t - 2, t^2)$ and $r(t) = (t^2 + 2t + 1, t + 1)$

Zero order parametric continuity, described as C^0 continuity, means simply that the curves meet. That is, the values of x , y and z evaluated at u_2 for the first curve section are equal, respectively, to the values of x , y and z evaluated at u_1 for the next curve section. The **zero-order geometric continuity** described as G^0 continuity, is the same as zero-order parametric continuity.

$$\text{We have, } n(1) = (1^2 + 2 - 2, 1^2) = (1, 1)$$

$$r(0) = (0^2 + 0 + 1, 0 + 1) = (1, 1)$$

Therefore, we can say that both curves are C^0 and G^0 continuous at $n(1)$ and $r(0)$. To check for C^1 and G^1 continuity we have to take first derivative of both the curves :

$$\text{Derivative } n(t) = (2t + 2, 2t)$$

$$\text{Derivative } r(t) = (2t + 2, 1)$$

$$\therefore n(1) = (2 + 2, 2) = (4, 2)$$

$$r(0) = (2, 1)$$

Since $n(1) \neq r(0)$, the two curves are not C^1 and G^1 continuous at $n(1)$ and $r(0)$.

11.5.3 Spline Specifications

There are three basic ways of specifying spline curve :

- We can state the set of boundary conditions that are imposed on the spline.
- We can state the matrix that characterizes the spline or
- We can state the set of blending functions that calculate the positions along the curve path by specifying combination of geometric constraints on the curve.

Why to use cubic polynomials ?

- Polylines and polygons are first-degree, piecewise linear approximation to curves and surfaces, respectively. But this lower degree polynomials give too little flexibility in controlling the shape of the curve.
- The higher-degree polynomials give reasonable design flexibility, but introduce unwanted wiggles and also require more computation.
- For this reason the third-degree polynomials are most often used for representation of curves. These polynomials are commonly known as **cubic polynomials**.

- We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of equations :

$$x(u) = ax u^3 + bx u^2 + cx u + dx$$

$$y(u) = ay u^3 + by u^2 + cy u + dy$$

$$z(u) = az u^3 + bz u^2 + cz u + dz \quad (0 \leq u \leq 1) \quad \dots (11.5.1)$$

- For each of these three equations, we need to determine the values of the four coefficients a, b, c and d in the polynomial representation for each of the n curve sections between the n + 1 control points.
- We do this by setting enough boundary conditions at the joints between curve sections so that we can obtain numerical values for all the coefficient.
- Let us see the common methods for setting the boundary conditions for cubic interpolation splines.

Review Questions

1. Give the various methods for specifying spline curve.

2. Why is cubic form chosen for representing curve ?

SPPU : Dec.-10, Marks 6

3. Explain the term control points and order of continuity in curve drawing. What is convex hull ?

SPPU : May-11, 12, Marks 8

4. Write a short note on interpolation and approximation.

SPPU : Dec.-18, Marks 4

11.6 Bezier Curves

SPPU : May-05,06,07,10,11,13,14,17,18,19, Dec.-05,08,12,14,15,18

- Bezier curve is another approach for the construction of the curve.
- A Bezier curve is determined by a defining polygon.
- Bezier curves have a number of properties that make them highly useful and convenient for curve and surface design.
- They are also easy to implement. Therefore, Bezier curves are widely available in various CAD systems and in general graphic packages.
- In this section we will discuss the cubic Bezier curve.
- The reason for choosing cubic Bezier curve is that they provide reasonable design flexibility and also avoid the large number of calculations.
- In general, a Bezier curve section can be fitted to any number of control points. However, as number of control points increases, the degree of the Bezier polynomial also increases. Because in a Bezier curve a degree of a polynomial is one less than the number of control points used.

- For example, three control points generate a parabola, four points generate cubic curve and so on. This is illustrated in Fig. 11.6.1.

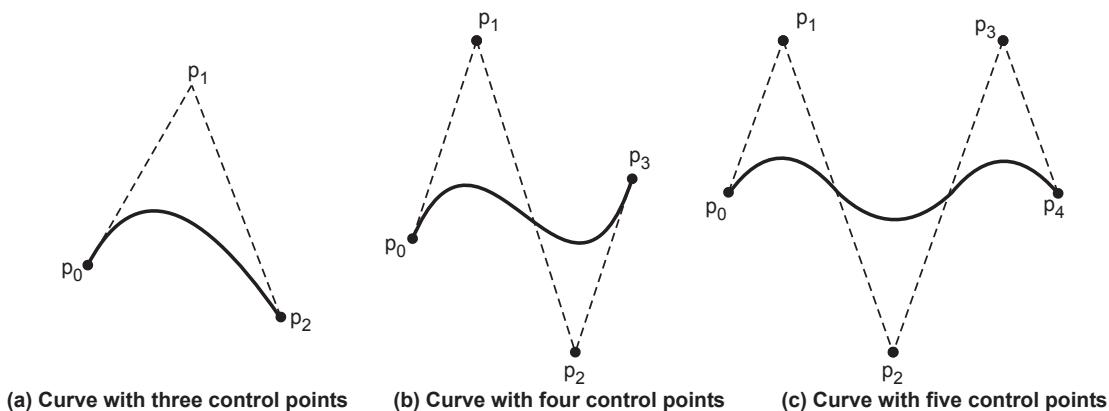


Fig. 11.6.1

- The Bezier curves can be specified with boundary conditions, with a characterizing matrix or with blending functions. Out of these, blending function specification is the most convenient way for general Bezier curves.
- Consider that the curve has $n + 1$ control points : $p_k = (x_k, y_k, z_k)$... where k varies from 0 to n . The co-ordinates of these control points can be blended to produce position vector $P(u)$, which gives the path of an approximating Bezier polynomial function between p_0 and p_n . The position vector can be given by,

$$P(u) = \sum_{k=0}^n p_k BEZ_{k,n}(u), \quad 0 \leq u \leq 1 \quad \dots (11.6.1)$$

Bernstein polynomials

- The Bezier blending functions $BEZ_{k,n}(u)$ are the Bernstein polynomials. They are specified as,

$$BEZ_{k,n}(u) = C(n, k) u^k (1-u)^{n-k}$$

where the $C(n, k)$ are the binomial coefficients. Binomial coefficients are given by,

$$C(n, k) = \frac{n!}{k! (n-k)!} \quad \dots (11.6.2)$$

- Equivalently, we can define Bezier blending functions with the recursive calculation as,

$$BEZ_{k,n}(u) = (1-u) BEZ_{k,n-1}(u) + u BEZ_{k-1,n-1}(u), \quad n > k \geq 1 \quad \dots (11.6.3)$$

with $BEZ_{k,k} = u^k$ and $BEZ_{0,k} = (1-u)^k$. The position vector equation represents a set of three parametric equations for the individual curve co-ordinates :

$$\begin{aligned} x(u) &= \sum_{k=0}^n x_k BEZ_{k,n}(u) \\ y(u) &= \sum_{k=0}^n y_k BEZ_{k,n}(u) \\ z(u) &= \sum_{k=0}^n z_k BEZ_{k,n}(u) \end{aligned} \quad \dots (11.6.4)$$

- The successive binomial coefficients can be calculated as

$$C(n, k) = \frac{n-k+1}{k} C(n, k-1) \quad \dots (11.6.5)$$

Properties of Bezier curve

1. The basis functions are real.
 2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
 3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
 4. The curve generally follows the shape of the defining polygon.
 5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
 6. The curve lies entirely within the convex hull formed by four control points.
 7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
 8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more often than the defining polygon.
 9. The curve is invariant under an affine transformation.
- In cubic Bezier curve four control points are used to specify complete curve.
 - Unlike the B-spline curve, we do not add intermediate points and smoothly extend Bezier curve, but we pick four more points and construct a second curve which can be attached to the first.
 - The second curve can be attached to the first curve smoothly by selecting appropriate control points.

- Fig. 11.6.2 shows the Bezier curve and its four control points. As shown in the Fig. 11.6.2,
- Bezier curve begins at the first control point and ends at the fourth control point. This means that if we want to connect two Bezier curves, we have to make the first control point of the second Bezier curve match the last control point of the first curve.
- We can also observe that at the start of the curve, the curve is tangent to the line connecting first and second control points.
- Similarly at the end of curve, the curve is tangent to the line connecting the third and fourth control point. This means that, to join two Bezier curves smoothly we have to place the third and the fourth control points of the first curve on the same line specified by the first and the second control points of the second curve.
- The Bezier matrix for periodic cubic polynomial is

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\therefore P(u) = U \cdot M_B \cdot G_B$$

where $G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$

and the product $P(u) = U \cdot M_B \cdot G_B$ is

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4$$

- Therefore, the four blending functions for cubic Bezier curve are :

$$BEZ_{0,3}(u) = (1-u)^3$$

$$BEZ_{1,3}(u) = 3u(1-u)^2$$

$$BEZ_{2,3}(u) = 3u^2(1-u)$$

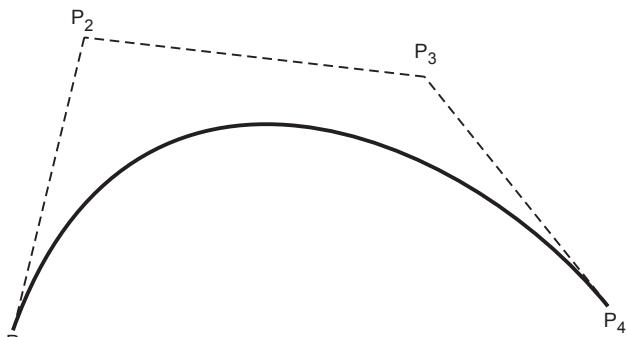


Fig. 11.6.2 A cubic Bezier spline

$$\text{BEZ}_{3,3}(u) = u^3$$

- The four blending functions for cubic Bezier curves, can also obtained by substituting $n = 3$ in equation (11.6.2).

Example 11.6.1 Construct the Bezier curve of order 3 and with 4 polygon vertices $A(1, 1)$, $B(2, 3)$, $C(4, 3)$ and $D(6, 4)$.

Solution : The equation for the Bezier curve is given as

$$P(u) = (1-u)^3 P_1 + 3u(1-u)^2 P_2 + 3u^2(1-u) P_3 + u^3 P_4 \quad \text{for } 0 \leq u \leq 1$$

where $P(u)$ is the point on the curve P_1, P_2, P_3, P_4

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}$

$$\therefore P(0) = P_1 = (1, 1)$$

$$\begin{aligned} \therefore P\left(\frac{1}{4}\right) &= \left(1 - \frac{1}{4}\right)^3 P_1 + 3 \frac{1}{4} \left(1 - \frac{1}{4}\right)^2 P_2 + 3 \left(\frac{1}{4}\right)^2 \left(1 - \frac{1}{4}\right) P_3 + \left(\frac{1}{4}\right)^3 P_4 \\ &= \frac{27}{64}(1,1) + \frac{27}{64}(2,3) + \frac{9}{64}(4,3) + \frac{1}{64}(6,4) \\ &= \left[\frac{27}{64} \times 1 + \frac{27}{64} \times 2 + \frac{9}{64} \times 4 + \frac{1}{64} \times 6, \quad \frac{27}{64} \times 1 + \frac{27}{64} \times 3 + \frac{9}{64} \times 3 + \frac{1}{64} \times 4 \right] \\ &= \left[\frac{123}{64}, \frac{139}{64} \right] = (1.9218, 2.1718) \end{aligned}$$

$$\begin{aligned} \therefore P\left(\frac{1}{2}\right) &= \left(1 - \frac{1}{2}\right)^3 P_1 + 3 \frac{1}{2} \left(1 - \frac{1}{2}\right)^2 P_2 + 3 \left(\frac{1}{2}\right)^2 \left(1 - \frac{1}{2}\right) P_3 + \left(\frac{1}{2}\right)^3 P_4 \\ &= \frac{1}{8}(1,1) + \frac{3}{8}(2,3) + \frac{3}{8}(4,3) + \frac{1}{8}(6,4) \\ &= \left[\frac{1}{8} \times 1 + \frac{3}{8} \times 2 + \frac{3}{8} \times 4 + \frac{1}{8} \times 6, \quad \frac{1}{8} \times 1 + \frac{3}{8} \times 3 + \frac{3}{8} \times 3 + \frac{1}{8} \times 4 \right] \\ &= \left[\frac{25}{8}, \frac{23}{8} \right] = (3.125, 2.875) \end{aligned}$$

$$\begin{aligned} \therefore P\left(\frac{3}{4}\right) &= \left(1 - \frac{3}{4}\right)^3 P_1 + 3 \frac{3}{4} \left(1 - \frac{3}{4}\right)^2 P_2 + 3 \left(\frac{3}{4}\right)^2 \left(1 - \frac{3}{4}\right) P_3 + \left(\frac{3}{4}\right)^3 P_4 \\ &= \frac{1}{64} P_1 + \frac{9}{64} P_2 + \frac{27}{64} P_3 + \frac{27}{64} P_4 \\ &= \frac{1}{64}(1,1) + \frac{9}{64}(2,3) + \frac{27}{64}(4,3) + \frac{27}{64}(6,4) \end{aligned}$$

$$\begin{aligned}
 &= \left[\frac{1}{64} \times 1 + \frac{9}{64} \times 2 + \frac{27}{64} \times 4 + \frac{27}{64} \times 6, \frac{1}{64} \times 1 + \frac{9}{64} \times 3 + \frac{27}{64} \times 3 + \frac{27}{64} \times 4 \right] \\
 &= \left[\frac{289}{64}, \frac{217}{64} \right] = (4.5156, 3.375)
 \end{aligned}$$

$$P(1) = P_3 = (6, 4)$$

The Fig. 11.6.3 shows the calculated points of the Bezier curve and curve passing through it.

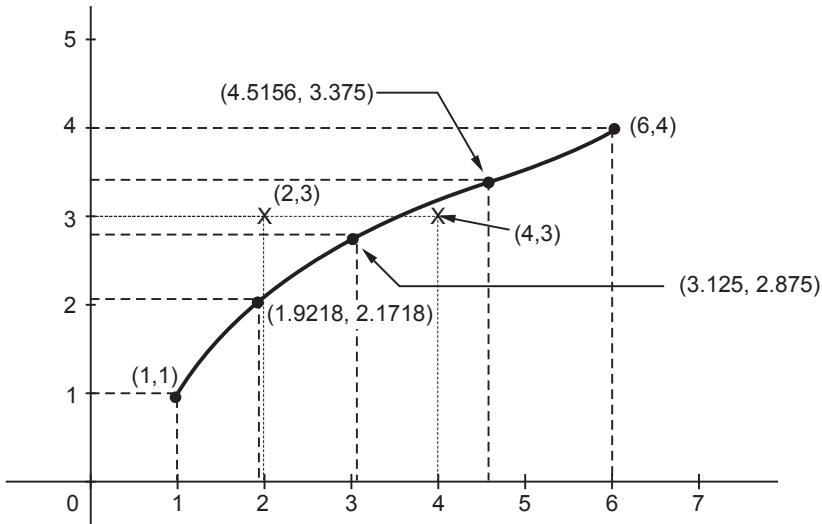


Fig. 11.6.3 Plotted Bezier curved

Example 11.6.2 Set up the equation of Bezier curve and roughly trace it for three control points $(1, 1)$, $(2, 2)$ and $(3, 1)$.

Solution : For $n = 2$, the three blending functions for Bezier curves using equation (11.6.2) are :

$$BE Z_{0,2}(u) = \frac{2!}{0!(2-0)!} \quad u^0(1-u)^{2-0} = (1-u)^2$$

$$BE Z_{1,2}(u) = \frac{2!}{1!(2-1)!} \quad u^1(1-u)^{2-1} = 2u(1-u)$$

$$BE Z_{2,2}(u) = \frac{2!}{2!(2-2)!} \quad u^2(1-u)^{2-2} = u^2$$

Now equation for the Bezier curve is given as

$$P(u) = (1-u)^2 P_1 + 2u(1-u)P_2 + u^2 P_3 \quad \text{for } 1 \leq u \leq 1$$

Let us take $u = 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$

$$\therefore P(0) = P_1 = (1, 1)$$

$$\begin{aligned} P\left(\frac{1}{4}\right) &= \left(1 - \frac{1}{4}\right)^2 P_1 + 2\left(\frac{1}{4}\right)\left(1 - \frac{1}{4}\right)P_2 + \left(\frac{1}{4}\right)^2 P_3 \\ &= \frac{9}{16}(1, 1) + \frac{3}{8}(2, 2) + \frac{1}{16}(3, 1) \\ &= \frac{9}{16} \times 1 + \frac{3}{8} \times 2 + \frac{1}{16} \times 3, \quad \frac{9}{16} \times 1 + \frac{3}{8} \times 2 + \frac{1}{16} \times 1 \\ &= \left[\frac{3}{2}, \frac{11}{8}\right] = (1.5, 1.375) \end{aligned}$$

$$\begin{aligned} P\left(\frac{1}{2}\right) &= \left(1 - \frac{1}{2}\right)^2 P_1 + 2\left(\frac{1}{2}\right)\left(1 - \frac{1}{2}\right)P_2 + \left(\frac{1}{2}\right)^2 P_3 \\ &= 0.25(1, 1) + 0.5(2, 2) + 0.25(3, 1) \\ &= 0.25 \times 1 + 0.5 \times 2 + 0.25 \times 3, 0.25 \times 1 + 0.5 \times 2 + 0.25 \times 1 \\ &= (2, 1.5) \end{aligned}$$

$$\begin{aligned} P\left(\frac{3}{4}\right) &= \left(1 - \frac{3}{4}\right)^2 P_1 + 2\left(\frac{3}{4}\right)\left(1 - \frac{3}{4}\right)P_2 + \left(\frac{3}{4}\right)^2 P_3 \\ &= 0.0625(1, 1) + 0.375(2, 2) + 0.5625(3, 1) \\ &= 0.0625 \times 1 + 0.375 \times 2 + 0.5625 \times 3, 0.0625 \times 1 + 0.375 \times 2 + 0.5625 \times 1 \\ &= (2.5, 1.375) \end{aligned}$$

$$P(1) = P_2 = (3, 1)$$

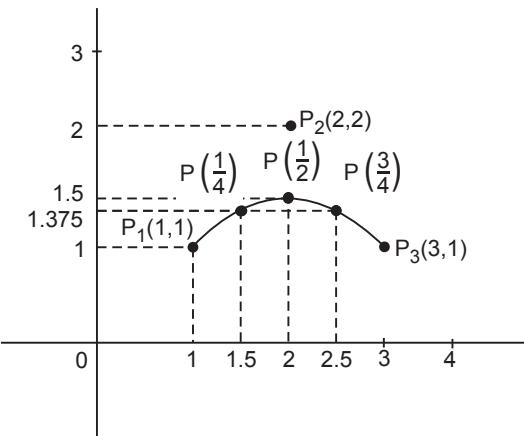


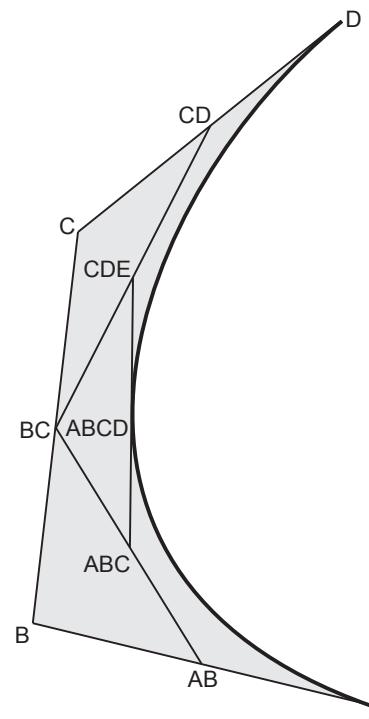
Fig. 11.6.4 Plotted Bezier curve

Examples for Practice

Example 11.6.3 : Given the vertices of Bezier polygon as : $P_0(1,1)$, $P_1(2,3)$, $P_2(4,3)$ and $P_3(3,1)$, determine five points on Bezier curves.

Example 11.6.4 : Given a set four 2D control points $(0, 1)$, $(2, 5)$, $(5, 5)$, $(8, 0)$ determine four points on Bezier curve.

- Another approach to construct the Bezier curve is called **midpoint approach**.
- In this approach the Bezier curve can be constructed simply by taking midpoints.
- In midpoint approach midpoints of the lines connecting four control points (A, B, C, D) are determined (AB, BC, CD).
- These midpoints are connected by line segments and their midpoints ABC and BCD are determined.
- Finally these two midpoints are connected by line segments and its midpoint ABCD is determined. This is illustrated in Fig. 11.6.5.
- The point ABCD on the Bezier curve divides the original curve into two sections. This makes the points A, AB, ABC and ABCD are the control points for the first section and the points ABCD, CDE, CD and D are the control points for the second section.
- By considering two sections separately we **Fig. 11.6.5 Subdivision of a Bezier spline** can get two more sections for each separate section i.e. the original Bezier curve gets divided into four different curves. This process can be repeated to split the curve into smaller sections until we have sections so short that they can be replaced by straight lines or even until the sections are not bigger than individual pixels.



Algorithm

1. Get four control points say A (x_A, y_A), B (x_B, y_B), C (x_C, y_C), D (x_D, y_D)
2. Divide the curve represented by points A, B, C and D in two sections

$$x_{AB} = (x_A + x_B) / 2$$

$$y_{AB} = (y_A + y_B) / 2$$

$$\begin{aligned}
 x_{BC} &= (x_B + x_C) / 2 \\
 y_{BC} &= (y_B + y_C) / 2 \\
 x_{CD} &= (x_C + x_D) / 2 \\
 y_{CD} &= (y_C + y_D) / 2 \\
 x_{ABC} &= (x_{AB} + x_{BC}) / 2 \\
 y_{ABC} &= (y_{AB} + y_{BC}) / 2 \\
 x_{BCD} &= (x_{BC} + x_{CD}) / 2 \\
 y_{BCD} &= (y_{BC} + y_{CD}) / 2 \\
 x_{ABCD} &= (x_{ABC} + x_{BCD}) / 2 \\
 y_{ABCD} &= (y_{ABC} + y_{BCD}) / 2
 \end{aligned}$$

3. Repeat the step 2 for section A, AB, ABC and ABCD and section ABCD, BCD, CD and D.
4. Repeat step 3 until we have sections so short that they can be replaced by straight lines.
5. Replace small sections by straight lines.
6. Stop

Review Questions

1. Explain bezier curve with properties. SPPU : May-14, Dec.-15, Marks 6
2. What are the properties of Bezier curve ? SPPU : May-05, 06,17,19, Dec.-08, Marks 3
3. Describe the procedure to generate Bezier curve. SPPU : May-05, Dec.-05, Marks 3
4. Derive blending function of Bezier curve. SPPU : May-05, Dec.-05, Marks 6
5. Write a note on Bezier curve. SPPU : May-07, 10, 13, Dec.-12, Marks 4
6. How blending function is calculated for cubic polynomial curve ? SPPU : May-10, Marks 4
7. Write short note on blending functions of Bezier curve. SPPU : Dec.-12,18, May-18, Marks 6
8. List various methods for drawing curved lines. Write a short note (with diagram) on Bezier curve. Write necessary blending function. SPPU : May-11, Marks 10
9. Define Bezier curve. State its properties. Derive blending function of Bezier curve. SPPU : Dec.-14, Marks 8

11.7 B-Spline Curve

SPPU : May-05,06,07,08,10,11,13,15,17,18,19, Dec.-05,06,10,11,12,16,17

Limitations of Bezier curve

- The Bezier curve produced by the Bernstein basis function has a limited flexibility.

- First the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve. For example, polygon with four vertices results a cubic polynomial curve.
- The only way to reduce the degree of the curve is to reduce the number of vertices, and conversely the only way to increase the degree of the curve is to increase the number of vertices.
- The second limiting characteristics is that the value of the blending function is nonzero for all parameter values over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change with in a curve.

B-spline

- There is another basis function, called the **B-spline** basis, which contains the Bernstein basis as a special case.
- The B-spline basis is nonglobal. It is nonglobal because each vertex B_i is associated with a unique basis function. Thus, each vertex affects the shape of the curve only over a range of parameter values where its associated basis function is nonzero.
- The B-spline basis also allows the order of the basis function and hence the degree of the resulting curve is independent on the number of vertices.
- It is possible to change the degree of the resulting curve without changing the number of vertices (control points) of the defining polygon.
- If $P(u)$ be the position vectors along the curve as a function of the parameter u , a B-spline curve is given by

$$P(u) = \sum_{i=1}^{n+1} B_i N_{i,k}(u) \quad u_{\min} \leq u < u_{\max}, \quad 2 \leq k \leq n+1 \quad \dots(11.7.1)$$

where the B_i are the position vectors of the $n + 1$ defining polygon vertices and the $N_{i,k}$ are the normalized B-spline basis functions. For the i^{th} normalized B-spline basis function of order k , the basis function $N_{i,k}(u)$ are defined as

$$N_{i,1}(u) = \begin{cases} 1 & \text{if } x_i \leq u < x_{i+1} \\ 0 & \text{Otherwise} \end{cases}$$

$$\text{and } N_{i,k}(u) = \frac{(u-x_1)N_{i,k-1}(u)}{x_{i+k-1}-x_i} + \frac{(x_{i+k}-u)N_{i+1,k-1}(u)}{x_{i+k}-x_{i+1}} \quad \dots(11.7.2)$$

- The values of x_i are the elements of a knot vector satisfying the relation $x_i \leq x_{i+1}$. The parameter u varies from u_{\min} to u_{\max} along the curve $P(u)$.

- The choice of knot vector has a significant influence on the B-spline basis functions $N_{i,k}(u)$ and hence on the resulting B-spline curve.
- There are three types of knot vector : **uniform**, **open uniform** and **nonuniform**.
- In a uniform knot vector, individual knot values are evenly spaced. For example,
 $[0 \ 1 \ 2 \ 3 \ 4]$

For a given order k , uniform knot vectors give periodic uniform basis functions for which

$$N_{i,k}(u) = N_{i-1,k}(u-1) = N_{i+1,k}(u+1)$$

- An open uniform knot vector has multiplicity of knot values at the ends equal to the order k of the B-spline basis function. Internal knot values are evenly spaced. Examples are,

$$k = 2 [0 \ 0 \ 1 \ 2 \ 3 \ 3]$$

$$k = 3 [0 \ 0 \ 0 \ 1 \ 2 \ 3 \ 3 \ 3]$$

$$k = 4 [0 \ 0 \ 0 \ 0 \ 1 \ 2 \ 2 \ 2 \ 2]$$

- Generally, an open uniform knot vector is given by,

$$x_i = 0 \quad 1 \leq i \leq k$$

$$x_i = i - k \quad k+1 \leq i \leq n+1$$

$$x_i = n - k + 2 \quad n+2 \leq i \leq n+k+1 \quad \dots (11.7.3)$$

- The curves resulted by the use of open uniform basis function are nearly like Bezier curves. In fact, when the number of defining polygon vertices is equal to the order of the B-spline basis and an open uniform knot vector is used, the B-spline basis reduces to the Bernstein basis. Hence, the resulting B-spline curve is a Bezier curve.

Example 11.7.1 Calculate the four third-order basis function $N_{i,3}(u)$, $i = 1, 2, 3, 4$ with an open uniform knot vector.

Solution : We have to calculate four basis functions, therefore $n = (4 - 1) = 3$ and it is of order three, therefore $k = 3$. From equation (11.5.3) the open uniform knot vector is given as

$$[X] = [0 \ 0 \ 0 \ 1 \ 2 \ 2 \ 2]$$

Now, from equation (11.7.2), the basis functions for various parameters are as follows :

$$0 \leq u < 1$$

$$N_{3,1}(u) = 1; \quad N_{i,1}(u) = 0, \quad i \neq 3$$

$$N_{2,2}(u) = 1 - u; \quad N_{3,2}(u) = u, \quad N_{i,2}(u) = 0 \quad i \neq 2, 3$$

$$N_{1,3}(u) = (1-u)^2; \quad N_{2,3}(u) = u(1-u) + \frac{(2-u)}{2}u$$

$$N_{3,3}(u) = \frac{u^2}{2}; \quad N_{i,3}(u) = 0; i \neq 1, 2, 3$$

1 ≤ u < 2

$$N_{4,1}(u) = 1; \quad N_{i,1}(u) = 0, \quad i \neq 4$$

$$N_{3,2}(u) = (2-u); \quad N_{4,2}(u) = (u-1); \quad N_{i,2}(u) = 0, \quad i \neq 3, 4$$

$$N_{2,3}(u) = \frac{(2-u)^2}{2}; \quad N_{3,3}(t) = \frac{u(2-u)}{2} + (2-u)(u-1);$$

$$N_{4,3}(u) = (u-1)^2; \quad N_{i,3}(u) = 0; \quad i \neq 2, 3, 4$$

Example 11.7.2 Construct the B-spline curve of order 4 and with 4 polygon vertices A(1, 1), B(2, 3), C(4, 3) and D(6, 2).

Solution : Here n = 3 and k = 4 from equation (11.7.3) we have open uniform knot vector as

x = [0 0 0 0 1 1 1 1] and from equation (11.7.2) we have basis functions are

0 ≤ u < 1

$$N_{4,1}(u) = 1; \quad N_{i,1}(u) = 0, \quad i \neq 4$$

$$N_{3,2}(u) = (1-u); \quad N_{4,2}(u) = u, \quad N_{i,2}(u) = 0, \quad i \neq 3, 4$$

$$N_{2,3}(u) = (1-u)^2; \quad N_{3,3}(u) = 24(1-u);$$

$$N_{4,3}(u) = u^2; \quad N_{i,3}(u) = 0; \quad i \neq 2, 3, 4$$

$$N_{1,4}(u) = (1-u)^3; \quad N_{2,4}(u) = u(1-u)^2 + 2u(1-u)^2 = 3u(1-u)^2;$$

$$N_{3,4}(u) = 2u^2(1-u) + (1-u)u^2 = 3u^2(1-u); \quad N_{4,4}(u) = u^3$$

Using equation (11.7.1) the parametric B-spline is

$$P(u) = AN_{1,4}(u) + BN_{2,4}(u) + CN_{3,4}(u) + DN_{4,4}(u)$$

$$\therefore P(u) = (1-u)^3 A + 3u(1-u)^2 B + 3u^2(1-u)C + u^3D$$

Let us take u = 0, $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, 1

$$\therefore P(0) = A = [1, 1]$$

$$\begin{aligned}
 \therefore P\left(\frac{1}{4}\right) &= \left(1 - \frac{1}{4}\right)^3 A + 3 \frac{1}{4} \left(1 - \frac{1}{4}\right)^2 B + 3 \left(\frac{1}{4}\right)^2 \left(1 - \frac{1}{4}\right) C + \left(\frac{1}{4}\right)^3 D \\
 &= \left(\frac{27}{64}\right) A + \left(\frac{27}{64}\right) B + \left(\frac{9}{64}\right) C + \left(\frac{1}{64}\right) D \\
 &= \left[\frac{27}{64}(1, 1) + \frac{27}{64}(2, 3) + \frac{9}{64}(4, 3) + \frac{1}{64}(6, 2)\right] \\
 &= \left[\frac{27}{64} \times 1 + \frac{27}{64} \times 2 + \frac{9}{64} \times 4 + \frac{1}{64} \times 6, \frac{27}{64} \times 1 + \frac{27}{64} \times 3 + \frac{9}{64} \times 3 + \frac{1}{64} \times 2\right] \\
 &= \left[\frac{123}{64}, \frac{137}{64}\right] = [1.9218, 2.14]
 \end{aligned}$$

$$\begin{aligned}
 \therefore P\left(\frac{1}{2}\right) &= \left(1 - \frac{1}{2}\right)^3 A + 3 \frac{1}{2} \left(1 - \frac{1}{2}\right)^2 B + 3 \left(\frac{1}{2}\right)^2 \left(1 - \frac{1}{2}\right) C + \left(\frac{1}{2}\right)^3 D \\
 &= \frac{1}{8} A + \frac{3}{8} B + \frac{3}{8} C + \frac{1}{8} D = \left[\frac{1}{8}(1, 1) + \frac{3}{8}(2, 3) + \frac{3}{8}(4, 3) + \frac{1}{8}(6, 2)\right] \\
 &= \left[\frac{1}{8} \times 1 + \frac{3}{8} \times 2 + \frac{3}{8} \times 4 + \frac{1}{8} \times 6, \frac{1}{8} \times 1 + \frac{3}{8} \times 3 + \frac{3}{8} \times 3 + \frac{1}{8} \times 2\right] \\
 &= \left[\frac{25}{8}, \frac{21}{8}\right] = [3.125, 2.625]
 \end{aligned}$$

$$\begin{aligned}
 \therefore P\left(\frac{3}{4}\right) &= \left(1 - \frac{3}{4}\right)^3 A + 3 \frac{3}{4} \left(1 - \frac{3}{4}\right)^2 B + 3 \left(\frac{3}{4}\right)^2 \left(1 - \frac{3}{4}\right) C + \left(\frac{3}{4}\right)^3 D \\
 &= \frac{1}{64} A + \frac{9}{64} B + \frac{27}{64} C + \frac{27}{64} D = \frac{1}{64}(1, 1) + \frac{9}{64}(2, 3) + \frac{27}{64}(4, 3) + \frac{27}{64}(6, 2) \\
 &= \left[\frac{1}{64} \times 1 + \frac{9}{64} \times 2 + \frac{27}{64} \times 4 + \frac{27}{64} \times 6, \frac{1}{64} \times 1 + \frac{9}{64} \times 3 + \frac{27}{64} \times 3 + \frac{27}{64} \times 2\right] \\
 &= \left[\frac{289}{64}, \frac{163}{64}\right] = [4.5156, 2.5468]
 \end{aligned}$$

$$\therefore P(1) = D = [6, 2]$$

The Fig. 11.7.1 shows the calculated points of the B-spline curve and curve passing through it.

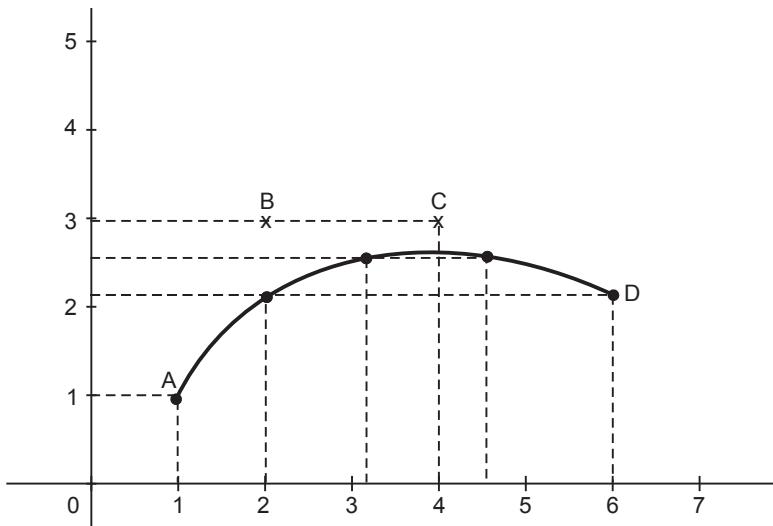


Fig. 11.7.1 Plotted B-spline curve

Properties of B-spline curve

- The sum of the B-spline basis functions for any parameter value u is 1.
i.e. $\sum_{i=1}^{n+1} N_{i,k}(u) = 1$
- Each basis function is positive or zero for all parameter values, i.e., $N_{i,k} \geq 0$.
- Except for $k = 1$ each basis function has precisely one maximum value.
- The maximum order of the curve is equal to the number of control points of defining polygon.
- The curve exhibits the variation diminishing property. Thus the curve does not oscillate about any straight line more often than its defining polygon.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

Advantages of B-splines over Bezier curves

- The degree of B-spline polynomial is independent on the number of control points of defining polygon (with certain limitations).
- B-spline allows local control over the curve surface because each control point affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.

Comparison between Bezier and B-spline curves

Sr. No.	Bezier curve	B-spline curve
1.	The degree of the polynomial defining the curve segment is one less than the number of defining polygon point (Control points). Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.	The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
2.	Bezier curve can't be forced to interpolate any of its n control points without repeating it.	B-splines can be forced to interpolate any of its n control points without repeating it.
3.	Bezier curve requires less computation.	B-spline curve requires more computation.
4.	Bezier curves are less flexible.	B-Spline curve offers more control and flexibility.
5.	Bezier curves require less information.	B-Spline curves require more information such as degree of the curve and a knot vector.
6.	In Bezier curves, it is not possible to use lower degree curves and still maintain a large number of control points.	In B-Spline curves, it is possible to use lower degree curves and still maintain a large number of control points.

Review Questions

1. Explain B-splines curves. SPPU : May-10,11,13,17,18,19, Dec.-12,16, Marks 8
2. Explain the technique of smoothing the curves used B-spline. SPPU : May-05, 07, Marks 6
3. Explain the procedure to generate B-spline curve. SPPU : Dec.-05, May-15, Marks 4
4. Explain B-spline curve functions for generating curves. SPPU : Dec.-05, 11, Marks 4
5. Discuss the properties of B-spline curve. SPPU : May-06, Marks 5
6. State advantages of B-spline curve over Bezier for generating curve. SPPU : Dec.-06,16, Marks 4
7. Compare Bezier and B-spline curves. SPPU : May-08, Dec.-10,19, Marks 8
8. Explain blending function for B - spline curve. SPPU : Dec.-17, Marks 4

11.8 Fractals

SPPU : May-05,06,08,09,13,14,15,17,18,19, Dec.-05,07,08,10,12,15,16,17,18,19

- So far we have discussed how to draw geometrical figures such as lines, polygons, circles and smooth curved surfaces. However, in nature we come across rough, jagged, random surfaces and edges when we try to draw mountains, trees, rivers and so on.

- Such rough, jagged and random surfaces are called **fractals**. Let us see the concept involved in drawing fractal images.

11.8.1 Classification of Fractals

- The fractals can be classified as
 - Self similar
 - Self affine and
 - Invariant

Self similar fractals

- These fractals have parts those are scaled-down versions of the entire object.
- In these fractals object subparts are constructed by applying a scaling parameter s to the overall initial shape. It is a choice of user to use the same scaling factor s for all subparts, or use different scaling factors for different scaled-down parts of the object.
- Another sub class of self similar fractals is a statistically self-similar fractals, in which user can also apply random variations to the scaled-down subparts. These fractals are commonly used to model trees, shrubs, and other plants.

Self-affine fractals

- These fractals have parts those are formed with different scaling parameters, s_x, s_y, s_z in different co-ordinate directions.
- In these fractals, we can also apply random variations to obtain statistically self-affine fractals.
- These fractals are commonly used to model water, clouds and terrain.

Invarient fractals

- In these fractals, **nonlinear transformation** is used.
- It includes **self squaring fractals** such as the Mandelbrot set, which are formed with squaring functions in complex space, and self inverse fractals, form with inversion procedures.

11.8.2 Topological Dimension

- Consider an object composed of elastic or clay. If the object can be deformed into a line or line segment we assign its dimension $D_t = 1$. If object deforms into a plane or half plane or disk we assign its dimension $D_t = 2$. If object deforms into all space or half space or sphere, we assign its dimension $D_t = 3$. The dimension D_t is referred to as the **topological dimension**.

11.8.3 Fractal Dimension

- It is the second measure of an object dimension. Imagine that a line segment of length L is divided into N identical pieces. The length of each line segment l can be given as

$$l = \frac{L}{N}$$

- The ratio of length of original line segment and the length of each part of the line segment is referred to as scaling factor and is given as

$$S = \frac{L}{l}$$

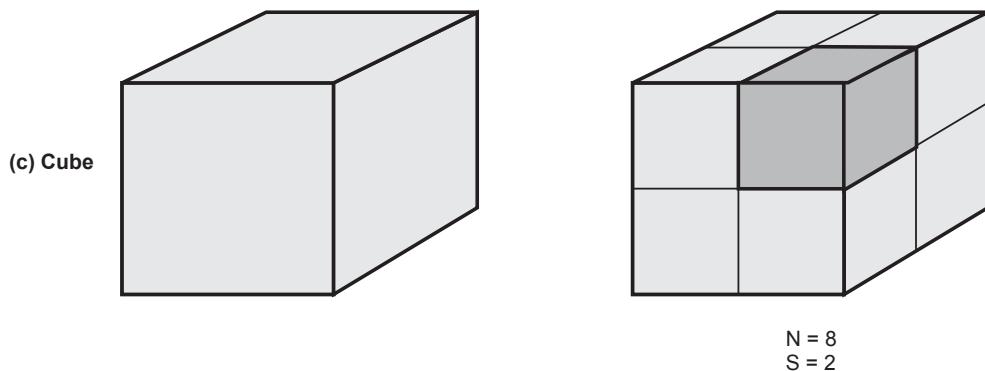
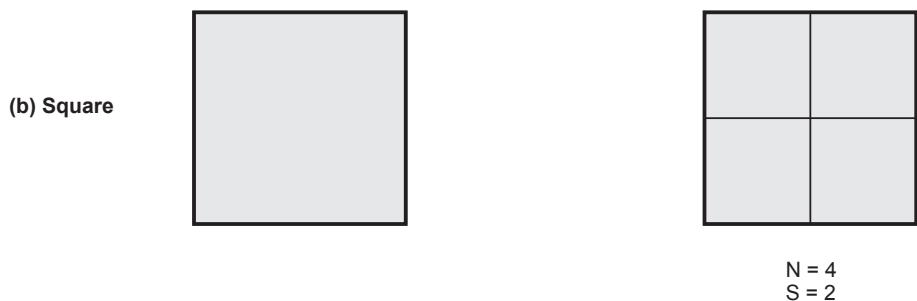
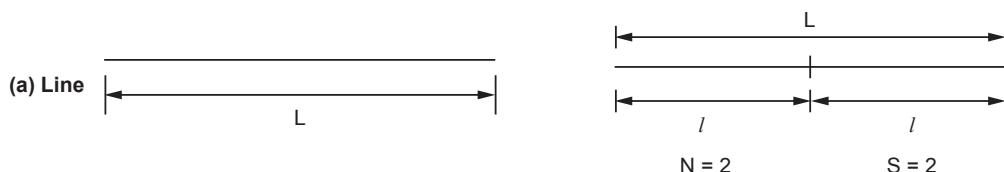


Fig. 11.8.1 Scaling of objects in various dimensions

- From above two equations we can write

$$N = s$$

i.e. $N = s^1$

- In other words we can say that if we scale a line segment by a factor $1/s$ then we have to add N pieces together to get the original line segment. If we scale square object by a factor $1/s$ we will get a small square. In case of $s = 2$, we require 4 pieces of square to get original square. In general we can write

$$N = s^2$$

- Similarly for cubical object we have, (Refer Fig. 11.8.1)

$$N = s^3$$

- We have seen that we can specify the dimension of the object by variable D . Here the exponent of s is a measure of object dimension. Thus we can write

$$N = s^D$$

Solving for D we get

$$D = \log N / \log s$$

- This D is called **fractal dimension**.

11.8.4 Hilbert's Curve

- The Hilbert's curve can be constructed by following successive approximations. If a square is divided into four quadrants we can draw the first approximation to the Hilbert's curve by connecting centre points of each quadrant as shown in Fig. 11.8.2.

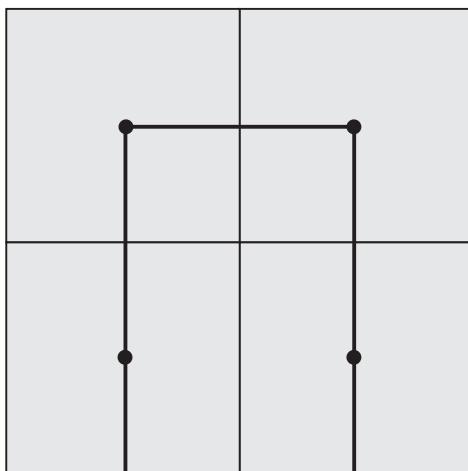


Fig. 11.8.2 The first approximation to Hilbert's curve

- The second approximation to the Hilbert's curve can be drawn by further subdividing each of the quadrants and connecting their centres before moving to next major quadrant (see Fig. 11.8.3).

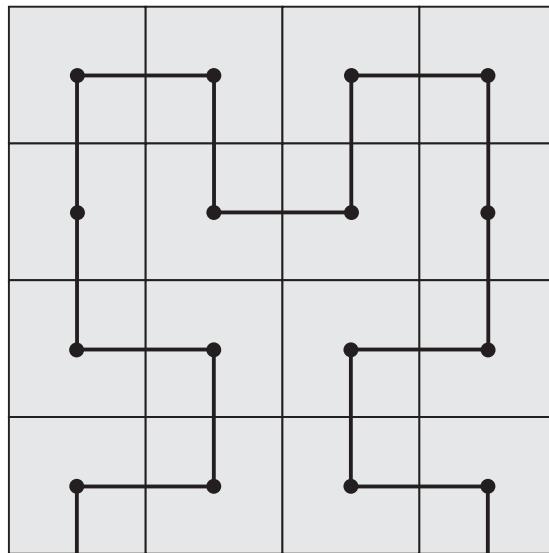


Fig. 11.8.3 Second approximation to Hilbert's curve

- The third approximation subdivides the quadrants again. We can draw third approximation to Hilbert's curve by connecting the centres of finest level of quadrants before stepping to the next level of the quadrant (see Fig. 11.8.4).

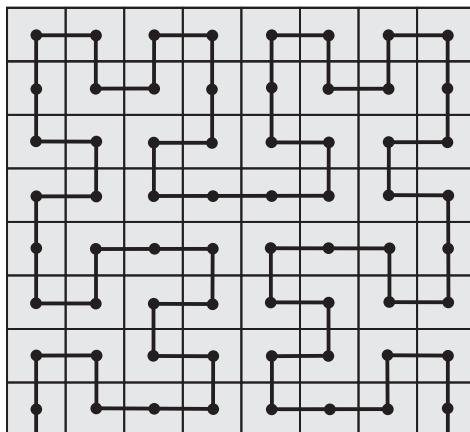


Fig. 11.8.4 Third approximation to Hilbert's curve

From the above three figures we can easily note following points about Hilbert's curve.

1. If we infinitely extend the approximations to the Hilbert's curve, the curve fills the smaller quadrants but never crosses itself.
2. The curve is arbitrarily close to every point in the square.
3. The curve passes through a point on a grid, which becomes twice as fine with each subdivision.
4. There is no limit to subdivisions and therefore length of curve is infinite.
5. With each subdivision length of curve increases by factor of 4.
6. At each subdivision the scale changes by 2 but length changes by 4 therefore for Hilbert's curve topological dimension is one but the fractal dimension is 2.

Application

Because of this locality property, the Hilbert curve is widely used in computer science. For example, the range of IP addresses used by computers can be mapped into a picture using the Hilbert curve. Code to generate the image would map from 2D to 1D to find the color of each pixel and the Hilbert curve is sometimes used because it keeps nearby IP addresses close to each other in the picture.

11.8.5 Koch Snow Flake Curve/Triadic Curve

- The koch snow flake curve is also known as koch curve.
- The Koch curve can be drawn by dividing line into 4 equal segments with scaling factor $1/3$ and middle two segments are so adjusted that they form adjacent sides of an equilateral triangle as shown in the Fig. 11.8.5. This is the first approximation to the koch curve.
- To apply the second approximation to the Koch curve we have to repeat the above process for each of the four segments. The resultant curve is shown in Fig. 11.8.6.
- The koch curve shown in Fig. 11.8.6 is more specifically known as the **triadic koch curve**. The triadic koch curve's name steps from the fact that the middle thirds of the line segments are modified at each step.



Fig. 11.8.5 First approximation of the Koch curve

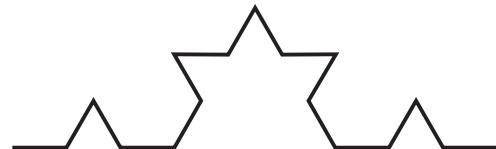


Fig. 11.8.6 The second approximation to Koch curve

- The resultant curve has more wiggles and its length is $16/9$ times the original length.
- From the figures we can easily note following points about the koch curve :
 - Each repetition increases the length of the curve by factor $4/3$.
 - Length of curve is infinite.
 - Unlike Hilbert's curve, it doesn't fill an area.
 - It doesn't deviate much from its original shape.
 - If we reduce the scale of the curve by 3, we find the curve that looks just like the original one; but we must assemble 4 such curves to make the original, so we have

$$4 = 3^D$$

Solving for D we get

$$D = \log_3 4 = \log 4 / \log 3 \approx 1.2618$$

- Therefore for koch curve topological dimension is 1 but fractal dimension is 1.2618.
- From the above discussion we can say that point sets, curves and surfaces which give a fractal dimension greater than the topological dimension are called **fractals**. The Hilbert's curve and koch curves are fractals, because their fractal dimensions (respectively, 2 and 1.2618) are greater than their topological dimension which is 1.

Review Questions

- | | |
|---------------------------------------------------------------------|------------------------------------------------------------|
| 1. What is fractal ? | SPPU : May-18, Dec.-18, Marks 2 |
| 2. What do you mean by topological and fractal dimension ? | SPPU : May-05, 06, 08, 09, Dec.-05, 07, 10, Marks 6 |
| 3. Define fractals ? Explain Hilbert curve and Koch curve. | SPPU : May-14,17, Marks 7 |
| 4. Define fractal and give any two examples of fractal. | SPPU : May-08, 09, Dec.-08, 15 Marks 8 |
| 5. Explain Hilbert's curve and give its fractal dimension. | SPPU : May-06, 15 Dec.-08, Marks 6 |
| 6. Explain koch curve in detail giving the fractal dimension. | SPPU : Dec.-07,17,19, May-08, 09,18, Marks 6 |
| 7. Write short note on Fractals. | SPPU : Dec.-12, May-13, Marks 6 |
| 8. What is fractals ? Explain any two applications of the fractals. | SPPU : Dec.-16, Marks 5 |
| 9. Write short notes on Hilbert's curve. | SPPU : Dec.-10,17,18 Marks 6 |
| 10. Explain Hilbert Curve and its application in detail. | SPPU : May-18, Marks 4 |

11. Explain Koch curve and its application in detail.
12. Explain Koch curve and Hilbert curve with example.
13. What is fractal ? Explain characteristics and classification of fractals.

SPPU : Dec.-18, Marks 4

SPPU : May-19, Marks 6

SPPU : Dec.-19, Marks 6



Notes

UNIT - VI

12

Segment

Syllabus

Introduction, Segment table, Segment creation, closing, deleting and renaming, Visibility.

Contents

12.1	<i>Introduction</i>	May-05,06,07,08,	
		Dec.-05,11, 14,15	Marks 2
12.2	<i>Segment Table</i>	May-05,06,07,08,12,13,	
		Dec.-05,11,15,18	Marks 4
12.3	<i>Functions for Segmenting the Display File</i>	May-05,06,07,08,10,11,12,13,14,15,19	
		Dec.-06,10,11, 14,17,19, ..	Marks 10
12.4	<i>More about Segments</i>	Dec.-06, May-15	Marks 4
12.5	<i>Display File Structures</i>	Dec.-16,17,	Marks 5
12.6	<i>Image Transformation</i>	May-11, Dec.-10,	Marks 8

12.1 Introduction**SPPU : May-05,06,07,08, Dec.-05,11, 14,15**

- In practice, the image on the display screen is often composed of several pictures or items of information.
- An image may contain several views of an object and related information.
- It may also contain close up view of a particular component. For example, we may wish to display an internal plan of a living room. The plan may contain various objects such as sofa-set, T.V., show-case, teapot, show-pieces, wall hangings and so on.
- Each object has a set of attributes such as size, colour and its position in the room.
- We might wish to see all these objects simultaneously or a single object at a time.
- To view the entire image or a part of the image with various attributes we need to organize the image information in a particular manner.
- The image information is stored in the display file.
- Existing structure of the display file does not satisfy our requirements of viewing the image. Hence the display structure is modified to reflect the subpicture structure. To achieve this display file is divided into **segments**.
- Each segment corresponds to a component or an object of the overall display and is associated with a set of attributes.
- Along with the attributes the segment is also associated with the image transformation parameters such as scaling along X and/or Y-direction, rotation and shearing.
- Therefore, presence of segment allows :
 - Subdivision of the picture.
 - Visualization of a particular part of the picture.
 - Scaling, rotation and translation of a particular part of the picture.

In this chapter, we will discuss the segmentation of display file along with the creation, renaming, closing and deletion of segments.

Review Questions1. *What is segment ?***SPPU : May-05, 06, 07, 08, Dec.-05, 11, 14, 15, Marks 2**2. *Why do we need segments ?***SPPU : Dec.11, Marks 2****12.2 Segment Table****SPPU : May-05,06,07,08,12,13, Dec.-05,11,15,18**

- To access a particular segment and the information associated with it we must have a unique name assigned to each segment.

- Along with the name we must have its display file position and its attribute information.
- The structure used to organize all this information related to segments is called **segment table**.

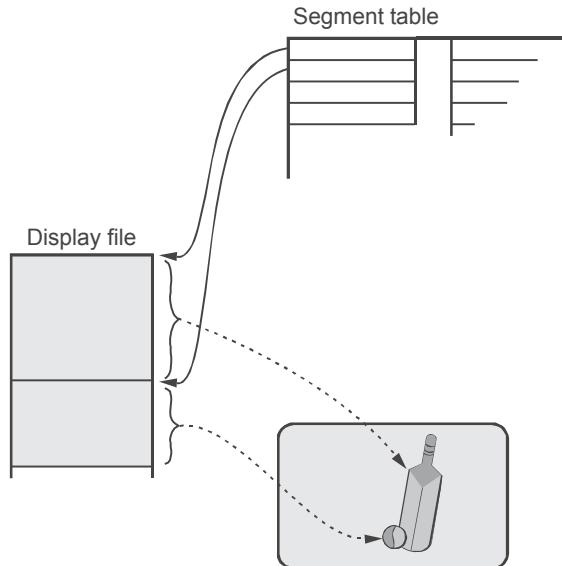


Fig. 12.2.1 Segment table and display file

- It indicates the portions of display file used to construct the picture. The segment table is formed by using arrays.
- First array holds the display file starting location for that segment, the second array holds the segment size information, while the third indicates the visibility and so on. This is illustrated in Fig. 12.2.2.

Segment no	Segment start	Segment size	Scale x	Scale y	Colour	Visibility
0						
1						
2						
3						
4						
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Fig. 12.2.2 Segment table

- Each row in the segment table represents information of one segment including its name, position, size, attributes and the image information parameters.

- If we wish to make segment 4 visible then this is achieved by setting the corresponding entry in the array to 'ON'.
- The display file interpreter initially checks the start, size and visible attribute of the segments and it interprets only those segments which are to be made visible.
- There are other possible schemes for implementing the segment table, many with the substantial advantages over array scheme. Here, we have chosen array scheme because it allows simple accessing, does not require any new data structure and its updating is straight forward.
- An alternative approach is the linked list. The linked list uses the additional field called the link or pointer which gives the location of the segment in the segment table.
- In case of arrays, maximum number of segments that can be included in the segment table are equal to the length of arrays. But with linked list there is no such limit on the maximum number of segments; the growth of a linked list is dynamic.
- Fig. 12.2.3 shows the segment table using linked list.

Segment number	Segment start	Segment size	Scale x	Scale y	Colour	Visibility	Link
1							3
2							4
3							2
4							5
5							Null /

Fig. 12.2.3 Segment table using linked list

- In case of hidden surface removal it is necessary to display different segments in specific order.
- Ordering/sorting of segments in the segment table is quite easy task with linked list structure.
- The ordering can be achieved by simply adjusting the links. However, in case of arrays we have to move actual segment information in the process of sorting the segments.
- In the display file there are few instructions which did not specify any segment. Such instructions are placed in a special "unnamed" segment.
- Normally index 0 of the array is assigned to unnamed segment. Therefore, segment-size [0] indicates the number of instructions in the unnamed segment.

Review Questions

1. Write the algorithm for change of visibility attribute of segments.

SPPU : Dec.-11, Marks 3

2. Explain a segment table with example.

SPPU : May-05,06,07,08,12,13, Dec.-05,15,18, Marks 4

12.3 Functions for Segmenting the Display File

SPPU : May-05,06,07,08,10,11,12,13,14,15,19, Dec.-06,10,11, 14,17,19

- We are familiar with the common methods of creating and modifying sequential disk files. We open a file before we add data to it, and we close the file when we have added the last data item and the file is complete. To change the contents of a file, we open it again, add the new data to replace the old, and close the new file. To get rid of a file, we delete it.
- The very same operations are used for manipulating display file segments.
- To create a new segment, we create it and then call graphic primitives to add to the segment the lines and text to be displayed, then we close the segment.
- To modify the contents we rename current segment with the modified segment and to remove segment from the display file, we delete it.
- Thus we have four basic functions.
 - Create segment
 - Close segment
 - Delete segment
 - Rename segment

12.3.1 Segment Creation

- In a segment creation process, initially, we have to check whether some other segment is still open.
- It is not allowed to open two segments at a same time because it is then difficult to assign the drawing instructions to particular segment. Hence, segment must be created or opened when no other segment is currently open.
- We must give the segment a name so that we can identify it. While doing this it is important to check whether a given segment name is valid or not and whether there already exists a segment with the same name. If so, we have to assign other valid name to the segment.
- Once a valid segment name is assigned, we have to initialize the items in the segment table under our segment name to indicate that this is a fresh new segment.
- The first instruction of this segment will be located at the next free storage area in the display file.
- As we have not entered any instructions into the segment yet, its size is initialized with value zero.

- The attributes of the segment are initialized as a default attribute values.

Algorithm : Create Segment

1. Check whether any segment is open; if so display error message : "Segment is still open" and go to step 9.
2. Read the name of new segment.
3. Check whether the new segment name is valid; if not display error message : "Not a valid segment name"and go to step 9.
4. Check whether the new segment name is already existing in the same-name list; if so display error message : "Segment name already exists" and go to step 9.
5. Initialize the start of the segment at the next free storage area in the display file.
6. Initialize size of this segment equal to zero.
7. Initialize all attributes of segment to their default values.
8. Indicate that the new segment is now open.
9. Stop.

12.3.2 Closing a Segment

- Once a segment is open we can enter the display file instructions in it. The entered commands are then associated with the open segment.
- After completion of entering all display file instructions, the segment must be closed.
- To close a segment it is necessary to change the name of the currently open segment. It can be achieved by changing the name of currently open segment as 0. Now the segment with name 0 is open i.e. unnamed segment is open.
- If there are two unnamed segments in the display file one has to be deleted.

Algorithm : Close Segment

1. Check whether any segment is open; if no, display error message : " No segment open" and go to step 6.
2. Change the name of currently open segment, say 0, i.e., unnamed segment.
3. Delete any unnamed segment instructions which may have been saved, i.e. initialize the unnamed segment with no instructions.
4. Initialize the start of the unnamed segment at the next free storage area in the display file.
5. Initialize size of unnamed segment equal to 0.
6. Stop.

12.3.3 Deleting a Segment

- When we want to delete a particular segment from the display file then we must recover the storage space occupied by its instructions and make this space free for some other segment. To do this we must not destroy and re-form the entire display file, but we must delete just one segment, while preserving the rest of the display file. The method to achieve this depends upon the data structure used to represent the display file.
- Here, we have used arrays to store the display file information.
- Use of arrays makes the recovery of storage space occupied by the segment very easy and straight forward. However using arrays is not as efficient as some other storage techniques such as link list.
- In case of arrays the gap left by deleted block is filled by shifting up all segments which are following the deleted segment. This is illustrated in Fig. 12.3.1.

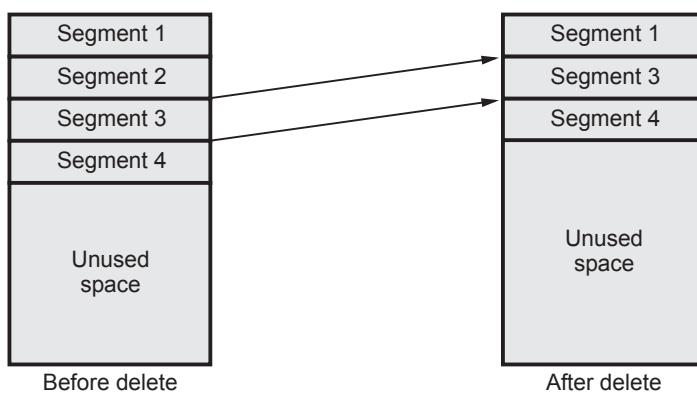


Fig. 12.3.1 Display file contents before and after deleting segment 2

Algorithm : Delete Segment

- Read the name of segment which is to be deleted.
- Check whether that segment name is valid; if not, display error message "Segment not valid" and go to step 8.
- Check whether the segment is open, if yes, display error message "Can't delete the open segment" and go to step 8.
- Check whether the size of segment is greater than 0; if no, no processing is required, as segment contains no instructions. Therefore, go to step 8.
- Shift the display file elements which follow the segment which is to be deleted by its size.
- Recover the deleted space by resetting the index of the next free instruction.

7. Adjust the starting positions of the shifted segments by subtracting the size of the deleted segment from it.
8. Stop.
 - When we want to start a completely new picture, we have to delete all of the segments.
 - One way of deleting all segments is to delete them individually. This approach is independent of the data structure used for the display file.
 - A more efficient approach for our array data structure is to simply set the size value of all segments to 0 and initialize the next free space as the first location in the display file.

12.3.4 Renaming a Segment

- We can display animated character on the display by presenting sequence of images, each with a slightly different drawing of the character.
- Let us assume that we have a segment in a display file for animated character. Then to display a new image in the sequence we have to delete the current segment and re-create it with the altered character. The problem in this process is that during the time after the first image is deleted and time before the second image is completely entered, only a partially completed character is displayed on the screen. We can avoid this problem by keeping the next image ready in the display file before deleting the current segment.
- This means that both segments, the segment which is to be deleted and the segment which is to be replaced with must exist in the display file at the same time.
- This can be achieved by creating a new invisible image under some temporary segment name.
- Now, when the current segment is deleted we can make the new image visible and rename it with the name of deleted segment.
- These steps can be repeated to achieve the animation.
- The idea of storing two images, one to show and other to create or alter, is called **double buffering**.

Algorithm : Rename Segment

1. Check whether both old and new segment names are valid; if not, display error message "Not valid segment names" and go to step 6.
2. Check whether any of the two segments are open.

If open, display error message "Segments still open" and go to step 6.

3. Check whether the new name we are going to give to the old segment is not already existing in the display file. If yes, display error message "Segment already exists" and go to step 6.
4. Copy the old segment table entry into the new position.
5. Delete the old segment.
6. Stop.

Review Questions

1. Write the algorithm for the following :

SPPU : Dec.-11, Marks 5

i) Delete a segment

ii) Delete all segments.

2. Explain a delete operation.

SPPU : May-05, Marks 4

3. Explain operations performed on segment table.

SPPU : May-06,07,08,12,13,14,15, Dec.-11,14, Marks 6

4. Explain with examples the functions needed to maintain a segmented display file.

SPPU : Dec.-06, Marks 4

5. Explain with illustration how segments are created, renamed and deleted.

SPPU : May-10, 11, Dec.- 10 Marks 10,

6. Describe segment and explain any three operations carried out on it.

SPPU : Dec.-17, Marks 4

7. Write an algorithm to rename a segment. Draw a sample segment table.

SPPU : May-19, Marks 4

8. Write algorithms to create a segment and delete a segment.

SPPU : Dec.-19, Marks 6

12.4 More about Segments

SPPU : Dec.-06, May-15

- Whenever we want to display any picture or image on the screen, we have to create segments for various parts of the picture, and we have to set attributes for them.
- By setting the visible attribute of the segment we can decide whether to display it or not.
- When visible attribute of a segment is 1 it is included in the active segment list and display file interpreter processes it. Such a segment is called **posted segment**.
- When visible attribute of a segment is 0 it is not included in the active segment list and display file interpreter skips this segment. Such a segment is called **unposted segment**.

- The visible attribute of a segment decides the posting or unposting of a segment.

Advantages of using segmented display files

1. Segmentation allows to organize display files in subpicture structure.
2. It allows to apply different set of attributes to the different portions of the image.
3. Due to segmentation selective portion of the image can be displayed.
4. Segmentation makes it easier to modify the picture by changing segment attributes or by replacing segments.
5. Segmentation allows application of transformation on selective portion of the image.

Review Questions

1. *What do you mean by posting and unposting of segments ?*
2. *State the advantages of segmented display files.*

SPPU : Dec.-06, May-15, Marks 4

12.5 Display File Structures

SPPU : Dec.-16,17

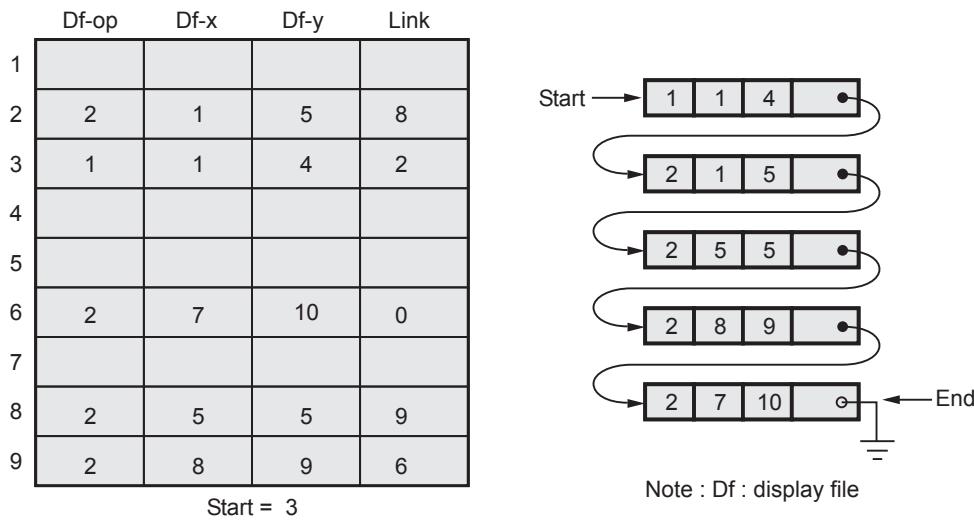
Various data structures are used to implement segment table. These are :

Array Structure

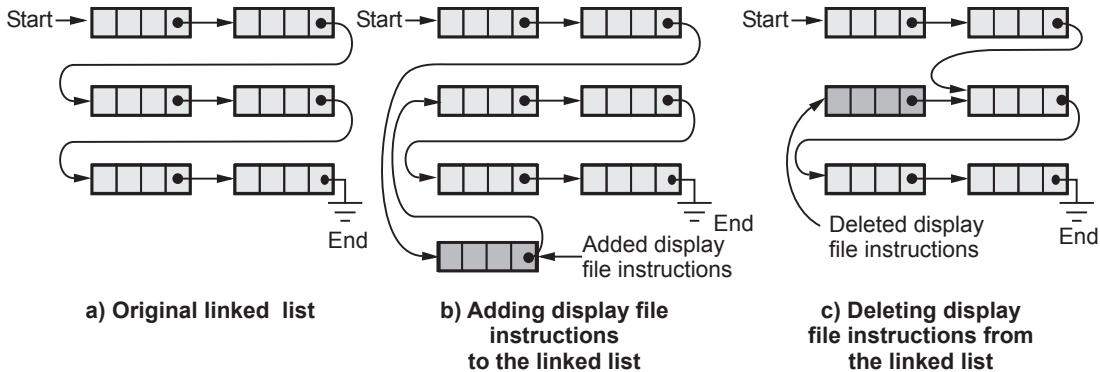
- The display file we have used is a simple array data structure. But array data structure is not much efficient.
- If we wish to remove an instruction at the beginning of the display file, we have to move all succeeding instructions.
- If the display file is large, a lot of processing is required to recover only a small amount of storage.

Linked List

- In a linked list the instructions are not stored in a sequence; rather a new field is added to the instruction. This field, called the **link** or **pointer**, gives the location of the next instruction.
- Here, we can step through the instructions by following the chain of links. This is illustrated in the Fig. 12.5.1.
- When we want to add an instruction to a display file, a free cell is obtained from the list, the correct instruction operation code and operands are stored and the cell is linked to the display file list.

**Fig. 12.5.1 Display file as linked list**

- Deletion of cells from a linked list is very easy. To delete a cell, we need only change the pointer which points to that cell so that it points to the succeeding cell. This is illustrated in Fig. 12.5.2.

**Fig. 12.5.2**

Paging Scheme

- The paging scheme is a combination of array and linked list structure.
- In this method the display file is organized into a number of small arrays called **pages**.
- The pages are linked to form a linked list of pages.
- Each segment starts at the beginning of a page.
- If segment ends before the page boundary, the remaining page is not used.
- In this method, display file instructions can be accessed within a page just as they were accessed in an array.

- When the end of a page is reached, a link is followed to find the next page. This is illustrated in Fig. 12.5.3.

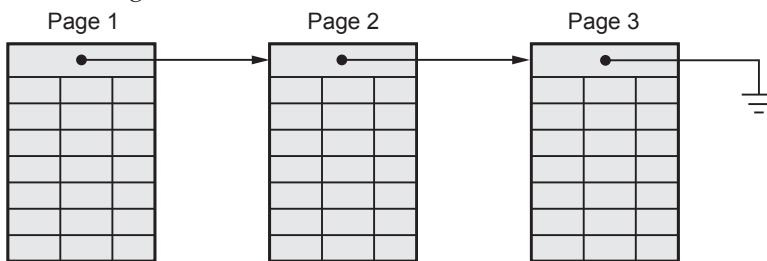


Fig. 12.5.3 Paging scheme for display file

Review Questions

1. Explain a segment table with an example along with data structure used to implement the segment table. SPPU : Dec.-16, Marks 5

2. Explain the following term with example : Display file structure. SPPU : Dec.-17, Marks 2

12.6 Image Transformation

SPPU : May-11, Dec.-10

- One of the major advantage of computer graphics is that we can display pictures with certain alterations if required. For example, we can see image from a different angle, we can zoom particular part of the image, we can change the size of the image, we can change the position of particular part of the image.
- These changes are easy to perform because the graphics image information is stored in different segments and we can apply changes to different segments independently.
- The Fig. 12.6.1 shows the original image and the images after applying some form of transformation.

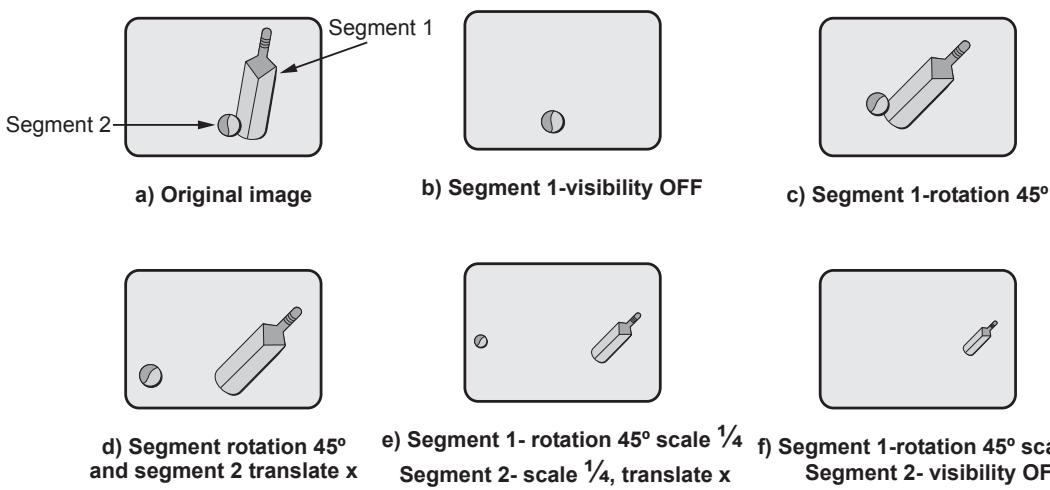


Fig. 12.6.1

- The changes in the original graphics information can be done by performing mathematical operations which are called **transformations**.

Algorithm : Setting Image Transformation

1. Read the name of segment, say s_name.
2. Check whether the name is valid or not. If not display error message " Not a valid segment " and go to step 8.
3. Read translation factor, say t_x and t_y .
4. Read the scaling factors, say s_x and s_y .
5. Read the angle of rotation, say θ .
6. Set various transformation parameters for the segment s_name.

Translate_x [s_name] = t_x ,

Translate_y [s_name] = t_y ,

Scale_x [s_name] = s_x ,

Scale_y [s_name] = s_y ,

Rotate [s_name] = θ .

7. Check the visibility of the segment s_name. If visible then create a new frame for it.
8. Stop.

Review Questions

1. Write a short note on image transformation.
2. Explain, how segment operations play major role in image presentation and processing.

SPPU : May-11, Marks 8

3. Discuss the concept of segmentation used in cricket animation with suitable example. Assume your animation is having at least 3 to 4 segments in it.

SPPU : Dec.-10, Marks 8



Notes

UNIT - VI

13

Animation

Syllabus

Introduction, Conventional and computer based animation, Design of animation sequences, Animation languages, Key-frame, Morphing, Motion specification.

Contents

13.1 <i>Introduction</i>	May-17, 19, Dec.-16, 17,	Marks 2
13.2 <i>Conventional and Computer Based Animation</i>		
13.3 <i>Design of Animation Sequences</i>	Dec.-17, May-18,	Marks 3
13.4 <i>Animation Languages</i>	Dec.11, 12,	Marks 8
13.5 <i>Keyframes and Morphing</i>	May-17, 18, 19, Dec.-17,	Marks 3
13.6 <i>Motion Specification</i>	Dec.-19,	Marks 3

13.1 Introduction

SPPU : May-17,19, Dec.-16,17

- Computer animation is the art of creating moving images via the use of computers. It is a subfield of computer graphics and animation.
- Increasingly it is created by means of 3D computer graphics, though 2D computer graphics are still widely used for low bandwidth and faster real-time rendering needs.
- The common applications of computer animations are entertainment (computer games, motion pictures and cartoons) advertising, scientific and engineering studies and training and education.
- In computer animation to create the illusion of movement, an image is displayed on the computer screen, then quickly replaced by a new image that is similar to the previous image, but slightly shifted. This technique is identical to how the illusion of movement is achieved with television and motion pictures.

Review Questions

1. *What is computer animation ? How is it created ?*
2. *What is animation ?*

SPPU : May-17,19, Dec.-16,17, Marks 2

13.2 Conventional and Computer Based Animation

Animated movies means the movies generated by animating drawn pictures. It is not the movie with actors in real-world scenes. Walt Disney movies are the good examples of animated movies. Now-a-days there are countless animated cartoons and commercial advertisements on TV. The newest, fastest and economical tool for creating such animation movies is a computer with its graphic capabilities.

13.2.1 Conventional Animation

In this section we will discuss the steps required for creating conventional animation. To study these sequential steps, some concepts should be cleared, which are explained below.

- **Storyboard :** With the help of sequence of sketches the ideas and of structure animation is created. This outline form of animation is called a 'storyboard'.
- **Keyframes :** These are the frames in animation in which the entities being animated are at extreme position.
- **Inbetween :** Once the key-frames of the animation are drawn, the intermediate positions can be inferred. Filling the intermediate frames in key-frames is called 'inbetween'.

- **Pencil test** : With key-frames and inbetween a trial film is made, which is called a 'pencil test'.
- **Cels** : A common technique is to decompose an animation picture into several parts that can move more or less independently. These parts are drawn individually on a clear plastic material that are called **cels** (Short for cellulose).

The individual cels can be overlaid to produce the whole picture and then filmed or video recorded as one frame. This allows us to make sequence of picture frames with changes in the picture by appropriately moving the individual cels.

- **Route-sheet** : The sheet which describes each scene and the people responsible for the various aspects of producing the scene is called 'route-sheet'.
- **Exposure sheet** : The sheet which is an immensely detailed description of animation, is called an 'exposure sheet'. The exposure sheet gives details for each frame such as the order of all the figures in the frame, dialogue description, choice of background and the camera position within the frame.

The sequence of steps needed for creating a conventional animation are explained below.

1. The story of animation is written and a storyboard is created.
2. If any sound track is to be recorded, according to the drawing for every scene in the animation, a detailed layout is produced.
3. The instants at which significant sounds occur are recorded in order. Then the detailed layout and the sound track are correlated.
4. Certain key-frames of the animation are drawn.
5. With key-frames and inbetween, pencil test frames are obtained.
6. The pencil test frames are then transferred to cels. This is done either by hand copying in ink or by photocopying directly onto the cels.
7. The cels are coloured in or painted and are arranged in a correct sequence. Then they are filmed.

[**Note :** In multiplane animation, some layers of cels are used for background and some for foreground characters. The background cels remains constant except for a translation. Foreground cels change over the time].

Since this animation uses key-frames and inbetween, it is also called 'key-frame animation'.

From all above discussion, we can say that the organizational process of an animation can be described with the help of storyboard, route sheet and exposure sheet.

13.2.2 Computer based Animation

Computer animation can be created with a computer and animation software. Some examples of animation software are : Amorphium, Art of illusion, 3D studio max, Adobe flash and many more. These software provide basic functions to create an animation and to process individual objects. The functions are also available to store and manipulate object database, motion generation and object rendering. In object data base object shapes and associated parameters are stored and they are updated as we proceed animation.

Let us see the basic steps needed in computer generated animation.

1. **Digitization** : Before using the computer for animation, the drawings, decomposed parts must be digitized. This can be done using methods such as,

Producing original drawings by animation paint systems.

Using optical scanning.

By tracking the drawings with a data tablet.

The resulting drawings may need further processing such as filtering to clean up the glitches (if any). It also smooths the contours.

2. **Composition** : In this stage, the individual key frames are generated with proper combination of cels. The foreground and background figures are also combined using image-composition techniques to prepare final key-frames.

3. **In-between frames** : Animation software allows user to generate in-between frames. These are the intermediate frames between the keyframes. Usually there are 3 to 5 in-between frames between two keyframes. Suppose we want to design an animation sequence for 10 seconds. For this we need to have 24×10 frames because film requires 24 frames per second. Out of these 240 frames we can have 48 keyframes and remaining 192 in-between frames. In this case there are four in-between frames between two keyframes.

4. **Equivalent of pencil test** : A rectangular array is taken and several small low-resolution frames of an animation are placed in it. In a frame buffer, a particular portion of such a low resolution image, typically one twenty-fifth or one thirty-sixth is taken. This portion is moved to the center of the screen which is called **panning**. Then using **zooming** technique this portion is enlarged so as to fill the entire screen. The same procedure is repeated on the several films of the animation which are stored in an array for the single image. If these steps are carried out fast enough, the effect of continuity is observed. Since each frame of the animation is reduced to a very small part of the total image (typically one twenty-fifth or one thirty-sixth), and is then expanded to fill the screen, this process effectively lowers the display device's resolution.

Review Questions

1. *What do you mean by computer assisted animation ?*
2. *What is conventional animation ?*
3. *Explain the steps needed for creating a conventional animation.*
4. *What is panning ?*

13.3 Design of Animation Sequences**SPPU : Dec.-17, May-18**

- The steps for designing animation sequences are as follows :
 - Storyboard layout
 - Object definitions
 - Keyframe specifications
 - Generation of in-between frames
- **Storyboard layout :** The storyboard is an outline of the action. It defines a set of basic events that are to take place in a specific order. Such an ordered set of events gives the motion sequence. Usually storyboard consists of rough sketches or it could be a list of the basic ideas for the motion.
- **Object definition :** Each active section of the scene is treated as an object. It is defined in terms of basic shapes, such as polygons or splines. Along with the object the motion associated with the object is also defined.
- **Keyframe specification :** A keyframe is a detailed drawing of the scene at a certain time in the animation sequence. They are positioned according to the time for that frame. Usually, some keyframes represent the extreme positions in the action and others are spaced so that the time interval between keyframes is not too great.
- **In-between frames :** These are the intermediate frames between the keyframes. Usually there are 3 to 5 in-between frames between two keyframes. Suppose we want to design an animation sequence for 10 seconds. For this we need to have 24×10 frames because film requires 24 frames per second. Out of these 240 frames we can have 48 keyframes and remaining 192 in-between frames. In this case there are four in-between frames between two keyframes.

Review Questions

1. *Explain the steps for designing animation sequences.*
2. *Define key frames.*

SPPU : May-18, Marks 3**SPPU : Dec.-17, May-18, Mark 1**

13.4 Animation Languages

Dec.11, 12

- There are many different languages for describing computer animation, and new ones are constantly being developed. These animation languages can be categorized as :
 - Linear-list notations
 - General-purpose languages with embedded animation directives.
 - Graphical languages

13.4.1 Linear-List Notations

- In linear-list notations, each event in the animation is described by a starting and ending frame number and an action that is to take place.
- The action is specified by a statement with relative parameters. For example,
30, 45, C ROTATE "ARM", 1, 60
- The above statement say that "between frames 30 and 45, rotate the object called ARM about axis 1 by 60 °, determining the amount of rotation at each frame from table C. Since the statements describe individual actions and have frame values associated with them, their order is, for the most part, irrelevant.
- If two actions are applied to the same object at the same time, their order is important. Different order may give different results.

13.4.2 General-Purpose Languages

- General purpose languages such as C, C++, Pascal or Lisp can be used to design and control the animation sequences.
- The values of variables in these languages can be used as parameter to whatever routines actually generate animations.
- These languages have great potential.
- They can certainly do everything that linear-list notations do; however, most of them require considerable programming expertise on the part of the user.
- ASAS is an example of general purpose language. It is built on top of LISP. Its primitives include vectors, colours, polygons, solids, groups (collection of objects), points of view, subworlds, and lights.
- It also includes a wide range of geometric transformations that operate on objects.
- They take an object as an argument and return a value that is a transformed copy of the object.

- These transformations include up, down, left, right, zoom-in, zoom-out, forward and backward.
- The advantages of ASAS over linear-list notations is the ability to generate procedural objects and animations with in the language.

13.4.3 Graphical Languages

- There are several specialized animation languages developed called **graphical languages**.
- These languages provide various animation functions which make it easy to design and control the animation.
- The animation functions include a graphics editor, a keyframe generator, an in-between generator, and standard graphics routines. Thus, specialized animation language allows us to design and modify object shapes, position the object and light source, define the photometric parameters such as light-source intensities and surface-illumination properties, and setting the camera parameters such as position, orientation and lens characteristics.
- Apart from this animation languages support action specific functions. These functions generate the layout of motion paths for objects and camera.

Keyframe systems : This is a specialized animation language used to generate in-between frame from the user-specified keyframes. In this the object is created using set of rigid bodies connected at the joints. The set of rigid bodies are provided with limited degrees of freedom. By proper controlling the movements of object bodies keyframe systems generate in-between frames.

Parameterized systems : This language specifies the object motion characteristics as a part of the object definition. Due to this the designer can have a control over object characteristics such as degree of freedom, motion limitations, and allowable shape changes.

Scripting systems : This language allows object specifications and animation sequences to be defined with a user-input script. It is possible to write a script for defining object and its motion. Such scripts can be stored in the library.

Review Questions

1. Write a note on animation languages.
2. Give different types of animation languages.

SPPU : Dec.-12, Marks 8

SPPU : Dec.-11, Marks 2

13.5 Keyframes and Morphing

SPPU : May-17,18,19, Dec.-17

- We know that, we have to generate in-between frames from the specified two or more keyframes. These in-between frames are generated with knowledge of motion that object is going to carry out. We can specify motion details as follows :
 - Motion paths can be specified with a kinematic description as a set of spline curves or
 - The forces acting on the objects can be specified.
- In case of complex scene, the frames are divided into individual components or objects called **cells** (**celluloid transparencies**) Then in-between frames of individual objects are generated using interpolation.
- Many times, we use complex transformations, in which shape of object may change. For example, in object like cloth, facial features the shape of the object may change. If such objects are specified with polygon meshes, then the number of edges per polygon can change from one frame to the next.
- The, total number of line segments may be different in different frames.

13.5.1 Morphing

- Morphing is a special effect in motion pictures and animations that changes one image into another through a seamless transition.
- It is a transformation of object shapes from one form to another.
- When object is described using polygon, the two keyframes for which in-between frames are to be generated are compared.
- The keyframes are compared in terms of number of vertices, number of edges and number of line segments. If they are unequal they are added or deleted to match the count as a preprocessing steps. This is illustrated in the Fig. 13.5.1. The Fig. 13.5.1 shows two keyframes, K and K + 1. The frame K has one line segment while frame K + 1 has two line segments. Since keyframe K + 1 has an extra vertex, we add a vertex between vertices 1 and 2 in keyframe K to balance the number of vertices in two frames as a preprocessing step. Then using linear interpolation we can translate the added vertex in keyframe K into vertex 3' along the straight line path. The intermediate position of added vertex gives us the in-between frames as shown in the Fig. 13.5.1.

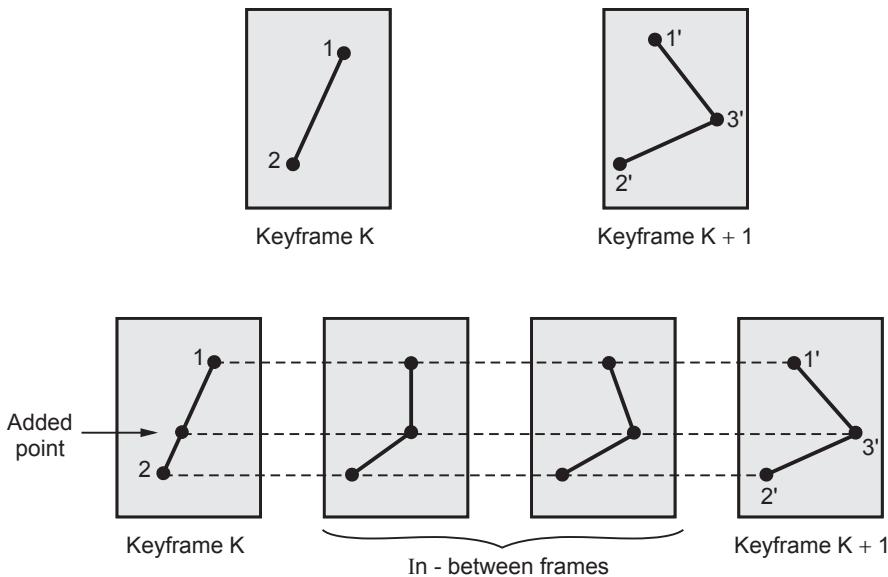


Fig. 13.5.1 Generation of in-between frames using linear interpolation

- Fig. 13.5.2 shows transformation of triangle into quadrilateral using linear interpolation.

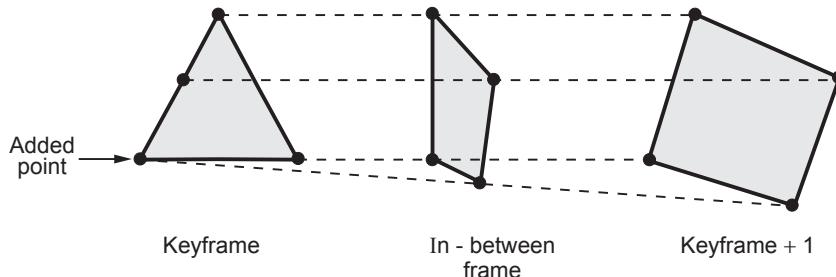


Fig. 13.5.2 Linear interpolation for transforming a triangle into a quadrilateral

- In general, to equalize two frames we can define

$$L_{\max} = \max(L_K, L_{K+1}), L_{\min} = \min(L_K, L_{K+1})$$

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

$$N_S = \text{int}\left(\frac{L_{\max}}{L_{\min}}\right)$$

where L represents line segment.

With these definitions, the preprocessing is accomplished by

- Dividing N_e edges of L_{\min} into $N_S + 1$ sections.
- Dividing the remaining lines of L_{\min} into N_S sections.

- As an example, if $L_K = 3$ and $L_{K+1} = 11$, then $n_e = 11 \bmod 3 = 2$ and $N_S = \text{int}(11/3) = 3$. Therefore, the preprocessing is accomplished by dividing 2 edges of L_K into 4 sections and dividing the remaining line of L_K into 3 sections. This is illustrated in Fig. 13.5.3.

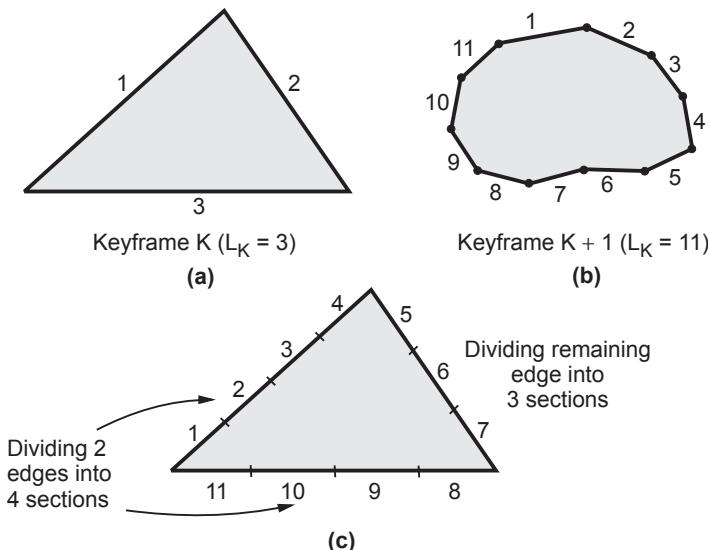


Fig. 13.5.3 General rules for preprocessing

- Like line segment count, we can equalize the vertex count. Here, we use V_K and V_{K+1} parameters to denote the number of vertices in the two consecutive frames. In this case, we define

$$V_{\max} = \max(V_K, V_{K+1}), \quad V_{\min} = \min(V_K, V_{K+1})$$

and $N_{1S} = (V_{\max} - 1) \bmod (V_{\min} - 1)$

$$N_P = \text{int}\left(\frac{V_{\max} - 1}{V_{\min} - 1}\right)$$

With these definitions, the preprocessing is accomplished by

1. Adding N_P points to N_{1S} line sections of keyframe_{min}.
2. Adding $N_P - 1$ points to the remaining edges of keyframe_{min}.

Simulating Acceleration

- In the previous section we have seen the linear paths to fit the positions in the in-between frame. We can also use non-linear path for this purpose.
- To use non-linear paths various curve fitting techniques are used to specify the animation paths between keyframes. This is illustrated in Fig. 13.5.4.

- The position or motion of the object between two or more keyframes changes continuously. If this change is at constant speed we say that the acceleration is zero. This is achieved by using equal-interval time spacing for the in-between frames as shown in the Fig. 13.5.5. For example, if there are n in-between frames with a time interval t ($t_2 - t_1$) between keyframes, then time interval between in-between frames can be given by

$$\Delta t = \frac{t}{n+1} = \frac{t_2 - t_1}{n+1}$$

Now, we can calculate the time for any in-between frame as

$$t_{Bi} = t_1 + i\Delta t$$

where i is the i^{th} in-between frame.

$$\text{Here, } n = 5 \quad \therefore \Delta t = \frac{t_2 - t_1}{6}$$

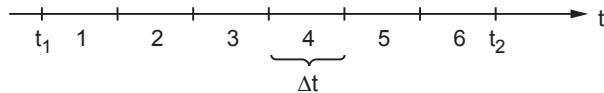


Fig. 13.5.5 Equal Δt in in-between frames to achieve constant speed

- We can produce the realistic display of speed changes using non-zero acceleration.
- We can vary the speed at different portions of the animation path with spline or trigonometric functions.
- We can use parabolic and cubic time functions for this purpose. However, the trigonometric functions are more commonly used in animation packages.
- To model increasing speed, i.e. positive acceleration, we want the time spacing between frames to increase so that greater changes in position occurs as we proceed. Thus by generating in-between frames with increase in time spacing and displaying them with equal time spacing we can see the increasing speed effect. In this, the initial frames have less changes in object positions while later frames have more changes in object positions.
- We can obtain an increasing time interval size with the function

$$1 - \cos\theta, \quad 0 < \theta < \pi / 2$$

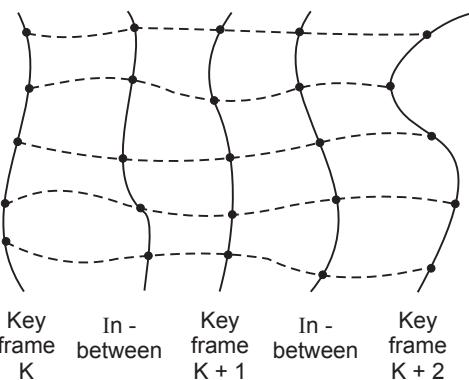


Fig. 13.5.4 Fitting keyframe using non-linear path

- For n in-between frames, the time for the i^{th} in-between frame is given by

$$t_{Bi} = t_1 + t \left[1 - \frac{\cos i\pi}{2(n-1)} \right]$$

where i is the i^{th} in-between frame and t is the time between two keyframes.

- Fig. 13.5.6 shows the plot of the trigonometric acceleration function and the in-between spacing for $n = 5$.

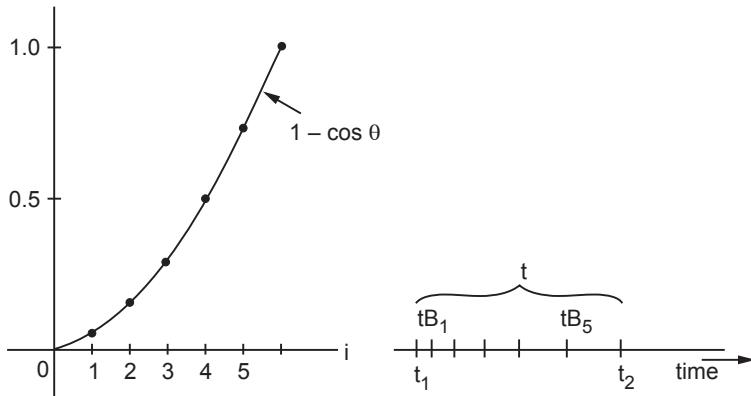


Fig. 13.5.6 A trigonometric acceleration function and the corresponding in-between spacing for $n = 5$

- Similarly, we can model decreasing speed, i.e. deceleration with $\sin \theta$ in the range of $0 < \theta < \pi/2$. In this case, the time for the i^{th} in-between frame is given by

$$t_{Bi} = t_1 + t \sin \frac{i\pi}{2(n-1)}$$

where i is the i^{th} in-between frame and t is the time between two keyframes. Fig. 13.5.7 shows plot of this function and the decreasing size of the time intervals.

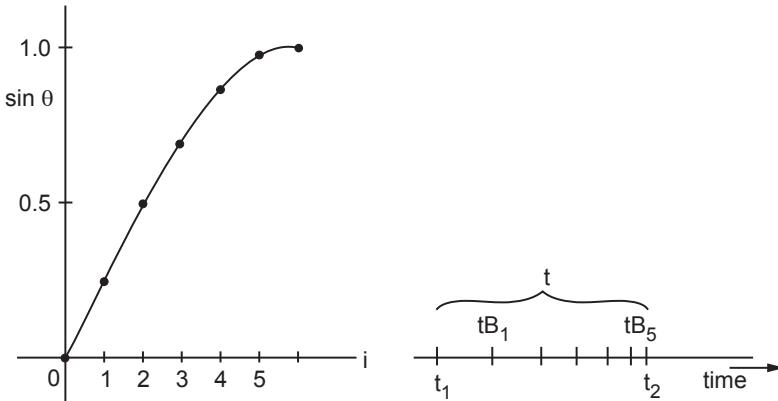


Fig. 13.5.7 Trigonometric deceleration function and the corresponding in-between spacing for $n = 5$

- We can also model a combination of increasing and decreasing speed by first increasing in in-between frame time spacing and then decreasing in in-between frame time spacing. We can obtain such a time interval with the function

$$\frac{1}{2}(1 - \cos \theta), \quad 0 < \theta < \pi / 2$$

For n in-between frames, the time for the i^{th} in-between frame is given by

$$t_{Bi} = t_1 + t \left\{ \frac{1 - \cos[i\pi / (n+1)]}{2} \right\}$$

where i is the i^{th} in-between frame and t is the time between two keyframes. Time intervals for the moving object initially increase, then the time intervals decrease, as shown in the Fig. 13.5.8.

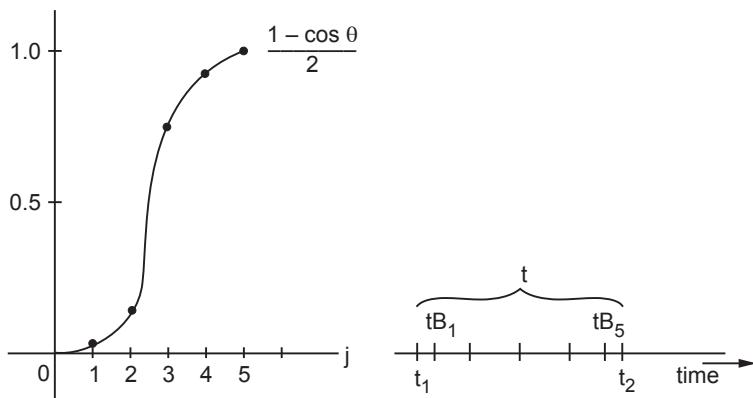


Fig. 13.5.8 A trigonometric accelerate-decelerate function and the corresponding in-between spacing for $n = 5$

13.5.2 Tweening

- Tweening is a short for in-betweening. It is the process of generating intermediate frames between two images to give the appearance that the first image evolves smoothly into the second image.
- It is an interpolation technique where an animation program generates extra frames between the key frames that the user has created. This gives smoother animation without the user having to draw every frame.
- In tweening a scene is described by a mathematical model - a set of two- or three-dimensional objects whose positions are given by sets of coordinates. It uses mathematical formulae to generate these coordinates at a sequence of discrete times.
- The simplest system would move each point at a constant rate in a straight line between its initial and final positions, though other kinds of path are possible. The

coordinates at each time step are used to generate (or "render") a two-dimensional image of the scene which forms one "frame" of the animation.

- Tweening is similar to morphing except that morphing is usually performed by interpolating between corresponding points marked by the user on two images, rather than between two configurations of a model.

Review Questions

1. What is morphing ?

SPPU : May-17,19, Dec.-17, Marks 2

2. Write short notes on

a) Key frame systems b) Morphing c) Simulating acceleration.

3. Explain morphing in detail.

SPPU : May-18, Marks 3

4. What do you mean by morphing ? Explain with example how it is used in animation along with necessary mathematical treatment.

5. What is tweening ?

6. Distinguish between raster animation and key frame animation in detail.

13.6 Motion Specification

SPPU : Dec.-19

- The various ways in which the motions of objects can be specified. These are :
 - Direct motion specification
 - Goal-directed systems
 - Kinematics and Dynamics
- **Direct motion specification**
 - It is most straightforward method for defining a motion sequence.
 - In this method, the rotation angles and the translation vectors are specified so that the geometrical transformations can be applied to the objects in the scene to generate animation sequences.
- **Goal-directed systems**
 - In these systems, instead of specifying motion parameters, action goal specific instructions are specified. Thus these systems are known as goal directed systems.
 - For example, we could specify that we want an object to **walk** or to run to a particular destination.
- **Kinematics and Dynamics**
 - We can also construct animation sequences using kinematic or dynamic descriptions.

- In case of kinematic descriptions, motion parameters such as position, velocity and acceleration are specified without reference to the forces that causes the motion to generate animation sequences.
- An alternative approach is to use **inverse kinematics**, in which, initial and final positions of objects at specified times are specified and from that motion parameters are computed by system to generate animation sequences.
- In dynamic descriptions, the forces that produce the velocities and accelerations are specified. Such descriptions of objects are referred to as a **physically based modeling**.
- Here, object motions are obtained from the force equations describing physical laws, such as Newton's laws of motion for gravitational and friction processes, Euler or Navier-stokes equations describing fluid flow and Maxwell's equations for electromagnetic forces.

Review Question

1. Explain various methods to specify the motion of the objects.

SPPU : Dec.-19, Marks 3



Notes

UNIT - VI

14

Gaming

Syllabus

Introduction, Gaming platform (NVIDIA, i8060), Advances in Gaming.

Contents

14.1 Introduction

14.2 Gaming Platforms Dec.-14,15,16,17,18,19,

. May-14,15,17,18,19, Marks 4

14.3 Advances in Gaming

14.1 Introduction

- Gaming is a part of our world and it is getting bigger and bigger. Now games are more realistic than ever and they take you away into a virtual, interactive and imaginary world where fiction and reality meet.
- Game design : It is the process of designing the content, background and rules of a game or class of games (i.e., using an existing engine to create game content).
- Game development : It is the process by which a specific game or class of games is produced (i.e., creating specific engines for games given current technologies).
- Games technology : It is the process of developing methodologies and techniques applicable to classes of current and future games (i.e., foundational computational, engineering and scientific concepts for making games and other types of applications).
- Gaming technologies is a technological aspects of media and games programming. This will enable you to use appropriate tools and techniques to integrate music, audio, graphics, animation and games for distributed applications.
- It is a science of tools and techniques used for the development of games, animation and graphics.
- Learning gaming technology is nothing but the understanding of video, audio, computers and an in-depth understanding of 3D modelling, graphics, animation and games programming.
- Gaming technology involves the gaming engine, art tool, scripting tool, artificial intelligence (AI) systems, physics systems and other technical elements that needed to provide desired game functionality.
- The technology used will depend on the schedule, the resources, the desired features and the quality of features. For example, if the main goal of the game is to have cutting-edge graphics, the leader engineer will spend some time evaluating what graphics technology is necessary.

14.2 Gaming Platforms

SPPU : Dec.-14,15,16,17,18,19, May-14,15,17,18,19

14.2.1 NVIDIA GPU

- GPU computing is the use of a GPU (graphics processing unit) together with a CPU to accelerate general-purpose scientific and engineering applications.
- Pioneered five years ago by NVIDIA, GPU computing has quickly become an industry standard.

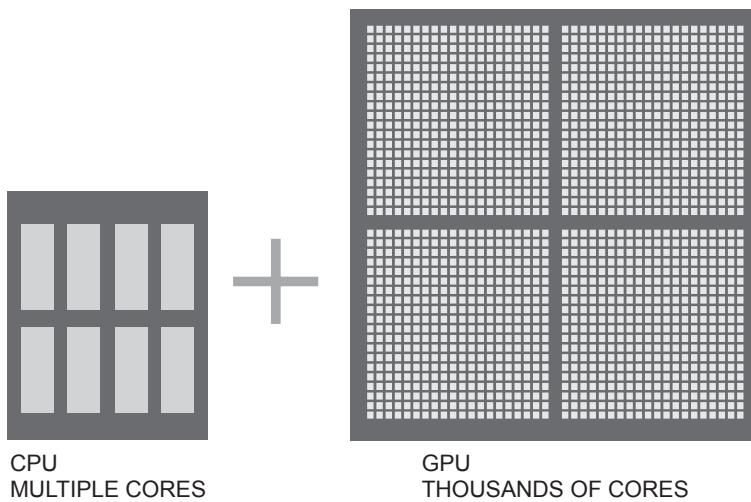


Fig. 14.2.1 CPU + GPU combination

- GPU computing offers unprecedented application performance by offloading compute-intensive portions of the application to the GPU, while the remainder of the code still runs on the CPU. From a user's perspective, applications simply run significantly faster.
- CPU + GPU is a powerful combination because CPUs consist of a few cores optimized for serial processing, while GPUs consist of thousands of smaller, more efficient cores designed for parallel performance. Serial portions of the code run on the CPU while parallel portions run on the GPU.

14.2.1.1 Connection between CPU and GPU

- Fig. 14.2.2 shows how a GPU is typically connected with a modern processor. A GPU is an accelerator (or a co-processor) that is connected to a host processor.
- The host processor and GPU communicate to each other via PCI Express (PCIe) that provides 4 Gbit/s (Gen 2) or 8 Gbit/s (Gen 3) interconnection bandwidth.
- This communication bandwidth often becomes one of the biggest bottlenecks; thus it is critical to offload the work to GPUs only if the benefits of using GPUs outweigh the offload cost.

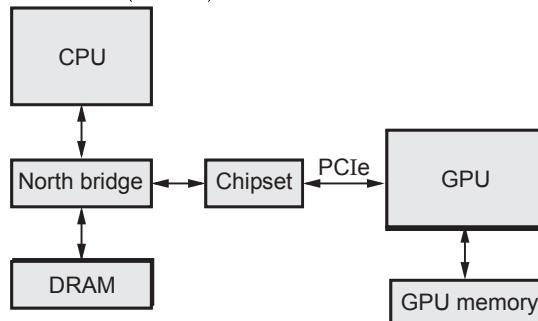


Fig. 14.2.2 A system overview with CPU and a discrete GPU

14.2.1.2 GPU Architecture

- Fig. 14.2.3 illustrates the major components of a general-purpose graphics processor.



Fig. 14.2.3 Block diagram of NVIDIA's graphics processor (the components required for graphics processing are not shown)

- At a high level, the GPU architecture consists of several **streaming multiprocessors (SMs)**, which are connected to the GPU's DRAM.
- Each SM has a number of single-instruction multiple data (SIMD) units, also called **stream processors (SPs)**, and supports a multithreading execution mechanism. GPU architectures employ two important execution paradigms :
 - SIMD / SIMT
 - Multithreading

SIMD / SIMT

- GPU processors supply high Floating Point (FP) execution bandwidth, which is the driving force for designing graphics applications. To make efficient use of the high number of FP units, GPU architectures employ a **SIMD** or **SIMT** execution paradigm.

- In SIMD, one instruction operates on multiple data (i.e., only one instruction is fetched, decoded and scheduled but on multiple data operands). Depending on the word width, anywhere from 32, 64 or 128 FP operations may be performed by a single instruction on current systems. This technique significantly increases system throughput and also improves its energy-efficiency.

Multithreading

- The other important execution paradigm that GPU architectures employ is ***hardware multithreading***.
- GPU processors use fast hardware-based context switching to tolerate long memory and operation latencies.
- The effectiveness of multithreading depends on whether an application can provide a high number of concurrent threads.
- Most graphics applications have this characteristics since they typically need to process many objects (e.g. pixels, vertices, polygons) simultaneously.
- In conventional CPU systems, thread context switching relatively much more expensive : all program states, such as PC (Program Counter), architectural registers and stack information, need to be stored by the operating system in memory. However, in GPUs, the cost of thread switching is much lower due to native hardware support of such process.
- The *degree* of multithreading (the number of simultaneous hardware thread contexts) is much lower in CPUs than in GPUs (e.g. two hyperthreads Vs. hundreds of GPU thread).

14.2.1.3 Features of NVIDIA Gaming Platform

1. Highest level performance and the smoothest experience possible from the moment you start playing.
2. Enables developers to add amazing graphics effects.
3. Dedicated ray tracing hardware enables fast real-time ray tracing with physical accurate shadows, reflection, refractions and global elimination.
4. Provides variable rate shading and faster frame rates.

14.2.2 i860

14.2.2.1 Features

- The Intel i860 Microprocessor architecture balances integer, floating-point and graphics performance.

- Target applications include engineering workstations, scientific computing, 3-D graphics workstations, numerics accelerators, multiuser and multiprocessor systems.
- The architecture achieves high throughput with **RISC design** techniques, pipelined and parallel processing units, wide data paths and large on-chip caches.
- The i860 microprocessor includes on a single chip :
 - Integer operations
 - Floating-point operations
 - Graphics operations
 - Memory management
 - Data and instruction caches
- Having a data cache as an integral part of the architecture provides **support for vector operations**.
- To sustain high performance, i860 microprocessors incorporate **wide information paths** that include :
 - 64-bit external data bus
 - 128-bit on-chip data bus
 - 64-bit on-chip instruction bus
- Floating-point and graphics programs can simultaneously use all three buses.
- The i860 microprocessors include a **RISC integer core processing unit** with one-clock instruction execution. The core unit processes integer instructions and provides complete up-port for operating systems, such as UNIX and OS/2.
- The core unit also drives the **graphics and floating-point hardware**.
- The i860 microprocessors support vector floating-point operations without special vector instructions or vector registers. They accomplish this by using the on-chip data cache and a variety of parallel techniques that include :
 - Pipelined instruction execution with delayed branch instructions to avoid breaks in the pipeline.
 - Instructions that automatically increment index registers so as to reduce the number of instructions needed for vector processing.
 - Simultaneous integer and floating-point processing.
 - Parallel multiplier and adder units within the floating-point unit.
 - Pipelined floating-point hardware, with both scalar (nonpipelined) and vector (pipelined) variants of floating-point instructions. Software can switch between scalar and pipelined modes.

- Large register set :
 - 32 general-purpose integer registers, each 32 bits wide.
 - 32 floating-point registers, each 32 bits wide, which can also be configured as 64 and 128-bit registers. The floating-point registers also serve as the staging area for data going into and out of the floating-point and graphics pipelines.

14.2.2.2 Block Diagram

Fig. 14.2.4 illustrate the registers and data paths of the i860 microprocessor respectively.

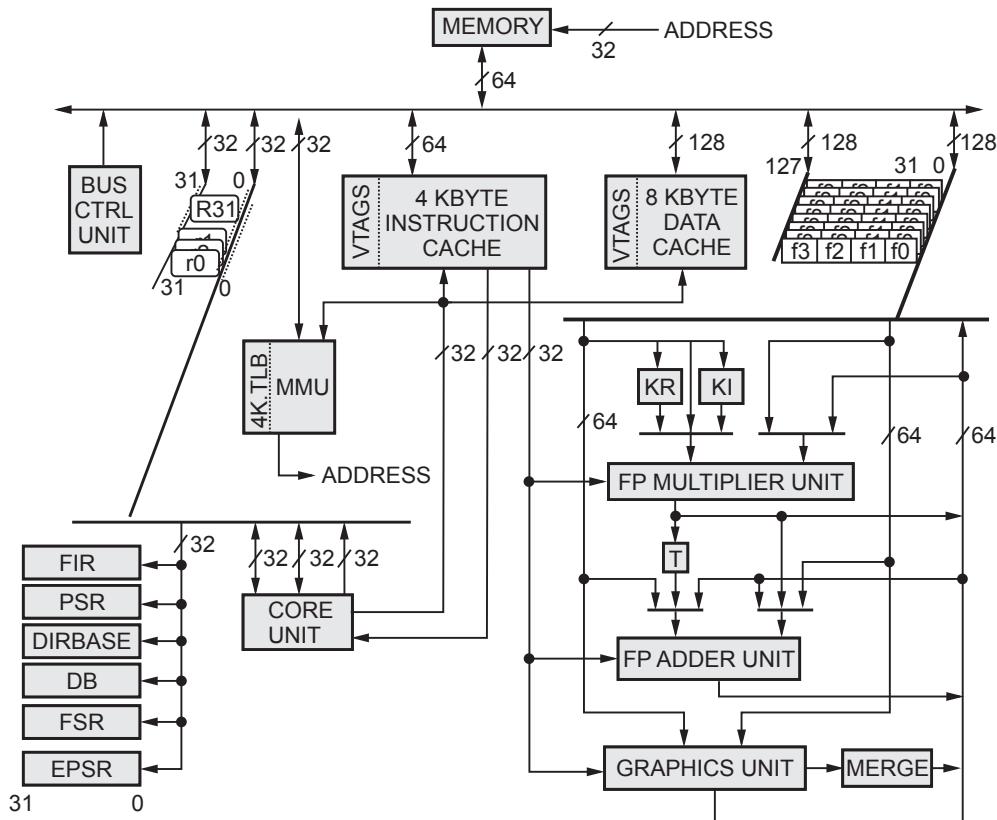


Fig. 14.2.4 i860 microprocessor architecture block diagram

Integer Core Unit

- The core unit is the administrative center of the processor. The core unit fetches both integer and floating-point instructions.
- It contains the integer register file and executes load, store, integer, bit and control-transfer operations. Its pipelined organization with extensive bypassing and scoreboard maximizes performance.

Floating-Point Unit

- The floating-point unit contains the **floating-point register file**. This file can be accessed as 8×128 -bit registers, 16×64 -bit registers or 32×32 -bit registers. Three additional registers (**KR KI, and T**) hold intermediate floating-point results.
- The floating-point unit contains both the floating-point adder and the floating-point multiplier. Both units support 64 and 32-bit floating-point values in IEEE Standard 754 format.
- Each of these units uses pipelining to deliver up to one result per clock. The adder and multiplier can operate in parallel, producing up to **two results per clock**.
- Furthermore, the floating-point unit can operate in parallel with the core unit, sustaining a rate of **two floating-point results per clock** rate by overlapping administrative functions with floating-point operations.

Graphics Unit

- The graphics unit has 64-bit integer logic that supports 3-D graphics drawing algorithms.
- This unit can **operate in parallel with the core unit**.
- It contains the special-purpose MERGE register and performs additions on integers stored in the floating-point register file.
- These special graphics features focus on applications that involve three-dimensional graphics with Gouraud or Phong color intensity shading and hidden surface elimination via the Z-buffer algorithm.

Memory Management Unit

- The on-chip MMU of i860 microprocessors performs the translation of addresses from the linear logical address space to the linear physical address for both data and instruction access.

Caches

- In addition to the page translation caches (TLBs), the i860 microprocessor contains separate on-chip caches for data and instructions.

Review Questions

1. Explain the architecture of NVIDIA GPU. **SPPU : Dec.-15,17,18, May-17,18,19, Marks 4**
2. Explain the features of i860.
3. Draw and explain the block diagram of i860.

SPPU : May-14,15,17, Dec.-14,16,17,19, Marks 4

4. Explain the significance of NVIDIA workstation in gaming.

SPPU : Dec.-16, Marks 4

14.3 Advances in Gaming

Currently, there are tremendous **advances in gaming** technologies to improve physical realism of environments and characters.

1. Facial Recognition

3D scanning and facial recognition technology allows systems to actually create your likeness in the gaming world or to inventively transfer your own expressions to other digital creations. On top of that, the Intel® RealSense™ 3D camera could allow developers to create games that adapt to the emotions of the gamer by scanning 78 different points on a person's face.

2. Voice Recognition

Voice controlled gaming has been around for a while, but the potential of using the technology in gaming systems has finally caught up to reality-computers are now able to easily recognize voice commands from the user. Not only you can turn the console on and off using this technique, but you can also use voice commands to control gameplay, interact on social media, play selections from your media library, or search the web, all by simply talking to your gaming system.

3. Gesture Control

Gesture control allows users to connect with their gaming experience by using the natural movements of your body.

4. Amazing Graphics

Cutting edge advancements now allow gamers to experience games in fully rendered worlds with photo realistic textures. The ability to increase playability with higher image quality makes it seem like you're right inside the game.

5. High-definition Displays

Televisions with 4K capabilities (meaning it must support at least 4,000 pixels) or 4K laptops are now available to watch the games we play.

6. Virtual Reality

Though many virtual reality gaming consoles haven't been commercially released as of yet, those developing VR headset displays are poised to grant gamers a fully immersive gaming experience the likes of which nobody has seen before.

7. Augmented Reality

Augmented Reality (AR) is a live direct or indirect view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data.

8. Wearable Gaming

Whether it's smartwatches or glasses, wearable games make gaming portable without being too invasive.

9. Mobile Gaming

With the advent of smartphones, the gaming experience has been taken out of the arcade and the living room and put into the palm of your hand. As evidenced by the countless people on your morning train commute huddled over games on their devices, mobile technology has made the love of digital gaming spread beyond hardcore console-consumers and online gamers.

10. Cloud Gaming

Instead of creating video game systems that require more powerful hardware, developers are looking to lighten the load with the cloud. Games no longer need be limited by the amount of memory that discs or consoles have to offer. Using the cloud opens games up to massive server-size limits where images are streamed to your screen through the Internet.

11. On-demand Gaming

Gamers can already watch and share live-streams of games, but what about playing them ? Much like similar movie streaming services, the ability to stream video games is becoming more and more a reality, and it could lead game developers both big and small to compete for gaming glory.

Review Question

1. Write a note on advance in gaming technology.

