Computer Graphics

(Code : 22318)

SECOND YEAR DIPLOMA

Maharashtra State Board of Technical Education (MSBTE)

Semester III - Computer Engineering Program Group(CO/CM/CW)

Strictly as per new revised syllabus of 'I' Scheme w.e.f. academic year 2018-2019

Dr. Mahesh M. Goyani

Ph.D., M.E., B.A., B.E.
Assistant Professor
Department of Computer Engineering
Government Engineering College, Modasa
Gujarat, India

TechKnowledge™ Publications

MDO12B Price ₹ 215/-

(Book Code : MDO12B)

Computer Graphics (22318)

Dr. Mahesh M. Goyani

(Semester III - Computer Engineering Program Group (CO/CM/CW))

First Printed in India : August 2015
First Edition : June 2019(As per 'I' Scheme)
Second Revised Edition : September 2020

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : 37/2, Ashtavinayak Industrial Estate,
Near Pari Company,
Narhe, Pune, Maharashtra State, India.
Pune – 411041

ISBN : 978-93-89233-96-4

Published by
TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex,
Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India
Ph : 91-20-24221234, 91-20-24225678.

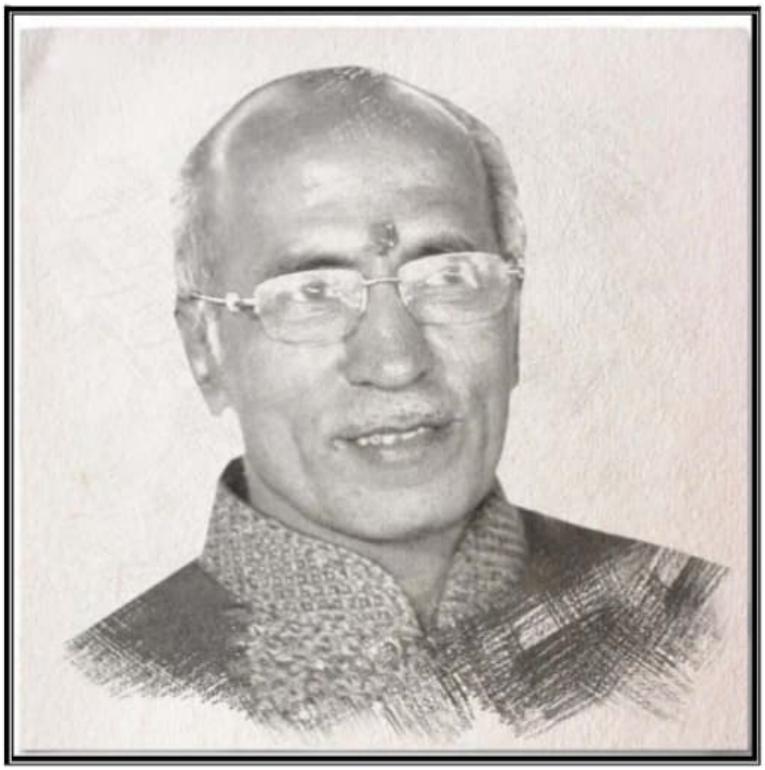
[22318](FID : MDO12)(Book Code : MDO12B)

We dedicate this Publication soulfully and wholeheartedly,

in loving memory of our beloved founder director,

Late Shri. Pradeepji Lalchandji Lunawat,

who will always be an inspiration, a positive force and strong support behind us.



"My work is my prayer to God"

- Lt. Shri. Pradeepji L. Lunawat

Soulful Tribute and Gratitude for all Your
Sacrifices, Hardworkand 40 years of Strong Vision...

Dedicated to.....

My dearest Nephews

Divy and Smit

Who made the time more brighter and happier for me

Preface

My Dear Students,

Computer Graphics is a very dynamic and rapidly growing area amongst others in Computer Science. Entertainment, education, animation, film industry, advertisements, presentations, virtual reality, computer aided design and many other fields rely extensively on computer graphics, owing to its importance Computer Graphics has become an indivisible part not only in academic but in professional ventures as well.

The book **"Computer Graphics"** has been written according to the new syllabus of Maharashtra State Board Technical Education for students pursuing Diploma., with a vision to provide students the necessary resources for the formation of a strong base in Computer Graphics. This book discusses the concepts of Computer Graphics with self-explanatory diagrams and examples. Care has been taken to provide the reader with an understanding of the principles of Computer Graphics along with their implementation and applications. Numerous examples have been covered to allow the readers an even better understanding of the concepts.

The book has been written in a lucid manner and the technical jargon has been simplified so as to make it comprehensible by people from all disciplines, the questions from previous university examinations have been included at the end of every chapter and their solutions are covered within the text. This facilitates the students to prepare a chapter or a concept from an examination point of view as well.

Key features:

- Simplified explanation in lucid manner
- Solved university question papers
- Algorithms of all problems
- Detail coverage of each and every topic
- Set of self-explanatory diagrams
- Wide range of programs
- Programs

Chapter 1 discusses the basics of computer graphics with its applications. The discussion is followed by detailed review on various kinds of video display devices and input devices.

Chapter 2 throws light on primitive drawing and area filling algorithms. It also provides brief notes on character generation methods followed by line, area and character attributes.

Chapter 3 covers theoretical foundations of two dimensional viewing and transformation with appropriate examples for each concept.

Chapter 4 introduces three dimensional transformation followed by various projection methods.

Chapter 5 describes various methods for line and polygon clipping. These concepts are very essential in realistic scene rendering

Chapter 6 is devoted to various polygon representation methods. It covers methods for the generation of various curves and surfaces.

(Book Code : MDO12B)

Acknowledgement

This book is the result of the numerous discussions I had in lectures with my students and peers while offering this course at different levels of study. I appreciate all the constructive comments from colleges and doubts raised by students.

I would like to express my gratitude to Dr. N. M. Patel who not only lead me towards my discovery of interest in computer graphics but also put efforts in helping me to maintain it. My friends and colleagues have been a constant source of inspiration for me. I thank everyone who has contributed to this book directly or indirectly.

I am thankful to Shri. J. S. Katre, Shri. Shital Bhandari, Shri. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they haveextended.I am also thankful to Seema Lunawat for technology enhanced reading, E-books support and the staff members of TechKnowledge Publications for their efforts to make this book as good as it is. We have jointly made every possible effort to eliminate errors in the book.

Last but not least, I would like to sincerely thank my family for supporting me throughout my venture of writing this book, without them it wouldn't have been possible.

– Mahesh Goyani

Syllabus

Unit	Unit Outcomes (UOs) (incognitivedomain)	TopicsandSub-topics
Unit - I **Basics of Computer Graphics** **Refer Chapter 1**	1a. Differentiate attributes of the given mode. 1b. Compare features of the given Scan Display. 1c. Write a program to draw the given type of primitives using "C". 1d. Describe application of the given display device. 1e. Convert the given 2D co-ordinates to physical device co-ordinates.	1.1 Image and object, Pixel and resolution, Text mode, Graphic mode, Basic graphics pipeline, Bitmap and Vector based graphics, Applications of computer graphics. 1.2 Display Deices : Raster-Scan display, Random-Scan display, Flat panel display, LED, LCD display, Plasma, Touch screen. 1.3 Output primitives : Line, Polygon, Marker, Text. 1.4 Graphics functions and standards. 1.5 Latest trends in Computer Graphics : Virtual reality, Augmented reality.
Unit - II **Raster Scan Graphics** **Refer Chapters 2**	2a. Write a program to draw a line using the given algorithm. 2b. Use the given algorithm to rasterize the given line. 2c. Apply the given algorithm to generate the circle. 2d. Draw the Polygon using the given algorithm. 2e. Apply character generation method to display the given character.	2.1 Basic concepts in line drawing : Line drawing algorithms : Digital Differential Analyzer (DDA) algorithm, Bresenham's algorithm. 2.2 Circle generating algorithms : Symmetry of circle, Bresenham's circle drawing algorithm. 2.4 Polygons - Types of polygons, inside-outside test, Polygon Filling : Seed fill algorithms : Flood fill, Boundary fill, Scan line algorithms. 2.5 Scan conversion, Frame Buffers. 2.6 Character generation methods : stroke, starburst, bitmap.

Unit	Unit Outcomes (UOs) (incognitivedomain)	TopicsandSub-topics
Unit - III **Overview of Transformations** **Refer Chapters 3 & 4**	3a. Perform the given operation in 2D transformation. 3b. Perform the given operation in 3D transformation. 3c. Solve the given problem based on Composite Transformation. 3d. Apply the given type of projection on object.	3.1 Two Dimensional Transformations : Translation, Scaling, Rotation, Reflection, Shearing. 3.2 Matrix representations and homogeneous co-ordinates : Translation Scaling, Rotation, Reflection, Shearing. 3.3 Composite Transformations - rotation about an arbitrary point. 3.4 Three dimensional transformations : Translation, Scaling, Rotation. 3.5 Types of Projections : Perspective and Parallel Projection.
Unit - IV **Windowing and Clipping** **Refer Chapter 5**	4a. Apply Window to-viewport transformation on the given object. 4b. Write a program using the given line clipping algorithms. 4c. Apply the given line clipping algorithms to clip the line. 4d. Apply text clipping on the given text. 4e. Write a program using the given polygon clipping algorithm.	4.1 Windowing and clipping concepts : Window to-viewport transformation. 4.2 Line clipping : Cohen Sutherland clipping algorithm, Cyrusbeck, Liang Barsky, Midpoint subdivision. 4.3 Polygon clipping : Sutherland-Hodgeman. 4.4 Text clipping.
Unit - V **Introduction to Curves** **Refer Chapter 6**	5a. Describe the given curve generation methods. 5b. Draw curve using the given curve algorithms. 5c. State properties of the given curve. 5d. Generate arc using the given algorithm.	5.1 Curve generation : Arc generation using DDA algorithm, Interpolation. 5.2 Types of curves : Hilbert's Curve, Koch curve, B-spline, Bezier curves.

UNIT I

Chapter 1 : Basics of Computer Graphics 1-1 to 1-33

Syllabus :

Image and Objects, pixels and resolution, Text mode, Graphics Mode. Basic Graphics pipeline, Bitmap and Vector based graphics, Applications of Computer Graphics,

Display Devices : Raster Scan Display, Random Scan Display, Flat Panel Display, LED-LCD Display, Plasma, Touch Screen,

Output Primitives : line, polygon, marker, text, Graphics functions and standards, Latest trends in Computer Graphics : Virtual reality, Augmented reality

1.1 Introduction to Computer Graphics 1-1
1.2 Image and Object 1-2
1.3 Pixels, Resolution and Frame Buffer 1-2
1.3.1 Pixel 1-2
1.3.2 Resolution 1-3
1.3.3 Frame Buffer 1-3
1.4 Display Adapters 1-4
1.4.1 Video Adapters 1-4
1.4.2 Display Modes 1-8
1.4.2.1 Text Mode 1-8
1.4.2.2 Graphics Mode 1-9
1.5 Basic Graphics Pipeline 1-9
1.5.1 Application Stage 1-9
1.5.2 Geometry Stage 1-9
1.5.3 Rasterization 1-10
1.6 Types of Graphics 1-10
1.6.1 Bitmap based Graphics 1-10
1.6.2 Vector based Graphics 1-10
1.6.3 Bitmap vs. Vector Graphics 1-11
1.7 Applications of Computer Graphics 1-11
1.8 Display Devices 1-14
1.8.1 Cathode Ray Tube (CRT) 1-14
1.8.2 Raster Scan Display 1-16
1.8.3 Random Scan Display 1-17
1.8.4 Raster scan vs. Random Scan Display 1-18
1.8.5 Interlaced Display System 1-18
1.8.6 Flat Panel Display 1-19
1.8.6.1 LED Display 1-20
1.8.6.2 LCD Display 1-20
1.8.6.3 Plasma Panel 1-21
1.8.6.4 Touch Screen 1-21
1.9 Output Primitives 1-21
1.9.1 Point 1-22
1.9.2 Line 1-22
1.9.3 Polygon 1-22
1.9.4 Marker 1-23
1.9.5 Text 1-23
1.10 Graphics Functions and Standards 1-24
1.10.1 Coordinate Representation 1-24
1.10.1.1 Cartesian Coordinate System 1-25
1.10.1.2 Polar Coordinate System 1-25
1.10.1.3 Spherical Coordinate System 1-26
1.10.1.4 Cylindrical Coordinate System 1-26
1.10.2 Graphics Functions 1-26
1.10.3 Software Standards 1-27
1.11 Latest trends in Computer Graphics 1-27
1.11.1 Virtual Reality 1-27
1.11.1.1 Introduction to Virtual Reality 1-27
1.11.1.2 Components of Virtual Reality 1-28
1.11.1.3 Types of VR Systems 1-28
1.11.1.4 Input / Output Devices 1-30
1.11.1.5 Applications of VR 1-30
1.11.2 Augmented Reality 1-31
1.11.2.1 Introduction to Augmented Reality 1-31
1.11.2.2 Types of Augmented Reality 1-31

UNIT II

Chapter 2 : Raster Scan Graphics 2-1 to 2-53

Syllabus :

Basic Concepts in Line Drawing : Line drawing algorithms, Digital Differential Analyzer (DDA) algorithm, Bresenham's algorithm, Circle Generating Algorithms: Symmetry of circle, Bresenham's Circle Drawing Algorithm, Polygons, Types of Polygons, Inside Outside Test, Polygon Filling : Seed fill algorithms : Flood Fill, Boundary Fill, Scan Line Algorithm, Scan Conversion, Frame Buffers, Character Generation Methods : Stroke, Starbust, Bitmap

2.1 Basic Concepts in Line Drawing 2-1
2.1.1 Point 2-1
2.1.2 Basics of Line 2-1
2.1.2.1 Characteristics of Ideal Line 2-2
2.1.2.2 Line Representation 2-3

2.2 Line Drawing Algorithms 2-3
2.2.1 Digital Differential Analyzer (DDA) Algorithm 2-3
2.2.2 Bresenham's Line Drawing Algorithm 2-12
2.2.3 DDA vs. Bresenham's Line Drawing Algorithm 2-18
2.3 Circle Generation Algorithms 2-18
2.3.1 Symmetry of Circle 2-19
2.3.2 Polar Circle Drawing Algorithm 2-20
2.3.3 Bresenham's Circle Drawing Algorithms 2-21
2.4 Polygons 2-26
2.4.1 Types of Polygons 2-26
2.4.1.1 Convex Polygon 2-27
2.4.1.2 Concave Polygon 2-27
2.4.1.3 Complex Polygons 2-27
2.4.2 Inside-Outside Test 2-27
2.4.2.1 Even–Odd Method 2-28
2.4.2.2 Winding Number Method 2-29
2.5 Polygon Filling 2-29
2.5.1 Pixel Connectivity 2-30
2.5.2 Seed Fill Algorithms 2-30
2.5.2.1 Boundary Fill Algorithm 2-30
2.5.2.2 Flood Fill Algorithm 2-32
2.5.3 Scan Line Polygon Filling Algorithms 2-34
2.5.4 Seed Fill vs. Scan Line Algorithm 2-35
2.6 Scan Conversion 2-38
2.7 Frame Buffer 2-39
2.8 Character Generation Methods 2-40
2.8.1 Stroke Method 2-40
2.8.2 Starburst Method 2-41
2.8.3 Bitmap Method 2-41
2.9 Lab Programs 2-42

UNIT III

Chapter 3 : 2D Transformation 3-1 to 3-60

Syllabus :

Two Dimensional Transformations : Translation, Scaling, Rotation, Reflection, Shearing,

Matrix Representations and Homogeneous Coordinates : Translation, Scaling, Rotation, Reflection, Shearing, Composite Transformations : Rotation About and Arbitrary Point

3.1 Introduction 3-1
3.1.1 Geometric Transformation 3-1
3.1.2 Coordinate Transformation 3-2
3.2 Matrix Operations 3-2
3.2.1 Representation of Matrix 3-2
3.2.1.1 Column Measure Representation 3-2
3.2.1.2 Row Measure Representation 3-3
3.2.2 Matrix Properties 3-3
3.2.3 Determinant of Matrix 3-3
3.2.4 Multiplying Two Matrices 3-4
3.3 Translation 3-4
3.4 Scaling 3-6
3.4.1 Scaling with respect to Origin 3-6
3.4.2 Scaling with respect to Reference Point 3-7
3.4.3 Uniform vs. Non Uniform Scaling 3-8
3.5 Rotation 3-10
3.5.1 Rotation with respect to Origin 3-11
3.5.2 Rotation with Respect to Reference Point 3-12
3.6 Matrix Representation and Homogeneous Coordinates 3-14
3.6.1 Translation 3-15
3.6.2 Scaling 3-15
3.6.3 Rotation 3-16
3.7 Composite Transformation 3-16
3.7.1 Successive Transformations 3-16
3.7.2 General Pivot Point Rotation 3-17
3.7.3 General Pivot Point Scaling 3-23
3.8 Reflection 3-24
3.8.1 Reflection about X-Axis (Y = 0 Line) 3-24
3.8.2 Reflection About Y Axis (X = 0 Line) 3-25
3.8.3 Reflection about X = Y Axis 3-26
3.8.4 Reflection about X = – Y Axis 3-26
3.8.5 Reflection about Origin 3-26
3.8.6 Reflection about Any Line y = mx + c 3-27
3.8.7 Reflection about a Line Parallel to X Axis 3-27
3.8.8 Reflection about a Line Parallel to Y Axis 3-28
3.9 Shearing 3-33
3.9.1 X-direction Shearing 3-33
3.9.2 Y direction Shearing 3-36
3.10 Solved Examples 3-39
3.11 Lab Programs 3-51

UNIT III

Chapter 4 : 3D Transformation and Projection 4-1 to 4-41

Syllabus :

Three Dimensional Transformations : Translation, Scaling, Rotation,

Types of Projections : Perspective and Parallel Projection

4.1 Three Dimensional Transformations 4-1
4.1.1 Translation 4-1
4.1.2 Scaling 4-2
4.1.2.1 Scaling with respect to Origin 4-2
4.1.2.2 Scaling with respect to Reference Point 4-3
4.1.3 Rotation 4-4
4.1.3.1 Rotation about a Principal Axis 4-4
4.1.3.2 Rotation about a Line Parallel to Principal Axis 4-6
4.1.3.3 Rotation about an Arbitrary Line 4-7
4.1.4 Solved Examples 4-11
4.2 Types of Projection 4-20
4.2.1 Introduction to Projection 4-20
4.2.2 Parallel Projection 4-21
4.2.2.1 Orthographic Parallel Projection 4-22
4.2.2.2 Oblique Parallel Projection 4-24
4.2.3 Perspective Projection 4-29
4.2.3.1 One-point Perspective Projection 4-31
4.2.3.2 Two-point Perspective Projection 4-33
4.2.3.3 Three-point Perspective Projection 4-33
4.2.4 Parallel Vs. Perspective Projection 4-37
4.3 Lab Programs 4-37

UNIT IV

Chapter 5 : Windowing and Clipping 5-1 to 5-49

Syllabus :

Windowing and Clipping Concepts : Window-to-Viewport Transformation,

Line Clipping : Cohen Sutherland Clipping Algorithm, Cyrus beck, Liang Barsky, Midpoint Subdivision, Polygon Clipping Sutherland-Hodgeman, Text Clipping

5.1 Windowing and Clipping Concepts 5-1
5.1.1 Introduction 5-1
5.1.2 Viewing Pipeline 5-1
5.1.3 Window-to-viewport Transformation 5-3
5.2 Point Clipping 5-5
5.3 Line Clipping 5-6
5.3.1 Introduction 5-6
5.3.2 Line Clipping Algorithm 5-6
5.3.2.1 Cohen Sutherland Line Clipping Algorithm 5-7
5.3.2.2 Midpoint Subdivision Line Clipping Algorithm 5-17
5.3.2.3 Cyrus-Beck Line Clipping Algorithm 5-19
5.3.2.4 Liang Barsky Line Clipping Algorithm 5-22
5.4 Polygon Clipping 5-30
5.4.1 Introduction to Polygon Clipping 5-30
5.4.2 Sutherland-Hodgeman Algorithm 5-30
5.5 Text Clipping 5-33
5.5.1 All or None String Clipping 5-33
5.5.2 All or none Character Clipping 5-33
5.5.3 Individual Character Clipping 5-34
5.6 Lab Programs 5-34

UNIT V

Chapter 6 : Introduction to Curves 6-1 to 6-30

Syllabus :

Curve Generation : Arc Generation Using DDA Algorithm, Interpolation,

Types of Curves : Hilbert's Curve, Koch Curve, B-Spline, Bezier Curves

6.1 Introduction to Curve Generation 6-1
6.1.1 Arc Generation using DDA Algorithm 6-1
6.1.2 Interpolation 6-3
6.1.2.1 Lagrange Interpolation 6-3
6.2 Types of Curve 6-5
6.2.1 Spline Curve Representation 6-5
6.2.1.1 Bezier Curve 6-6
6.2.1.2 B-Spline Curve 6-14
6.2.1.3 Bezier vs. B-Spline 6-18
6.2.2 Fractal Representation 6-18
6.2.2.1 Hilbert's Curve 6-21
6.2.2.2 Koch Curve 6-22
6.3 Lab Programs 6-23

➢ **Appendix A :** Solved MSBTE Question Papers of Summer 2019 and Winter 2019 ..... **A-1 to A-8**

❑❑❑

List of Practicals

Lab No.	Program Name	Program No.	Page No.
1	Write Programs to draw following graphics object using built-in "C' functions. (i) Pixel (ii) Lines (iii) Circles (iv) Rectangle (v) Ellipse	**Program 2.9.1**	2-42
2	Implement DDA algorithm to draw line.	**Program 2.9.2**	2-44
3	Implement Bresennham's algorithm to draw line.	**Program 2.9.3**	2-45
4	Implement Bresennham's algorithm to draw circle.	**Program 2.9.4**	2-47
5	Write a program to fill polygon using Flood fill.	**Program 2.9.5**	2-49
6	Write a program to fill polygon using Boundary fill.	**Program 2.9.6**	2-50
7	Write a program for two-dimensional transformation – Translation and Scaling	**Program 3.11.1**	3-51
8	Write a program for two-dimensional transformation – Rotation.	**Program 3.11.2**	3-53
9	Write a program for two-dimensional transformation – Reflection.	**Program 3.11.3**	3-54
	Write a program for two-dimensional transformation – Shearing.	**Program 3.11.4**	3-56
10 & 11	Write a program for three-dimensional transformation – Translation. Write a program for three-dimensional transformation – Scaling. Write a program for three-dimensional transformation – Rotation.	**Program 4.3.1**	4-37
12	Write a program to clip line using Cohen- Sutherland algorithm.	**Program 5.6.1**	5-34
13	Write a program to clip line using Cohen Midpoint subdivision algorithm.	**Program 5.6.2**	5-38
14	Write a program to clip polygon using Sutherland -Hodgeman Algorithm.	**Program 5.6.3**	5-42
15	Write a program to draw Hilbert's curve	**Program 6.3.1**	6-23
16 (a)	Write a program to draw Koch curve	**Program 6.3.2**	6-26
16 (b)	Write a program to Bezier curves	**Program 6.3.3**	6-28

❑❑❑

Basics of Computer Graphics

Syllabus

Image and Objects, pixels and resolution, Text mode, Graphics Mode. Basic Graphics pipeline, Bitmap and Vector based graphics, Applications of Computer Graphics,

Display Devices : Raster Scan Display, Random Scan Display, Flat Panel Display, LED-LCD Display, Plasma, Touch Screen,

Output Primitives : line, polygon, marker, text, Graphics functions and standards, Latest trends in Computer Graphics : Virtual reality, Augmented reality

1.1 Introduction to Computer Graphics

Q. What is computer graphics?

- Computers have been a core part of efficient and economical image generation and manipulation. Advancement in computer technology has made the graphics processing faster and economical.
- Today, the scope of term computer graphics is not limited to what kind of scene can be displayed or what kind of images we can generate with the help of a computer. In broad mean, computer graphics has covered almost each and every field, like education, animation, entertainment, film industry, fine arts, gaming, engineering, training, advertisement, medicine, statistical representation and many more.
- **Computer graphics** is a process of generating, manipulating, storing and displaying images. Computer graphics has made things more easy and interesting.
- A person is more interested to study monthly or annual statistics in the form of graphs, tables or charts, instead of plain boring textual data.
- Computer graphics is the key component for presentation and user interface. Old slow and time-consuming command line interfaces have been replaced by quick, fascinating Graphical User Interface (GUI). GUI provides quick and easy to use interface to the application users.
- Due to inefficient and expensive hardware, before a couple of decades, application areas of computer graphics were limited. But with evolution in a digital system and personal computers, there is a flood of inexpensive graphics-based applications and devices.
- With the help of GUI, multiple users can control simultaneous applications like text editors, code editors, entertainment tools etc. very easily. Computer graphics was initially started with displaying data on hard copy display devices like plotters, printers and then Cathode Ray Tube (CRT). Today, the applications of computer graphics have been extended to display devices like Light Emitting Diodes (LED), Liquid Crystal Display (LCD), electroluminescent display, flat CRTs etc.

– Computer graphics performs various operations on objects like translation, rotation, scaling, reflection, shearing, hidden surface removal, texture mapping, shading etc. to generate a realistic scene.

Operations of computer graphics can be classified as follows :

- **Geometry :** How to draw and represent the scene.
- **Animation :** How to add dynamic content in the scene.
- **Rendering :** How to add realistic light in the scene.
- **Imaging :** Image acquisition and image editing.

1.2 Image and Object

Q. Define : Object and Image.

Q. How real world object is converted to image?

Objects are real-world entities representing actual things. Objects may be natural like flowers, cloud, mountain, trees etc. or it may be artificial like a car, cell phone, book, ball etc. Objects are represented using a three-dimensional coordinate system. Computer graphics deals with mapping the real world objects to the computer screen.

Representation of real-world objects on the digital display device is known as an **image**. Objects in the real world are continuous in nature, they have a smooth boundary. When they are mapped on a monitor screen, they are discretized and often they have zigzag boundary due to the grid structure of the monitor screen.

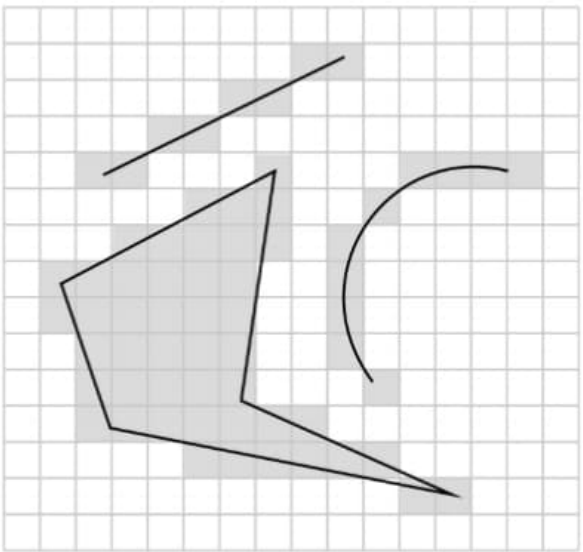


Fig. 1.2.1 : Discretized shapes in a digital image.

Formally, Image is a two-dimensional array of pixels. **Pixel** is the smallest addressable unit of an image. Mathematically, image is defined as a function of its spatial coordinates (x, y). Where (x, y) defines the position of the pixel in image and function *f(x, y)* represents the intensity or color at that position. The Fig. 1.2.1 shows how the continuous shapes are converted to the discrete entity while converted into digital image or mapped on the monitor screen.

1.3 Pixels, Resolution and Frame Buffer

Q. Explain the terms : Pixels, Resolution, Frame Buffer.

1.3.1 Pixel

W-18

Q. Define pixel. **(W-18, 2 Marks)**

Q. Explain how pixels are related to the display memory.

As discussed earlier, the ***pixel*** is the smallest addressable unit of an image. It is also known as **pel** or **picture element**. Real world objects are continuous in nature and hence they have infinite samples. When they are converted into an image

and displayed on the monitor screen, only finite samples are taken into consideration. Ultimately, the object on the computer screen is a collection of pixels.

Like the digital image, the monitor screen is also divided into a grid, consisting *n* rows and *m* columns. Each cell on this grid is also referred to as a pixel. Each grey/black cell in the Fig. 1.3.1 corresponds to one pixel.

On zooming in the image, we can get the better idea of the pixel. Each square in the Fig. 1.3.1 represents one pixel.

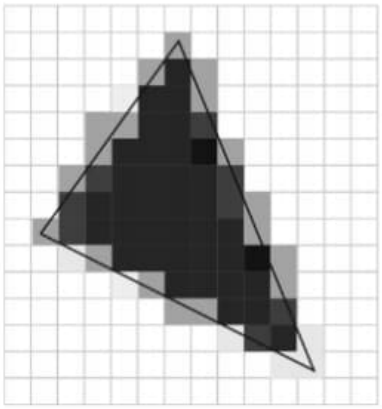


Fig. 1.3.1 : Pixel-wise representation of shape on the monitor screen

- For a grayscale image, pixel value varies from 0 to 255, where 0 represents black color and 255 represents white color. In between range represents the grey shades with varying intensities.
- For a color image, each pixel is a triplet of color (R, G, B) consisting proportion of red, green and blue color. The mixture of these three components determines the final color of the pixel on the monitor screen. Value of each component varies from 0 to 255, resulting in 255 × 255 × 255 = approximately 17 million different shades of color.

1.3.2 Resolution

Q. What is resolution?

Q. Define and explain resolution.

- A maximum number of pixels that can be displayed without overlap on the monitor screen is called screen **resolution**. Typical resolution on high-quality systems is 1280 by 1024. Such high-resolution systems are often referred to as **high-definition** systems.
- Images on the monitor screen are described by its resolution. Image with resolution 800 × 600 means that the image has 800 pixels in each row and 600 pixels in each column. Image contains total 800 × 600 = 4, 80, 000 pixels.
- As resolution decreases, fewer samples will be captured and images quality gets poor. Image with resolution 256 × 256, 64 × 64 and 16 × 16 are shown in the Fig. 1.3.2.

Fig. 1.3.2 : Images with different resolution

1.3.3 Frame Buffer

W-18

Q. Define Frame Buffer. **(W-18, 2 Marks)**

Q. Describe Frame buffer display in computer graphics.

- We can consider the screen picture as two-dimensional grid of pixels. Raster system displays the picture by drawing pixel in a row by row from left to right. Picture definition is stored in a memory called the **refresh buffer** or **frame buffer**.

- Scene to be displayed is first loaded into frame buffer in the form of intensity values. Ideally, the size of the frame buffer is the same as screen resolution. Intensity value from the top left location of the frame buffer is retrieved and painted at a top-left location on the screen. The second pixel of the same row is painted soon after that and this process continues.
- Frame buffer in raster display stores individual pixels of the scene, whereas in case of random scan display, frame buffers stores commands for drawing the scene.
- Suppose a system has resolution 1280 × 1280 and it supports 24 bits per pixel. The memory requirement for the frame buffer is :

Number of pixel = 1280 × 1280 = 16, 38, 400

Number of bits = Number of pixels × Number of bits/pixel

= 16, 38, 400 × 24

= 3, 93, 21, 600 bits

= 39321600/8 = 49,15,200 Bytes (∵ 1 Byte = 8 bits)

= 49,15,200/1024 = 4800 KB (∵ 1 KB = 1024 Bytes)

= 4800/1024 = 4.69 MB (∵ 1 MB = 1024 KB)

- Thus, the system having screen resolution 1280 × 1280 and supporting 24 bits per pixel require 4.69 MB of memory for the frame buffer.
- The frame buffer is known as **bitmap** if it uses one bit per pixel. And it is known as **pixmap** if it uses multiple bits per pixel. A bitmap is used in bi-level system while pixmap is used in a system supporting multiple colors.
- Sometimes additional alpha channel is retained to adjust the transparency of pixel.

1.4 Display Adapters

1.4.1 Video Adapters

Q. Write the properties of video display devices?

Q. What is video adapter? Explain its characteristics in detail.

- A **Video adapter** or **Video card** or **Graphics card** is a plug-in card in a desktop computer that performs graphics processing. It is also known as display card or graphics adapter. Modern display adapters use the PCI Express interface, while earlier cards were using PCI and AGP.
- Maximum resolution supported by system, refresh rate and number of colors displayed on screen – all depends on video adapter used in monitor.
- A monochrome monitor cannot display color no matter how powerful the video adapter.
- Nowadays the graphics circuits are built into the chipset on motherboard, and a separate plug-in card is not required in such case.
- Various graphics adapter cards are available in market like CGA, VGA, IBM. Each adapter offers several different video modes.

Display adapters are generally characterized by following properties :

1. **Maximum Resolution :** Resolution is defined as total number of pixels that can be displayed without overlapping on monitor screen. Sometimes it is expressed by number of dots in horizontal direction by number of dots in vertical direction. Higher resolution gives better visualization of scene. Commonly used resolutions are, 640 × 480, 800 × 600, 1024 × 768 and 1280 × 1024. Screen resolution decides the size of frame buffer.
2. **Color depth :** Color depth is defined by number of bits used to represent the color. Size of frame buffer also depends

on number of color supported by system. System using *n* bit per pixel can support maximum 2^n different colors.

System with resolution 800 × 600, which supports true color system (24 bit/pixel) needs (800 × 600 × 24) / (8 × 1024 × 1024) = 1.37 MB memory for frame buffer. System with same resolution which supports 256 colors requires (800 × 600 × 8) / (8 × 1024 × 1024) = 0.458 MB memory for frame buffer.

3. **Refresh rate :** Refresh rate defines how many number of time screen can be painted in 1 second. Normal refresh rate in raster CRT monitors and color TV set is 60 Hz. Lower refresh rate may produce flickering effect. Different adapter cards may support multiple resolutions with same or different colors. For example, VGA supports 640 × 200, 640 × 350 and 640 × 480 with 16 color support for each mode.
4. **TV Tuner :** A card with a built-in TV tuner can turn PC into a TV set. We can use a traditional antenna or connect the card to cable or satellite system.
5. **Accelerator :** A graphics accelerator is a special type of video adapter that contains its own processor to boost performance. Graphics rendering is the responsibility of accelerator. They are specialized to do such task and so they are much better than the general purpose CPU. CPU doesn't need to render the graphics at all, so they can focus on executing other commands.

Example 1.4.1 : If 8 bit per pixel, per color scheme is used for a RGB display device, how much memory is required to hold picture data worth one screen, if the resolution is 800 × 600. If the refresh rate is 50Hz, how much memory is required to hold picture data for duration of one sec.

Solution :

Total number of pixel = Screen resolution = 800 × 600 = 4, 80, 000 pixels

Each pixel takes 8 bit to represent the color, so total number of bits required to store display information = 4, 80, 000 × 8 = 38, 40, 000 bits

Size of frame buffer in Byte = 38, 40, 000 / 8 = 4, 80, 000 Bytes

Size of frame buffer in KB = 4, 80, 000 / 1024 = 468.75 KB

Size of frame buffer in MB = 468.75 / 1024 = 0.46 MB

To store the picture information of resolution 800 × 600 pixels having 8 bit per pixel, 0.46 MB memory is required.

Refresh rate is 50 Hz, so it paints the screen 50 times in a second. On every cycle, new data is loaded in same buffer. So, size of frame buffer do not alter, it is just reloaded 50 times in a second.

Example 1.4.2 : Assume that certain true color system supports resolution of 1200 × 1024 pixels. Determine the size of frame buffer for the system.

Solution :

True color system uses 24 bit / pixel to represent the color.

Total number of pixel = Screen resolution = 1200 × 1024 = 12,28,800 pixels

Each pixel takes 24 bit to represent the color, so total number of bits required to store display information = 12,28,800 × 24 = 2,94,91,200 bits

Size of frame buffer in Byte = 2,94,91,200 / 8 = 36,86,400 Bytes

Size of frame buffer in KB = 36,86,400 / 1024 = 3,600 KB

Size of frame buffer in MB = 3,600 / 1024 = 3.5156 MB

Example 1.4.3 : How much time is required to scan individual pixel during screen refresh on a raster system with resolution 600 × 400 and refresh rate of 60 Hz. (Assume horizontal and vertical retrace time is negligible).

Solution :

Total number of pixels = Screen resolution = 600 × 400 = 2,40,000

Refresh rate = 60 Hz, so 2, 40,000 pixels are accessed 60 time per second. So number of pixels accessed in one second = 2, 40,000 × 60 = 1,44,00,000.

So time required to access individual pixel = 1 / 1,44,00,000 = 0.0694 microsecond.

Example 1.4.4 : How much time is spent in scanning each pixel, each row and each column during screen refresh on a system with resolution 300 × 200 and a refresh rate of 60 frames per second? (Assume horizontal and vertical retrace time is negligible).

Solution :

Total number of pixels = Screen resolution = 300 × 200 = 60, 000

Refresh rate = 60 Hz, so 60, 000 pixels are accessed 60 time per second. So number of pixels accessed in one second = 60, 000 × 60 = 36, 00, 000

So time required to access individual pixel = 1 / 36,00,000 = 0.277 microsecond

Each row contains 300 pixels,

So time required to access one row = 300 × 0.277 = 83.1 microsecond

Each column contains 200 pixels,

So time required to access one column = 200 × 0.277 = 55.4 microsecond

---

Example 1.4.5 : If transfer rate of system is 10^4 bits / second, what amount of time is require to load frame buffer of size 400 × 300 which supports 256 colors?

Solution :

As system supports 256 colors, each pixel in frame buffer needs 8 bit to represent color.

So, Time required to load buffer = $\frac{\text{Total number of pixels}}{\text{Transfer rate}} = \frac{400 \times 300 \times 8}{10^4}$

= 96 second

Example 1.4.6 : What is the fraction of total time per frame is spent in retrace of electron beam for a non-interlaced raster system with resolution of 600 × 400, a refresh rate of 60 Hz, horizontal retrace time 5 micro seconds and a vertical retrace time of 500 micro seconds.

Solution :

Here, number of rows are 400.

Refresh rate is 60 Hz, so one frame is scanned in 1/60 = 16.67 micro seconds

Refresh time per frame = (Number of rows × Horizontal retrace time) + Vertical retrace time

= (400 × 5) + 500 = 2.5 microsecond

Fraction of total time per frame spent in retrace = (2.5 microsecond) / (16.67 microsecond)

= 0.15 = 15 %

Example 1.4.7 : If a true color display system has 300 scan lines and aspect ratio of 3:4, how many bits per second are required to show 60 frames per second?

Solution :

Number of scan lines = 300. So number of pixel in one column is 300.

Number of pixel in each scan line = (300 × 4) / 3 = 400

So, screen resolution is 400 × 300.

True color system needs 24 bit per pixel to represent color.

So, size of frame buffer = 400 × 300 × 24 = 28, 80, 000 bits = 2.88×10^6 bits

Refresh rate is 60 Hz, so 2.88×10^6 bits are displayed 60 times in one second

So, number of bits required per second = $2.88 \times 10^6 \times 60 = 1.728 \times 10^8$

∴ 1.728 bits per second are required to show 60 frames per seconds

---

Example 1.4.8 : How long would it take to load a 1280 by 1024 frame buffer with 12 bits per pixel if transfer rate is 1Mbps?

Solution :

Each pixel in frame buffer takes 12 bits to represent color.

$$\text{So, Time required to load buffer} = \frac{\text{Total number of pixels}}{\text{Transfer rate}}$$

$$= \frac{1280 \times 1024 \times 12}{1024 \times 1024} = 15 \text{ second}$$

Example 1.4.9 : How long it would take to load a 640 by 400 frame buffer with 12 bits per pixel, If 10^6 bits can be transferred per second?

Solution :

Each pixel in frame buffer takes 12 bits to represent color.

$$\text{So, Time required to load buffer} = \frac{\text{Total number of pixels}}{\text{Transfer rate}}$$

$$= \frac{640 \times 400 \times 12}{10^6} = 3.07 \text{ second}$$

Example 1.4.10 : How long would it take to load a 640 by 480 frame buffer with 12 bits per pixel if transfer rate is 1Mbps? What is the size of frame buffer? How many colors it support?

Solution :

Each pixel in frame buffer takes 12 bits to represent color.

$$\text{So, Time required to load buffer} = \frac{\text{Total number of pixels}}{\text{Transfer rate}}$$

$$= \frac{640 \times 480 \times 12}{1024 \times 1024} = 3.52 \text{ second}$$

Total number of pixel = Screen resolution = 640 × 480 = 3,07,200 pixels

Each pixel takes 12 bit to represent the color.

So total number of bits required to store display information = 3,07,200 × 12 = 36,86,400 bits

Size of frame buffer in Byte = 36,86,400 / 8 = 4,60,800 Bytes

Size of frame buffer in KB = 4,60,800 / 1024 = 450 KB

Size of frame buffer in MB = 450 / 1024 = 0.4395 MB

Frame buffer uses 12 bits per pixels, so supported colors = 2^{12} = 4096 colors

---

Example 1.4.11 : Consider a raster system with resolution of 1280 by 1024. What size of frame buffer is needed for given system to store 24bits per pixel? How many colors are possible in given system? What is the access time per pixel if refreshing rate is 60 frames per second?

Solution :

Total number of pixel = Screen resolution = 1280 × 1024 = 13,10,720 pixels

Frame buffer requires 24 bits to represent the color.

So, total number of bits required to store display information = 13,10,720 × 24 = 3,14,57,280 bits

Size of frame buffer in Byte = 3,14,57,280 / 8 = 39,32,160 Bytes

Size of frame buffer in KB = 39,32,160 / 1024 = 3840 KB

Size of frame buffer in MB = 3840 / 1024 = 3.75 MB

Frame buffer uses 24 bits per pixels, so supported colors = 2^{24} = Nearly 17 million colors

Here refresh rate is 60 Hz, so 1280 × 1024 = 13,10,720 pixels are accessed 60 times per second.

So number of pixels accessed in one second = 13,10,720 × 60 = 7,86,43,200

Time required to access individual pixel = 1 / 7,86,43,200 = 1.27×10^{-8} Second

Example 1.4.12 : If image of size 1024 by 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 image after changing the width from 1024 to 640 with the same aspect ratio is computed as,

$$\text{Height} = 800 / 1024 \times 640 = 500 \text{ pixels}$$

1.4.2 Display Modes

Q. What are the different video display modes?

Operational modes of the adapter card can be categorized either as text/character mode or graphics mode.

1.4.2.1 Text Mode

Q. Explain the working of text mode in brief.

(a) Text mode **(b) Graphics mode**

Fig. 1.4.1 : Operational modes of adapter

In the text mode, screen is internally represented as text instead of Pixel. Screen is divided in a rectangular grid (cell or referred box), each cell is capable to bold a single character only. All video standards support text mode with 25 rows and 80 columns of characters. Text mode has mono-space fonts, means each character has an equal width on screen. So that output can be aligned properly. Screen manipulation in text mode is faster compared to graphics mode and it requires less memory.

1.4.2.2 Graphics Mode

Q. Explain the working of graphics mode in brief.

Graphics mode is a computer display mode that generates an image using pixels. In graphics mode screen is internally represented as a grid of pixels. Pixel is the smallest addressable entity on the screen. Everything including characters is represented in terms of pixels. Graphics mode supports more smooth shapes and fonts. Graphical objects like a circle, line, triangle etc. cannot be displayed in text mode. As per the requirement, the application programmer can switch between text and graphics mode.

1.5 Basic Graphics Pipeline

Q. Write a short note on graphics pipeline.

Q. Explain various stages of graphics pipeline.

Computer graphics is a process of mapping 3D scenes on a 2D screen. **Graphics pipeline** or **rendering pipeline** defines the primitive operations required for this mapping. These operations depend on hardware and software, so various graphics APIs like OpenGL and DirectX were created to unify the underlying rendering process. Such APIs free the application developer from writing the code to manipulate graphics hardware accelerator.

As shown in the Fig. 1.5.1, graphics pipeline primarily works in three stages : Application, Geometry and Rasterization.

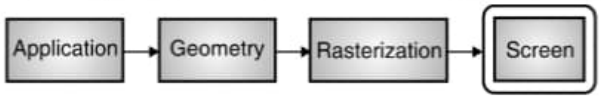


Fig. 1.5.1 : Abstract view of the computer graphics pipeline

1.5.1 Application Stage

At the application level, the userinteracts with the scene and alters it as per his requirements. These changes must be reflected on the screen. The new scene with its primitive shapes like a point, line, triangles are passed to the next stage.

Application stage mostly performs the collision detection, morphing, animation, quadtree or octree generation. Such preprocessing steps help to reduce the memory requirement later stage in the rendering process.

1.5.2 Geometry Stage

This pipeline can be seen from a different perspective. Majority work of operations with polygons and vertices are performed in the geometry stage.

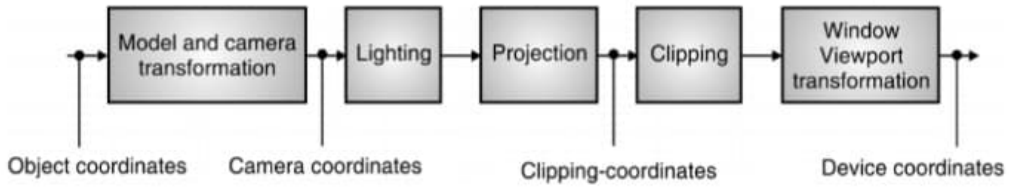


Fig. 1.5.2 : Operations in the geometry stage

Objects are first created in world coordinates with their actual dimensions. Required objects are placed in the world coordinate system. Camera position is determined in a world coordinate system, which determines the view of placed objects. Appropriate light sources are considered to illuminate the scene. Parallel or perspective projection transforms the 3D scene from the real world to 2D representation as per the camera viewing direction. The real world scene which is

outside the clipping window (camera viewfinder) will be rejected to reduce the further computation. The projected and the clipped scene is then mapped to the desired device display device. Window to viewport transformation does this mapping.

1.5.3 Rasterization

Each object is first converted into triangles via the process called **tessellation**. In the later stage, triangles are converted into the smallest fragments called pixels. Entire continuous scene is drawn in a discrete form on a monitor through the process of rasterization. Color of the pixel is determined by the light source, viewing direction, object material property, etc. Finally, the output is merged and displayed on the monitor screen.

The process is depicted in detail in the Fig. 1.5.3.

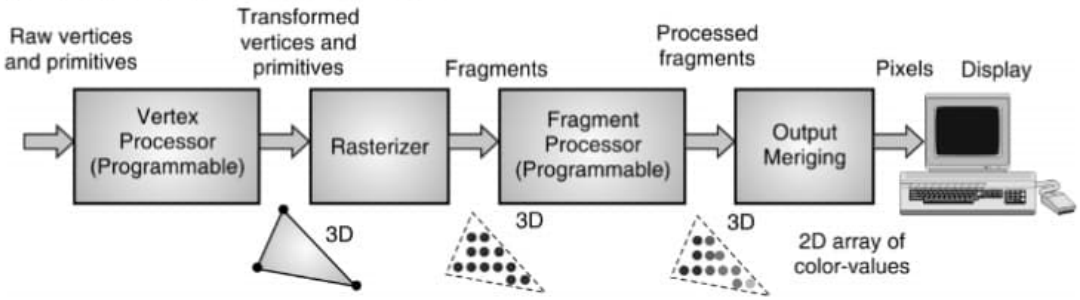


Fig. 1.5.3 : Rasterization process in the graphics pipeline

1.6 Types of Graphics

Q. Explain different types of graphics.

Q. Write short note on bitmap and vector based graphics.

Computer synthesized images are classified in one of the two categories.

1.6.1 Bitmap based Graphics

Q. Explain characteristics of bitmap graphics.

- Bitmap graphics is also known as ***raster graphics***. In bitmap graphics, image is stored using tiny dots, called pixel. The screen is divided intotwo-dimensional array. Each array position corresponds to one pixel. Manipulating bitmap image is simple. We can edit the picture by changing the intensity value of specific pixel locations.
- Bitmap images are resolution dependent. Image size is determined by its resolution.
- Bitmap images preserve the scene by storing individual pixel, whereas vector images are stored using primitive parameters. So vector images are smaller in size than their counterpart.
- Almost all types of graphics packages (such as paint, photoshop, powerpoint etc..) support the creation of bitmap images.
- Bitmap images scale poorly (Refer Fig. 1.6.1 (bottom row)).
- Bitmap images are good at representing realistic images as they allow to access and manipulate individual pixel.

1.6.2 Vector based Graphics

Q. Explain characteristics of vector graphics.

- Vector graphics represent the scene using mathematical formulas and necessary parameters. Typically, such images are displayed using basic primitives like points, line and curves.

- Vector based images are resolution independent and hence scale well. If we scale bitmap graphics beyond a certain scale, shape border then it begins to convert into zig-zag pattern. In the vector image, on scaling, the entire image is redrawn with scaled primitive parameters. So vector image can easily sustain the scale change.
- Scaled image is as smooth as the original one. But unfortunately, vector images are not well suited for web applications.
- Vector images can be generated in higher level graphics packages like (Photoshop, Coral Draw, Adobe Illustrator etc.)
- Bitmap images are well suited for the applications which need to display a wide range of colors. On the other hand, vector images are better for the images having few areas of solid colors. Logos and fonts are often designed as vector graphics.

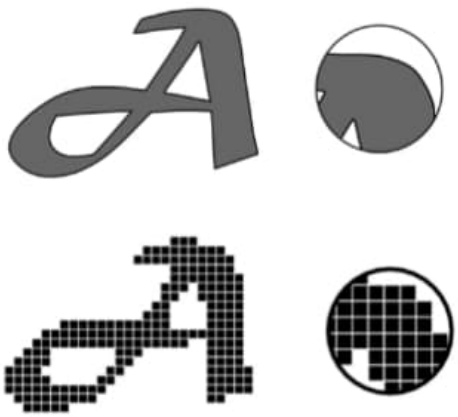


Fig. 1.6.1 : Vector graphics (top row) vs Bitmap graphics (bottom row)

- Fig. 1.6.1 shows the difference between vector and bitmap display. Top and bottom row shows zoom in effect for vector and bitmap images respectively.

1.6.3 Bitmap vs. Vector Graphics

W-18

Q. Compare Bitmap Graphics and Vector based graphics. **(W-18, 4 Marks)**

Q. Differentiate between bitmap and Vector based Graphics.

Sr. No.	Bitmap Graphics	Vector Graphics
1.	Represents scene using pixels.	Represents scene using mathematical formulas.
2.	Bitmap graphics is resolution dependent.	Vector graphics is resolution independent.
3.	Bitmap images scale poorly.	Vector images are good at scaling.
4.	Good for representing realistic scenes.	Good for the images having large area of constant colors.
5.	On scaling, border starts appearing zigzag.	Sustains the change in scale.
6.	Such images are suited for web applications.	Not suited for web applications.
7.	Requires more memory compared to vector graphics.	Requires less memory compared to bitmap images.
8.	Modification is difficult.	Modification is easier.
9.	Conversion from bitmap to vector graphics is difficult.	Conversion from vector to bitmap graphics is easy.
10.	Examples : JPEG, GIF, BMP etc.	Examples : True type fonts, Postscripts, Logo etc.

1.7 Applications of Computer Graphics

Q. State and explain the applications of computer graphics.

Computer graphics has its root almost everywhere. In this section, we will discuss a few applications which are used in day to day life.

1. **Computer Aided Designing :** One of the fundamental applications of computer graphics is designing and modelling of objects, which is known as Computer-Aided Design (CAD). CAD methods are common in designing models of machines, buildings, crafts, computers, automobiles textiles, and many other products. CAD packages allow designers to make wireframe or solid models of components. Wireframe display gives flexibility to designers to quickly see the effects of changes in shapes and adjust it.

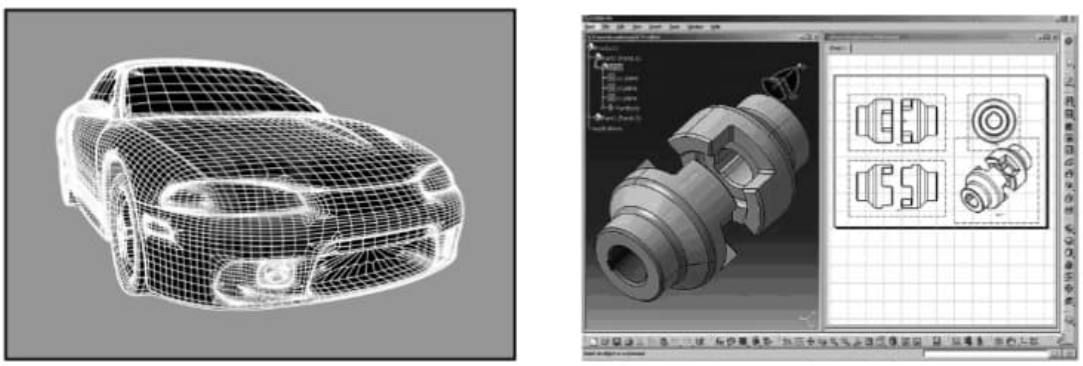


Fig. 1.7.1 : Role of computer graphics in Computer Aided Designing

Software packages for CAD applications also provide the designer with a multi-window environment which can display different views of objects. Standard shapes for various logical circuits are also often supplied by the electronic circuit design packages. Animations are often used in CAD applications. Real-time wireframe display is useful for testing the performance of a vehicle or system. Also, wireframe displays allow the designer to see into the interior of the vehicle and to watch the behaviour of inner components during motion. Wireframe design of the car and multi-window view of the object are shown in the Fig. 1.7.1.

Fig. 1.7.2 : Virtual vehicle navigation system

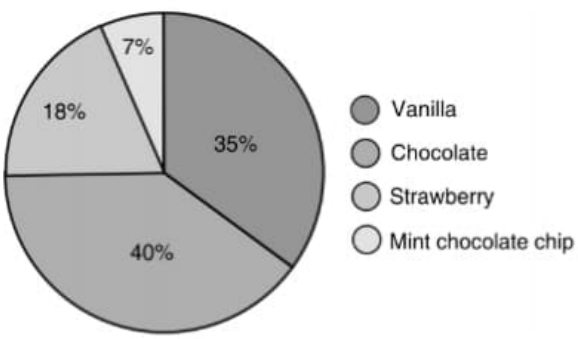


Fig. 1.7.3 : Graphical representation of the survey

2. **Virtualreality :** With the help of computer graphics, it is possible to simulate various virtual environments which look seamlessly similar to the realworld. Such environments help to train the person without providing them with physical resources. Doctors can be trained to operate a patient without operating on the actual physical body. The pilot can be trained to fly the aircraft without physically seating in the craft. One such vehicle navigation simulator is shown in the Fig. 1.7.2. It is also possible to have simulated "walk" in and around the home using virtual reality.

3. **Presentation Graphics :** Another major application of computer graphics is presentation graphics. It is used to produce diagrams, tables and charts for reports or presentations. Presentation graphics is commonly used to summarize research or survey data. 3D graphs are sometimes used to provide multi-angle analysis of statistical data. They provide a more intuitive and more attractive presentation of data relationships.

4. **Computer Art :** Computer graphics plays a major role in fine art and commercial art. Various computer methods, symbolic packages, software packages, CAD packages, etc. are used by the designer to generate computer art. Using such packages, the artist can easily design shapes and can add motion. Morphing is the common graphics processing method used in animation. It is a process of transforming one object into another. Morphing is being used in movies to transform one shape to others, e.g. human to tiger, car or any random shape etc.

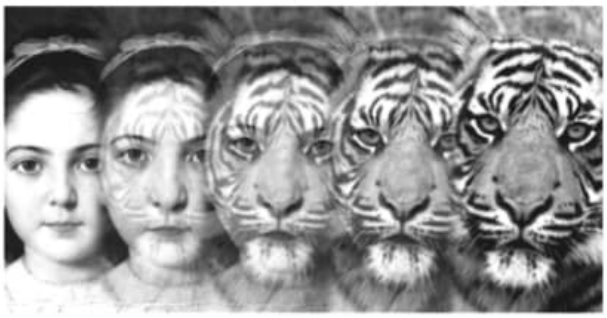


Fig. 1.7.4 : Morphing

5. **Entertainment :** Computer graphics is commonly used in film industries for making animation and cartoon movies, for providing special effects to the scene, television shows etc. Sometimes the character itself is displayed as graphics object, and sometimes it is combined with the real scene and synthesized objects.

Fig. 1.7.5 : Use of computer graphics in animation and movie

6. **Education and Training :** Another promising application area of computer graphics is education and training. Nowadays in the market, many educational toolkits are available for child education. They can learn through videos and animations. Thousands of simulation software are available to understand the behaviour of certain processes or environment or response of the system. Training a person for vehicle driving using such simulators saves his time and efforts.

Fig. 1.7.6 : Scope of computer graphics in training

7. **Visualization :** Data produced from many research applications is too big to analyze on paper. Representing data in form of graphs, charts or in some other visual form may help to understand the results in quick time. Colour coding can also provide a clue to understanding the data. For example, plain satellite image may not be as useful as if it is segmented using different colors, green indicating forest region, blue indicating water region, black indicating mountains and roads, grey indicating soil etc.

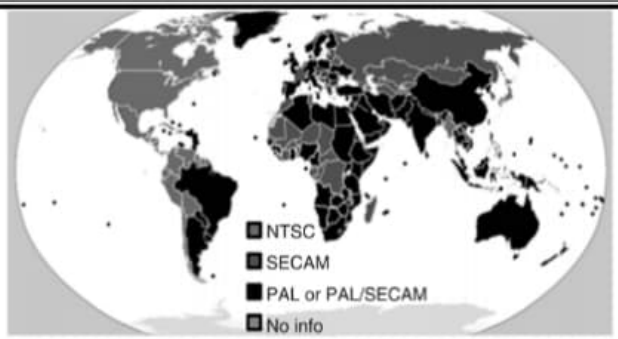


Fig. 1.7.7 : Satellite images with augmented data

8. **Graphical User Interface :** The old computer systems were mostly command-based systems. The user had to write commands to perform any task. But with the introduction of computer graphics, things have changed. People get more interested in the graphical user interface. It is very obvious that nobody would like to remember or write commands. GUI provides an easy to use interface, with more accuracy and quick access time. GUI packages allow to design and interact with multiple windows like menu-based selection, mouse and keyboard driven commands, zooming of objects etc. simultaneously.

Fig. 1.7.8 : Graphics User Interface

1.8 Display Devices

Display devices are output devices used to display information on the monitor, television set or on some other media. The scene is rendered on screen by applying appropriate voltage. The information displayed on video display devices is transient. It is displayed for some time and then erased.

1.8.1 Cathode Ray Tube (CRT)

Q. Write a short note on Cathode Ray Tube.

- Cathode ray tube (CRT) is a common technology for traditional monitors and television sets. Functional diagram of CRT is shown in Fig. 1.8.1.
- As shown in the Fig. 1.8.1, when current is supplied to the filament, electrons are produced and they get dispersed all around. Electrons generated from electron guns fly in arbitrary directions. Focusing anode focuses all dispersed electrons in a beam called an **electron beam**. For higher precision, additional focusing hardware is used to keep the beam in focus at all screen positions.
- The monitor screen is generally curvature in shape, the distance of the all screen pixels from the electron gun is not the same. Electron beam focuses properly only at the center of the screen. As the beam moves away from the center i.e. near to the edges, distance increases and the scene on screen become blurred. To overcome this effect, the system can adjust the focusing according to the screen position of the beam.
- All the fired electrons form a beam start flowing towards phosphorus screen, which is exactly behind the glass of the monitor screen. A positively charged anode next to the grid provides acceleration to these electrons. Electrons are negatively charged and the anode is positively charged, so they get attracted towards the anode and get acceleration.

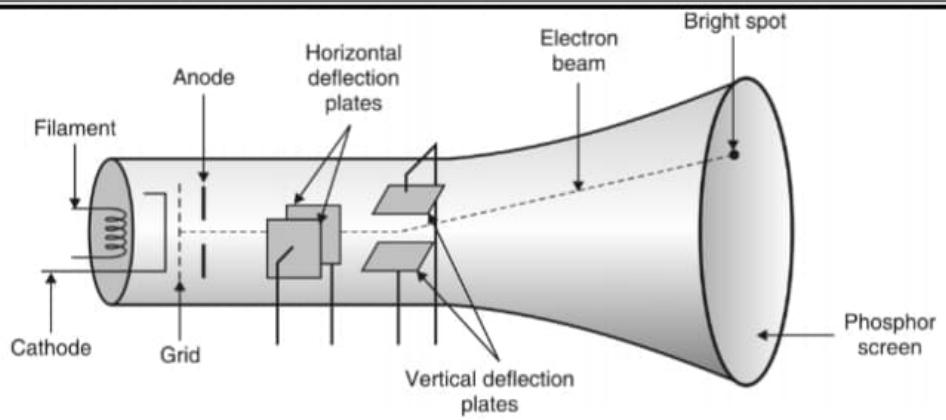


Fig. 1.8.1 : Functional diagram of CRT

- The beam of electrons in acceleration collides with phosphorus screen and their kinetic energy is absorbed by the screen. This action moves the electrons into their higher quantum energy level and they become unstable. According to the quantum phenomenon, electrons release the energy and try to become stable. This energy appears as an illuminated bright spot on the screen, which we call pixel.
- In RGB monitors, there are three layers of phosphorus behind screen, Red, Green and Blue. So color and intensity of illuminated pixel depend on how deep electron beam penetrates a particular layer and what amount of electrons were generated from electron guns.
- When electrons become stable, they stop releasing energy and after a few milliseconds pixel goes off from the screen. To persist the scene on the screen, it is essential to keep redrawing all the pixels repeatedly on the monitor screen. This process is called **refreshing** and such a display system is called **refresh CRT**.
- Apart from color, the main difference between phosphors is their persistence rate. Persistence rate defines how long pixel continues to emit light after the CRT beam is removed. The time it takes the emitted light to decay to one-tenth of its original intensity is called **persistence** of phosphor. If phosphor has a lower persistence rate, it requires a higher refresh rate to maintain a picture on the screen without flicker.
- Animation requires quick erase and redrawing of the scene. So low presentence phosphor is preferred in such applications. High-persistence phosphor is useful for displaying highly complex and static pictures. Usually, graphics monitors use phosphorus with persistence in the range from 10 to 60 microseconds.
- Brightness or intensity of a pixel is directly proportional to the amount of electrons colliding on phosphorus layers. We can control the amount of electrons by controlling the width of the control grid.
- Control grid acts as a gate for electrons. On applying high negative voltage, control grid shut off the beam by repealing each other and not allowing electrons passing through it means the gate is closed for electrons. On applying smaller negative voltage, control grid open ups the gate and more electrons would pass through it.
- If the movement of the flowing electron beam is not controlled, it continuously keeps hitting the center pixel of the screen. The energy released by electrons cumulated at a single point, and very soon the phosphorous layer will burn out at that location.
- The position of the colliding beam is controlled either by electric fields or by magnetic fields. Cathode-ray tubes are constructed with magnetic deflection coils which are mounted outside of the CRT. Electrostatic deflection system uses two pairs of parallel plates mounted inside the CRT (as shown in Fig. 1.8.1). In both cases, the pair of coils is used to control the movement of the electron beam in the horizontal and vertical direction.
- One set of plate controls the horizontal deflection of the beam and another set of plate controls the vertical deflection. Using such a focusing system, we can make an electron beam to collide with each and every pixel location on the screen to illuminate the entire scene at a time.

- A maximum number of pixels that can be displayed without overlap on a screen is called **screen resolution**. Typical resolution on high-quality systems is 1280 by 1024. Such high-resolution systems are often referred to as **high-definition systems**.
- Another property of video monitor is **aspect ratio,** which is the ratio of a number of vertical pixels to the number of horizontal pixels required to draw the same length of a line in both the direction. An aspect ratio of 9/16 means that a vertical line plotted with 9 points has the same length as a horizontal line plotted with 16 points. Generally, the distance between two vertical pixels is more than a distance between two horizontal pixels.

1.8.2 Raster Scan Display

W-18

Q. Explain Raster Scan. **(W -18, 2 Marks)**

- We can consider the screen picture as two-dimensional grid of pixels. Raster system displays the picture by drawing pixel in a row by row from left to right. Picture definition is stored in a memory called the **refresh buffer** or **frame buffer**.
- Scene to be displayed is first loaded into frame buffer in the form of intensity values. Ideally, the size of the frame buffer is the same as screen resolution. Intensity value from the top left location of the frame buffer is retrieved and painted at a top-left location on the screen. The second pixel of the same row is painted soon after that and this process continues.
- After plotting all pixels in the first row, the electron beam moves to the first pixel of the second row and paints all pixels in the second row on screen by retrieving intensity values in horizontal order. Fig. 1.8.2 describes the working principle of the raster scan system.

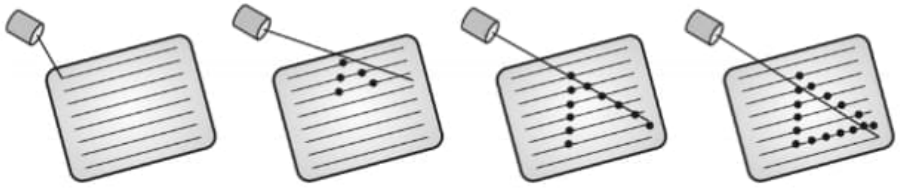


Fig. 1.8.2 : Scene rendering using a raster scan system

- After finishing each scan line (row), electron guns are turned off and moved back to the first pixel of the next scan line. That horizontal movement is called **horizontal retrace**. After reaching to the last pixel of the screen, electron guns are moved at first pixel of the first row. That vertical movement of electron guns is called **vertical retrace**.
- The downside of the raster system is that it scans the entire screen even if we want to display a few pixels, as shown in Fig. 1.8.2. It checks for each and every location in the frame buffer to turn *on* or *off* the electron guns. Pixel is the smallest unit that can be addressed and displayed. Raster scan displays are well opted for realistic scene due to its capability of storing intensity for each pixel. Intensity range for pixel depends on the capability of the raster system.
- A system which supports 1 bit per pixel can display only black and white scene, called *bi-level system*. A bit value of 1 indicates that the electron beam should be *on* for that location, and *off* otherwise. With n bits per pixel, the system can show maximum 2^n different colors /intensities. Current color CRT system provides 24 bits per pixel, which supports 2^{24} (approximately 17 million) colors.

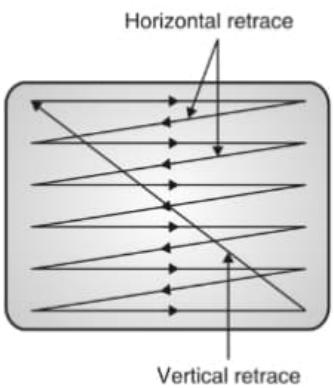


Fig. 1.8.3 : Horizontal and vertical retrace

- Suppose a system has resolution 1280 × 1280 and it supports 24 bits per pixel. The memory requirement for the frame buffer is,

Number of pixel = 1280 × 1280 =16, 38, 400

Number of bits = Number of pixels × Number of bits/pixel

= 16, 38, 400 × 24

= 3, 93, 21, 600 bits

= 39321600/8 = 49,15,200 Bytes (∵ 1 Byte = 8 bits)

= 49,15,200/1024 = 4800 KB (∵ 1 KB = 1024 Bytes)

= 4800/1024 = 4.69 MB (∵ 1 MB = 1024 KB)

- Thus, the system having screen resolution 1280 × 1280 and supporting 24 bits per pixel require 4.69 MB of memory for the frame buffer.
- The frame buffer is known as **bitmap** if it uses one bit per pixel. And it is known as **pixmap** if it uses multiple bits per pixel. A bitmap is used in bi-level system while pixmap is used in a system supporting multiple colors.
- In raster display, the screen is refreshed nearly 60 times per second to display picture without flickering. Rate of the refreshing screen in one second is called **refresh rate**. The refresh rate is measured in frames per second or in Hertz. If the refresh rate is set to less than 60, the screen starts flickering. It may happen that while the bottom part of the screen is being painted, the upper part may wipe out due to lower refresh rate.

1.8.3 Random Scan Display

- In a random scan system, the electron beam directly follows the part of the screen where the picture is to be displayed. Instead of an individual pixel, the picture is stored in terms of line drawing commands, and for this reason, it is also referred to as **vector displays** *or* **stroke-writing** or **calligraphic displays**.
- The memory area where picture definition is stored is referred to as a **refresh display file**. Sometimes it is also called **display list, display program**, or **refresh buffer**. In random scan displays, display processor reads line drawing command one by one from the display list and draws it. Fig. 1.8.4 shows the rendering of a polygon on the screen. Electron beam only follows the border of polygon instead of scanning each scan line and deciding which pixel should be turn on or off.

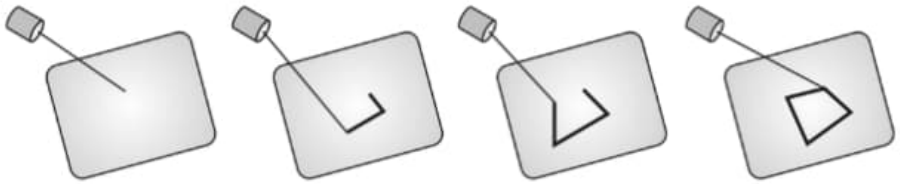


Fig. 1.8.4 : Scene rendering using random scan system

- A pen plotter is the example of a random-scan device. The refresh rate on a random-scan system depends on the number of lines to be displayed. High-quality vector systems are capable of handling near about 10^6small lines at the refresh rate of 60 Hz. When frame buffer contains a small set of lines, each refresh cycle is delayed to avoid refresh rates greater than 60 Hz. Otherwise, faster refreshing could burn out the phosphor.
- Random-scan systems are useful for line drawing applications and are not suitable to display realistic shaded scenes. Random displays produce smooth line because the CRT beam directly follows the line path, while raster system produces jagged lines. Random scan devices have generally limited color capability.

Advantages of Random Scan :

1. Good choice for a device with very high resolution.
2. Requires less memory.
3. Produces smooth lines.
4. Useful in displaying static drawing.

Disadvantages of Random Scan :

1. Cannot handle the complex or natural scene.
2. Supports limited color only.
3. Such devices are costly.
4. Not useful to show animation.

1.8.4 Raster scan vs. Random Scan Display

W-18

Q. Differentiate between Random Scan and Raster Scan. **(W-18, 4 Marks)**

Sr. No.	Raster Scan System	Random Scan System
1.	The electron beam scans the entire screen to draw a picture.	The electron beam scans only the part of the screen where picture information is present.
2.	Due to the discrete nature of the system, it has low resolution.	Electron beam directly follows the path of lines, it draws smooth lines, which gives better resolution.
3.	Picture definition is stored as a set of discrete intensity values in the frame buffer.	Picture definition is stored as a line drawing commands in the display list.
4.	The intensity value is stored for each pixel, it is well suited to display realistic scene.	The system is designed to display lines, so it cannot be opted to display a realistic scene.
5.	Pixel / spatial location of the screen is used to draw an image.	Mathematical functions are used to draw an image.
6.	The refresh rate is independent of a number of objects in the scene.	When a number of primitives are too large, random scan device flickers.
7.	Scan conversion is required.	Scan conversion is not required.
8.	Scan conversion hardware is required as it displays real-time dynamic scene with extensive computation.	Scan conversion hardware is not required.
9.	Such displays are economical.	They are more costly.
10.	Used to display a dynamic scene.	Used to display static picture.
11.	The video controller is required.	The video controller is not required.

1.8.5 Interlaced Display System

Q. Write a short note on interlaced display system.

Q. How to achieve higher refresh rate on old hardware?

– On some raster-scan systems and in TV sets, each frame is displayed in two passes using an **interlaced refresh** procedure. The scene is displayed using half-frames, called **fields**. What we discussed in the previous section is called progressive display, in which system display all scan lines one by one. In the interlaced display, all odd scan lines are displayed first, followed by all even scan lines.

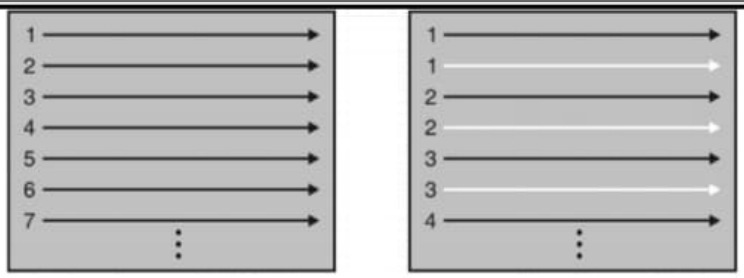
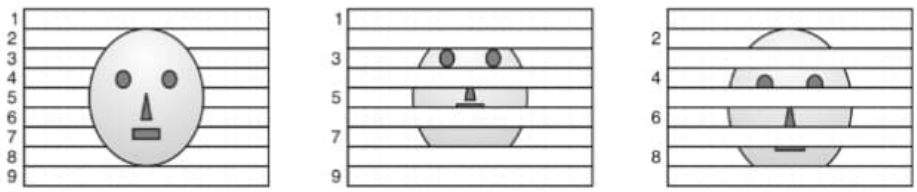


Fig. 1.8.5 : Displaying using progressive and interlaced scan line

- Each frame contains exactly two fields. **The odd field** is made up of the odd horizontal scan lines in a frame. The odd field is also called the **top field** since it contains the top line of the image. After the vertical retrace, the beam paints out the entire remaining even scan lines called **Even field**. Even the field is called the **bottom field**. Fig. 1.8.5(a) and 1.8.5(b) shows the working of the progressive and interlaced display.



(a) Progressive display **(b) Interlaced display**

Fig. 1.8.6 : Progressive vs interlaced display

- Interlaced display overcomes the limitation of the device having slower refreshing rates. On an older system with a refresh rate of 30 Hz, some flicker was noticeable. But by employing the interlaced technique, logically we can double the refresh rate by accomplishing each pass in 1/60 second. This approach makes an effective refresh rate nearer to 60 Hz. This is an effective and efficient way of avoiding flicker.

1.8.6 Flat Panel Display

Q. What is flat panel display?

Most of the graphics monitors are built using CRTs. The downside of CRT is their size and power consumption. Due to the horizontal arrangement of electron guns, CRT monitors are very thick. Flat panel displays are thin and lightweight. Even they require less power compare to CRT. We can hang large devices on the wall and small devices are wearable.

Tablets, digital pads etc. allow users to write on it. Nowadays, flat panel is the first choice for laptop, television set, advertisement boards, displays at commercial blocks etc.

Flat panel displays can be classified into two categories, emissive display and non-emissive display :

1. **Emissive display devices :** They convert electrical energy into light to generate a display, for example, LED (Light Emitting Diodes), plasma panels, electroluminescent displays etc. Flat CRT is an improved version of CRT, in which electron beams are accelerated parallel to screen rather than perpendicular. Then they are deflected 90 degrees. Although such flat CRT reduces the volume, they are not as successful as other emissive display devices.
2. **Non-emissive display devices :** They use the optical effect to covert natural or synthetic light to the graphics scene on screen. Example of such a device is LCD (Liquid Crystal Display).

Let us see the working principle of few flat panel devices.

1.8.6.1 LED Display

Q. Explain working of LED display.

- Light Emitting Diode (LED) display is a matrix of diodes arranged to form the pixel positions in the display. Picture definition is read out from refresh buffer and LEDs on appropriate locations are lit up.
- The LCD display gives the best vision when viewing direction is perpendicular to the screen. In LCD, quality of vision reduces with angle. LED displays can sustain the view of angle up to more than 80 degrees. And that is why LED is the first choice for advertisement hoardings and monitors.

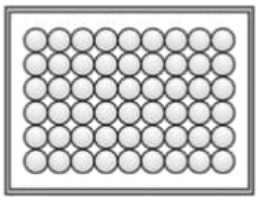
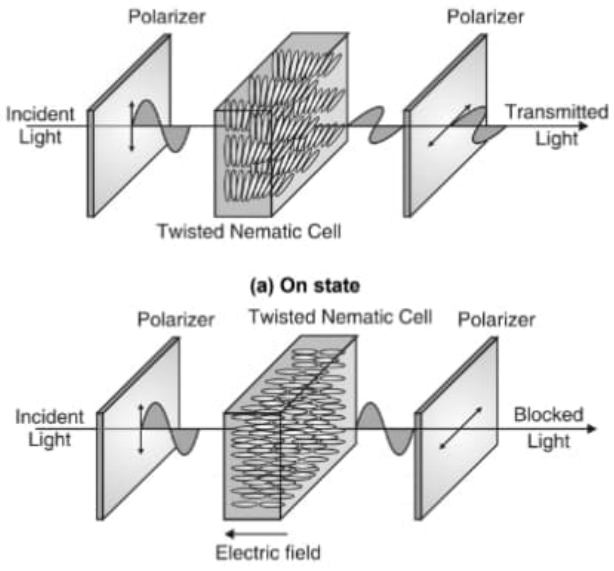


Fig. 1.8.7 : Arrangement of the LED grid

1.8.6.2 LCD Display

Q. Explain the working of LCD Display.

- Liquid Crystal Display (LCD) is used with small devices like laptops, calculators, television sets. LCD is a non-emissive display device.
- It uses the concept of polarization to generate a picture. A polarizer is the material which allows passing the light from it having a certain direction only. Light with remaining angle is blocked.
- Grid of crystalline cells called nematic cells are used to design LCD. Two light plates, with polarizer, are arranged at a right angle, opposite to each other. Horizontal conductors and vertical conductors are placed on each plate respectively. The intersection of both plates defines the screen resolution and hence pixel location.



(a) On state

(b) Off state

Fig. 1.8.8 : Working of liquid crystal display

- In *on state*, nematic cells are arranged as shown in Fig. 1.8.8(a). Cell twists the light so that it can pass through the opposite polarizer. The reflected light reaches the eye of the viewer and scene would be visible. In the *off state*, cells are aligned as shown in Fig. 1.8.8(b). In the *off state*, cell does not twist the light and so it cannot pass through the opposite polarizer.

– Such a flat panel device is called **passive matrix LCD**. Another way is to use a transistor at each pixel location is TFT? (Thin Film Transistor) technology. Transistors control voltage to decay the charge out of the nematic cell. Such an arrangement is called an **active matrix LCD**.

1.8.6.3 Plasma Panel

Q. Explain the working of Plasma Panel.

– Plasma panels are made up of glass plates filled with a mixture of gases like neon. They are also called **gas discharge displays** due to their characteristic component. Numbers of vertical and horizontal conducting plates are placed on opposite glasses.

– When voltage is applied to conducting plates, gas at the intersection of two conductor's glows ups the plasma. The device reads the picture definition from refresh buffer. Pixels are separated by providing an electric field to the conductor plates.

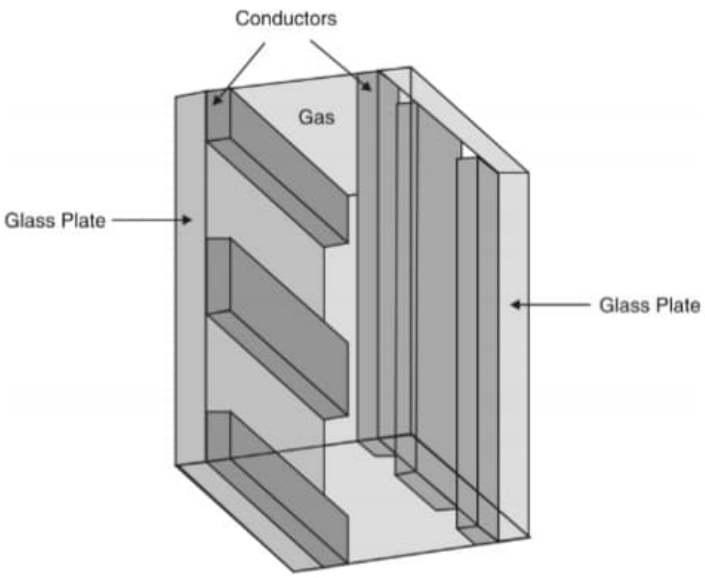


Fig. 1.8.9 : Plasma Panel

Disadvantages :

Strictly monochromatic.

1.8.6.4 Touch Screen

Q. Explain the working of Touch Screen.

– A touch screen is a visual display that the user can control through single or multi-touch. Some touch screens can also detect objects such as a stylus or ordinary or specially coated gloves. User can interactively provide input to control the display. For example, a user can zoom in or rotate the photograph with finger touch.

– User can directly interact with the touch screen without using a mouse, touchpad, or any other intermediate device. The touch screen is more popular in tablets and mobile devices, ATM, PNR status checking units at the railway station etc. uses the touch panels for quick navigation.

1.9 Output Primitives

Q. What is output primitive? Discuss various output primitives with its attributes.

– **Output primitives** are the basic shapes such as point, line, triangle, circle, a polygon which can be used directly by the application programmer to describe any picture. In most of the graphics packages, a basic set of such output primitives is available. Any complex scene can be drawn with the help of basic primitives. We can consider output primitives as building blocks for scene generation on screen.

– Primitives themselves can be manipulated by specifying appropriate attributes.

  - Primitives define what to draw
  - Attributes define how to draw.

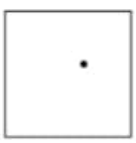
- For example, the line is the primitive which joins the two points, and the width and style are the attributes of line.
- Set of primitives and description of them differs from graphics package to package or standard to standard.
- Some of the most commonly used primitives are discussed here.

Note : Syntax of primitive and its attributes is generalized for easy understanding. It does not correspond to any programming language.

1.9.1 Point

1. Syntax :

Put Pixel(x, y) : Draws a single pixel on monitor screen at specified (x, y) position



2. Attributes : size, color

- SetPixelSize(n) : draw pixel of size n ×n where n = {1, 2, 3, ...}
- SetPixelColor(c) : Set the color of pixel to color c = {RED, GREEN, YELLOW, ...}

1.9.2 Line

1. Syntax :

- Line(x_1, y_1, x_2, y_2) : Draw a straight line between pixels (x_1, y_1) and (x_2, y_2)
- Line To(x, y) : Draw a straight line from current cursor position to (x, y)

2. Attributes :

- SetLineWidth(n) : Draw a line of width n = {1, 2, 3, ...} pixels
- SetLineType(style) : Draw a line with different styles = {solid, dotted, dashed, ...}
- SetLineColor(c) : Set the color of line to c = {RED, GREEN, YELLOW, ...}

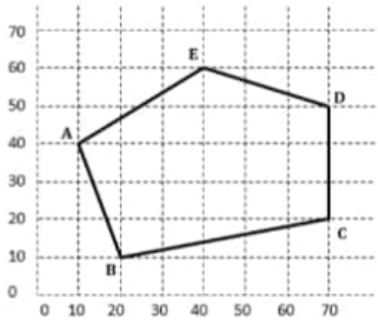
1.9.3 Polygon

1. Syntax :

- DrawPoly(A[2][n]) : Draw a polygon joining the vertices in sequence specified by 2D array A. Last vertex is joined back with first vertex to form closed polygon.
- Any random polygon including basic shapes like triangle, square, rectangle can be drawn by specifying an appropriate set of vertices in an array.
- A.A polygon with its vertices coordinates is shown in the Fig. 1.9.1.

Vertex	X	Y
A	10	40
B	20	10
C	70	20
D	70	50
E	40	60

(a) Polygon vertices



(b) Polygon

Fig. 1.9.1

1. Attributes :

- SetPolygonStyle(p) : Fills the polygon with various styles = {solid, hash, square, dots, ...}
- SetPolygonColor(c) : Sets the polygon boundary color to c = {RED, GREEN, YELLOW, ...}
- SetFillColor(c) : Fill the polygon with color c = {RED, GREEN, YELLOW, ...}

1.9.4 Marker

1. Syntax :

DrawMarker(A[2][n]) : This is similar to DrawPoly() function, but replaces the pixels along polygon boundary with the specified marker style.

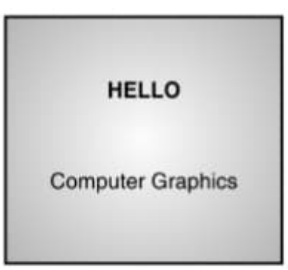
2. Attributes :

- SetPolyMarkerStyle(m) : Sets the marker style to m = {*, #, •, ...}.
- SetPolyMarkerSize(n) : Sets the size of marker to n = {10pt, 12pt, 14pt, ...}
- For example, SetPolyMarkerStyle('•') replaces the pixels on the polygon boundary by the symbol• with specified size.

1.9.5 Text

1. Syntax :

PutText(x, y, String): Paints the text specified by variable String starting from position (x, y).



2. Attributes :

- SetTextColor(c): Set the color of text to c = {RED, GREEN, YELLOW, ...}
- SetTextSize(n): Set text size to n = {10pt, 12pt, 14pt, ...}
- SetTextFont(f): Set text font to the f = {Times, Verdana, Helvetica, ...}
- SetTextStyle(s): Set text style to s = {Bold, Italic, Underlined, Strikethrough, ...}

1.10 Graphics Functions and Standards

Graphics software is classified into two categories :

1. **General programming packages :** They are like programming languages; they contain a rich set of graphics functions. Such graphics packages allow to draw some primitives, fill the colors, set the intensity, apply transformation etc. Open GL, Open CV is the examples of such packages. They are more of programmer oriented and provide more flexibility compared to special purpose application packages. Use of such packages requires good programming skill.
2. **Special-purpose applications packages :** By contrast, application graphics packages are designed for non-programmers, so users can create required graphics without worrying about underlying concepts. The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. Examples of such application packages are painted, CAD etc. Such package often provides a set of drag and drop components, which free the user from writing codes for basic building blocks.

1.10.1 Coordinate Representation

Q. What is the need of coordinates systems? Explain various coordinate systems.

- The coordinate system determines how and where to display objects. Multiple coordinate systems can work in parallel.
- Objects can be specified using the world coordinate system whereas object displayed on display device is specified in device coordinate. Viewing coordinate system specifies the position of the viewer.
- Generally, graphics packages are designed to be used with a Cartesian coordinate system. Special-purpose packages may allow using another coordinate system.
- In general, several different Cartesian reference frames are used to construct and display a scene. We can create objects, such as a pen or chair, with their actual dimensions, called **modelling coordinates**, or **local coordinates** or **master coordinates** i.e. (x_{mc}, y_{mc}). Once the object is designed, we can place it into an appropriate position in the scene using **world coordinates** i.e. (x_{wc}, y_{wc}).
- The scene may be displayed on different devices, so it must be mapped to the proper dimension to render it properly. World coordinates are then represented in **normalized coordinate** (x_{nc}, y_{nc}), usually between 0 and 1. This makes the system independent of the various display devices.
- These normalized coordinate can be easily scaled to fit on any size of display by multiplying it with a proper scaling factor that is known as **device coordinate** (x_{dc}, y_{dc}). Fig. 1.10.1 shows sequence of coordinate transformations.

$$(x_{mc}, y_{mc}) \rightarrow (x_{wc}, y_{wc}) \rightarrow (x_{nc}, y_{nc}) \rightarrow (x_{dc}, y_{dc})$$

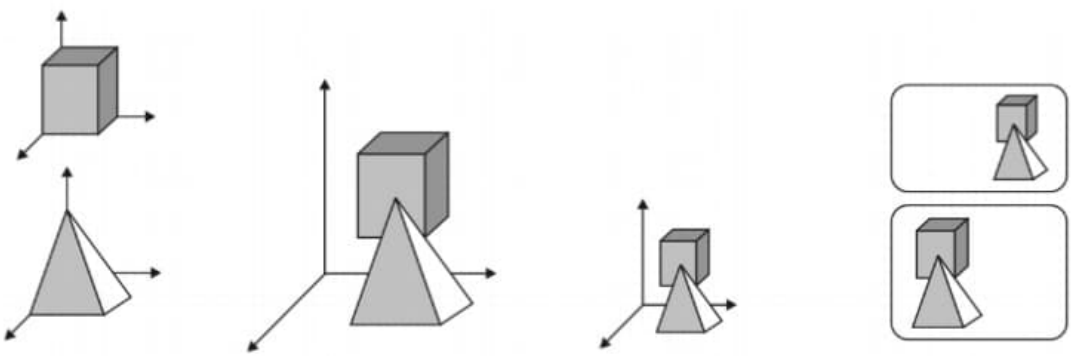


Fig. 1.10.1 : Coordinate transformation

– If normalized coordinates are mapped into a square area of the output device then only appropriate scale and aspect ratio are maintained. On non-square display device, the shape of object deforms.

1.10.1.1 Cartesian Coordinate System

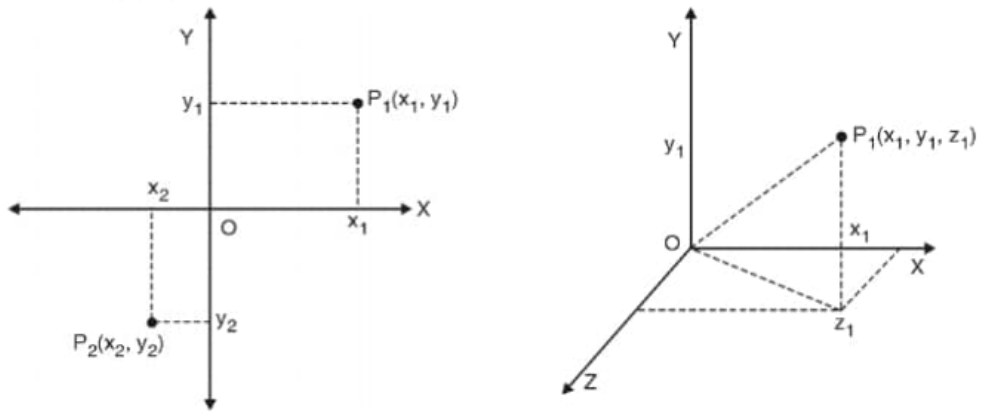
Q. Write a short note on Cartesian coordinate system.

– The cartesian coordinate system uses spatial location in plane or space to represent point uniquely. To represent a point in the plane, we need two-dimensional coordinate systems, which represents point using (x, y) pair.

– And to represent a point in space, we need three-dimensional coordinate system, which represents point using (x, y, z) triplet. Axes in the Cartesian coordinate system are orthogonal to each other.

– Various shapes like circle, ellipse, line etc. can be represented in a Cartesian coordinate system using a mathematical formula. The cartesian coordinate system is also known as a **rectangular coordinate system**.

– The cartesian coordinate system specifies the physical dimensions of objects. By conventions, X, Y and Z axis represents the length, height and depth respectively.



(a) 2D coordinate system **(b) 3D coordinate system**

Fig. 1.10.2 : Cartesian coordinate systems

1.10.1.2 Polar Coordinate System

Q. Write a short note on Polar coordinate system.

– The polar coordinate system is also known as a **circular coordinate system**. It uses the distance from origin and angle with the horizontal axis to define the coordinates of the point.

– Distance from the origin is called radius. Anti-clockwise direction is considered as the positive angle. The angle is measured either in degree (0° to 360°) or in radians (0 to 2π). Fig. 1.10.3 shows the polar representation of point P(x, y) in the 2D coordinate system.

– From Fig. 1.10.3, point P makes an angle θ with X-axis and distance of P from the origin is r. From the trigonometric rule, coordinates of P is given as,

$$\cos\theta = \frac{x}{r}$$

$$\sin\theta = \frac{y}{r}$$

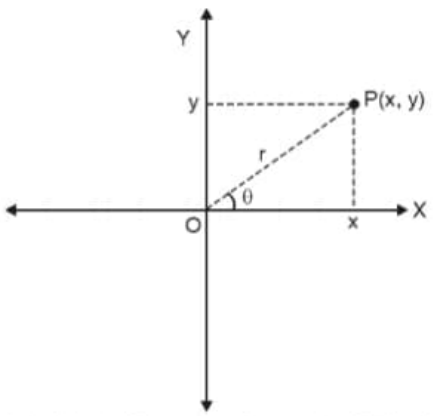


Fig. 1.10.3 : Polar representation of point (x, y)

So, $x = r \cos\theta$

$y = r \sin\theta$

- This coordinate system is useful in describing motion in objects and the human body. Objects and body are made up of joints and motions are best described by polar coordinates.

1.10.1.3 Spherical Coordinate System

Q. Write a short note on Spherical coordinate system.

- The 3D polar coordinate system is also known as a spherical coordinate system. The coordinate of the point is parameterized by two angles and distance from the origin. Angle with x-axis is called **Azimuth** and angle with Y axis is called **Elevation**.
- Fig. 1.10.4 shows the spherical coordinate system. The angle in counter clockwise direction is considered positive.
- Spherical coordinates are given as,

$x = r \sin\theta \cos\phi, \quad 0 \le \theta \le \pi, \ 0 \le \phi \le 2\pi$

$y = r \sin\theta \sin\phi, \quad 0 \le r \le \infty$

$z = r \cos\theta$

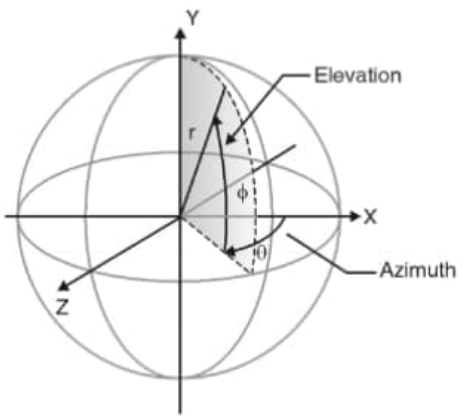


Fig. 1.10.4 : Spherical coordinate system.

1.10.1.4 Cylindrical Coordinate System

Q. Write a short note on Cylindrical coordinate system.

- Cylindrical coordinate system is extension of circular polar system with added height of the point from XY plane. Fig. 1.10.5 shows the representation of cylindrical coordinate system.
- Azimuth angle with X axis is θ and height of point from the surface is z. Point P is represented using these parameters as P(r, θ, z), where r is the distance between origin and projection of P in the XY plane.

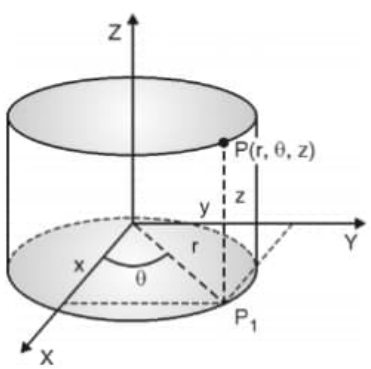


Fig. 1.10.5 : Cylindrical coordinate systems

1.10.2 Graphics Functions

Q. Explain various graphics functions

General-purpose graphics package provides users with a variety of functions for creating and manipulating pictures. Some of the fundamental functionalities of graphics packages are discussed here.

1. **Output primitives :** Output primitives are the basic building block, which includes shapes like a point, line, triangle, square, circle etc. It also includes area filling primitives to fill the area inside the polygon.
2. **Attributes :** They are used as a supplement to basic shapes. Functions related to attributes may include intensity and color specifications, line styles, text styles, and area-filling patterns. We may set the line width, pattern of line, size of brush etc.

3. **Geometric transformation :** Functions related to geometric transformation allow us to change the size, position, or orientation of an object.
4. **Viewing :** Using viewing options, we can set the viewing direction, type of projection to be used, the area of the scene to be selected for viewing etc.
5. **Structures :** Pictures can be subdivided into a component called structures or segments. Each structure defines one logical unit of the picture. Using such several segments with proper geometric transformations, objects can be designed easily.
6. **Input devices :** Modern graphics packages provide Interactive graphics, which needs support for various kinds of input devices, such as a mouse or a joystick. Input functions are used to control and process the data flow from the interactive devices.
7. **Control operations :** Graphics package contains a number of housekeeping tasks, such as clearing a display screen and initializing parameters.

1.10.3 Software Standards

Q. Write a short note on software standards.

- A standard graphics package such as GKS (Graphical Kernel System) and PHIGS (Programmers Hierarchical Interactive Graphics System) implements the specifications designated as standards by an international body ISO and ANSI. The main aim of such standards is to provide portability to application programs.
- Non-official standards are also developed and licensed by companies. Adobe's postscript and MIT's X-window system are such two industry standards. When packages are designed with standard graphics functions, the software can be moved easily from one hardware system to another and can be used in different implementations and applications. Without standards, programs require extensive rewriting to run on a different machine.
- GKS was originally designed as a two-dimensional graphics package; a three-dimensional GKS extension was subsequently developed. PHIGS is an extension of GKS with increased capabilities for object modelling, color specifications, surface rendering, and picture manipulations. An extension of PHIGS, called PHIGS provides an additional capability of three-dimensional surface-shading.

1.11 Latest trends in Computer Graphics

Q. Discuss the latest trends in computer graphics.

Nowadays, the role of computer graphics is not limited to traditional applications only. The applications of computer graphics are rather very interesting. Virtual Reality (VR) and Augmented Reality (AR) are promising application areas of computer graphics.

1.11.1 Virtual Reality

Q. Write a short note on virtual reality.

1.11.1.1 Introduction to Virtual Reality

Q. What is virtual reality?

Q. Define: Virtual reality.

- **Virtual reality** is the concept which tries to mimic the real world. Virtual reality is the environment which simulates user's expressions, emotions and activities in a way such that the synthetic world generated in the computer is

experienced as real. Virtual reality is a simulator which uses computer graphics to create a realistic looking virtual world, which is responsive. User can interact within objects of the virtual world and response of the user input is dynamically decided.

- A virtual world created in the simulator is not static. It changes with user input. Example of a video game is sufficient to explain everything. Virtual reality creates the feeling of immersion - being a part of scene actions. User does not just see or manipulate an object, he can also touch and feel it.
- To add more realism, visual images are provided with acoustic images, which can simulate the sound within a virtual environment.
- Virtual reality can be thought of as real-time simulation and interactions through multiple human sensorial channels. Sensorial channels are used to model visual, auditory, tactile, taste, smell etc.
- Virtual reality is all - immersive, interactive and imaginative.

1.11.1.2 Components of Virtual Reality

Q. Explain various components of virtual reality.

Fig. 1.11.1 describes the classical components of a virtual reality system.

- Tasks are the first component of the VR system. The usercommunicates with the system by a set of well-defined tasks. Tasks must be well defined. All components in the VR system have a very good interface between them. I/O devices are used either for user input (like data gloves, trackers etc.) or output (like a large display, robotic arms etc.). Using I/O activities, the user interacts with the VR system.
- Special purpose VR architecture is designed to handle the computational complexity of the VR system. Hardware architectures are designed to satisfy the demands of real-time input reading, scene computation and output rendering.

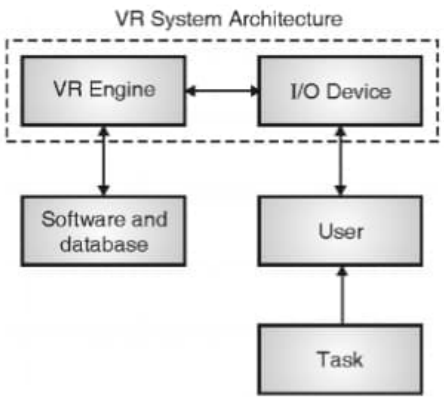


Fig. 1.11.1 : Components of the VR System

- Software and database are designed to process real-time requests coming from the VR engine. Rendering of the scene is done on demand and displayed on the output device. World Tool Kit, Java 3D etc. software are designed to ease the VR programming. Using VR programming language, many applications have been developed for various fields like medical, military, navigation, education, training etc.

1.11.1.3 Types of VR Systems

Q. List and explain the types of VR systems.

1. Desktop VR :

- This is also known as Window to World (WoW) system. Such VR systems are used for classroom teaching or training purpose. They are less interactive. The virtual environment is simulated on monitor screens in front of the user.
- This is non immersive and less expensive system. User does not need specialized input tools for interaction. Desktop VR relies on use of the mouse, joystick, trackball, space ball etc for the navigation and object control in the virtual world.

- Modern games are a good example of such a VR system. However, in-game, due to lack of sense of surrounding vision, user don't know what is going on around him. HTML and Virtual Reality Modelling Language (VRML) are the standard tools who have made the desktop VR so popular.

2. Video Mapping VR :

- Video mapping VR system uses projection technique. It uses multiple cameras, which captures the movement of the user and simulate it in the virtual world on the scene.
- Spatial coordinates of user data are mapped inside the virtual world. 2D image of a user is created in the virtual environment. Actions of a user in the real world are mapped in the virtual world through camera and sensors. User can see and feel his actions on the monitor screen.

3. Immersive VR :

- Immersive VR systems provide complete involvement of the user in the virtual environment. The user feels as if he is a part of the virtual world. Head Mounted Display (HMD) device makes it possible. User's position and viewpoint are mapped within a virtual environment. HMD device projects the virtual scene directly in the user's eye. Auditory information is passed through earphones. With the help of data gloves, action of the user is mapped in the virtual scene.
- The environment created with this technique is so real and interactive that user cannot distinguish what is real and what is virtual. Another variation of such a system is 'Cave'. In which, multiple projectors are used to project scene on walls of the room. The user feels as if he is standing in a virtual world.

4. Telepresence :

- Telepresence links human operator senses with real-world sensors. The human operator is connected with robots via sensors and cameras.
- The human operator can experience and see the environment where the robot is located. In dangerous conditions like fire, firefighters and fire vehicle can be remotely operated.
- Doctors use a micro camera mounted on the head of the cable and can see and perform the operation without cutting a big hole on the patient's body. This system is very useful in the application which needs to be performed in hostile conditions.

5. Mixed Reality :

- It is also known as augmented reality. This is semi immersive technique. Involvement of the user is somewhere between immersive and non immersive system, and hence the name is a mixed reality.
- In this system, telepresence input is merged with computer-generated inputs. Head Up Displays (HUD) used in military aircraft are the example of such a system. This HUD displays flight data like altitude, speed, height etc.

6. Fish Tank VR :

- Fish tank VR combines desktop VR system and stereoscopic monitor display with the mechanical head tracker. This hybrid system out performs a simple desktop VR system.
- Liquid crystal shutter glasses are used for stereoscopic images. The scene on screen is updated with user head movement.

1.11.1.4 Input / Output Devices

Q. List various input / output devices of VR systems.

Sound VR system needs to be equipped with various input-output interfaces. Few popularly used I/O devices are listed here :

Input devices :

- Trackers interface
  - 3D trackers
  - Mechanical trackers
  - Electromagnetic trackers
  - Ultrasonic trackers
  - Optical trackers
  - Inertial trackers
  - Trackballs
- Gesture interface
  - Pinch gloves
  - 5DT data gloves

Output devices :

- Graphics display interface
  - Personal Graphics Displays
  - Head Mounted Displays (HMDs)
  - Monoscopic HMD
  - Binoscopic HMD
  - Hand Supported Displays (HSDs)
  - Floor Supported Displays
  - Desk Supported Displays
- Large Volume Displays
  - Monitor Based Large Volume Displays
  - Projector-Based Displays
- Sound Display and Interface
- Haptic Feedback
  - Tactile and Force Feedback Interfaces

1.11.1.5 Applications of VR

Q. State few applications of VR systems.

- Arts and Entertainment
- Education
- Medical and Surgery
- Military
- Manufacturing and Architecture
- Scientific Visualization

1.11.2 Augmented Reality

Q. Write a brief note on augmented reality.

1.11.2.1 Introduction to Augmented Reality

- Virtual reality is becoming increasingly popular. However synthetic images (images generated on the computer) used in games and movies are detached from the physical surrounding. This is a limitation of VR based systems. User cannot use the real-time information from the environment around him in a computer system.
- Augmented Reality (AR) has been quite a hot topic since the last decade, but it gets the prime attention after the release of a product like Google glass. AR works on computer vision based algorithms to augmented various information like audio, video, graphics and other sensory data on real-world objects using the device camera. AR helps to integrate all such information with real-world objects and in an interactive way so that the virtual elements become the part of the real world and creates the mix reality.
- Augmented reality is the mix reality. It implies reality with a bit of virtual reality. Augmented reality possesses the following three primary characteristics :
    - It must be a mixture of real and virtual information, where primary actions take place in the real world and simulated in the virtual world.
    - It must have real-time updates.
    - Virtual information must be registered in 3D space of the real world
- While the user moves in the real world, the effect is directly rendered in the virtual environment within a computer. Game playing (Pokemon Go) is perhaps the best example to explain the concept.
- ARToolKit, ARKit, ARCore etc. are the popular tools and framework to implemented AR based applications.

1.11.2.2 Types of Augmented Reality

Q. Discuss various types of AR systems.

According to the objective and the application area, we can classify the AR system as follow :

1. **Marker Based AR Systems** : Such systems use a camera and some type of visual markers like Logo, Objects, QR code, Bar code or some other visual information to produce and augment the information from such markers. As shown in the following diagram, the camera scans the square on white paper to produce the virtual house in the camera screen.

2. **Markerless AR Systems :** Such systems are also known as location-based, position based or GPS based AR systems. In such systems, AR device is equipped with GPS, velocity meter or accelerometer, digital compass etc. to provide the location-based data. The main force behind the markerless systems is the availability of smartphones and the various

sensors they provide. Typically, markerless systems are used for finding nearby businesses and directions as shown in the following diagram.

3. **Projection based AR Systems :** As a name suggests, such systems work by projecting artificial light on real-world surface. Such systems work by sensing the interaction between human and the projected light on the world surface. Actions are performed by the previously known light projection and the altered projection. Projection based system utilizes laser plasma to project a 3D interactive hologram.

4. **Superimposition based AR Systems :** Such systems superimpose the new view on the old view of the object. It is very essential to recognize the underlying object to replace it with the newer object. It is not possible to augment the new object without detecting and recognizing the object in the real world. Such systems are highly used in determining the interior of the home. The application scans the objects in the room and creates the view with different augmented objects from the digital furniture catalogue.

Review Questions

Topic : Basics of Computer Graphics

Q. 1 What is computer graphics?

Q. 2 Define terms : Image, Object, Pixel, Resolution.

Q. 3 List and explain applications of computer graphics.

Topic : Display Devices

Q. 1 Explain CRT with neat diagram.

Q. 2 Describe working principal of flat panel display.

Q. 3 Describe working of LCD.

Topic : Software and Packages

Q. 1 List three basic primitive functions and explain any one of it.

Q. 2 Explain text mode graphics function used to display the string "Hello" at some specified location in specific color with its syntax.

Q. 3 How to initialize graphics mode of your display system?

Q. 4 Write a C code to display a single pixel on screen.

Q. 5 Write a C program to draw rectangle and circle on screen.

Q. 6 Write a C program to draw a triangle using line commands on screen.

Q. 7 Describe the command used to color a given shape with its syntax.

Q. 8 Write a C code to draw a polygon and then fill it with yellow color.

Q. 9 What is need of closegraph() function in graphics program?

Q. 10 Explain any three graphics standards.

Topic : Advanced Topic

Q. 1 Discuss virtual reality system.

Q. 2 Describe applications of virtual reality and augmented reality.

Chapter Ends...

❑❑❑

CHAPTER 2

Unit II

Raster Scan Graphics

Syllabus

Basic Concepts in Line Drawing : Line drawing algorithms, Digital Differential Analyzer (DDA) algorithm, Bresenham's algorithm, Circle Generating Algorithms: Symmetry of circle, Bresenham's Circle Drawing Algorithm, Polygons, Types of Polygons, Inside Outside Test, Polygon Filling : Seed fill algorithms : Flood Fill, Boundary Fill, Scan Line Algorithm, Scan Conversion, Frame Buffers, Character Generation Methods : Stroke, Starbust, Bitmap

2.1 Basic Concepts in Line Drawing

2.1.1 Point

Q. What is point?

- **Point** or **pixel** is the smallest unit that we can address on the screen. Point is the basic unit of the scene.
- The point in the plane is defined using coordinate pair (x, y), whereas point in space is defined by a triplet (x, y, z). Here, *x* and *y* are the plane coordinates and *z* is the distance of a point from XY plane.
- Relationship between points defines the shape. If all points are arranged along a straight path, it defines a line. If points are at equidistance from the center, it defines a circle. If the distance of all points from two foci point is constant, it defines ellipse and so on.
- Process of turning '*on*' or '*off*' particular pixel of a given shape is called **rasterizing / rasterization** of point.

2.1.2 Basics of Line

Q. Explain the way of computing pixel positions using slope-intercept formula.

Q. Explain explicit formula of line drawing.

- The line is a collection of points along a straight path. Typically, the line is described by its two endpoints (x_1, y_1) and (x_2, y_2). Intermediate points on line are calculated by line generation algorithm.
- Given the endpoints $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, any point P(x, y) on the line can be computed using slope-intercept formula as follow :

  It is evident from the Fig. 2.1.1 that the slope of the line is,

$$m = \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1}$$

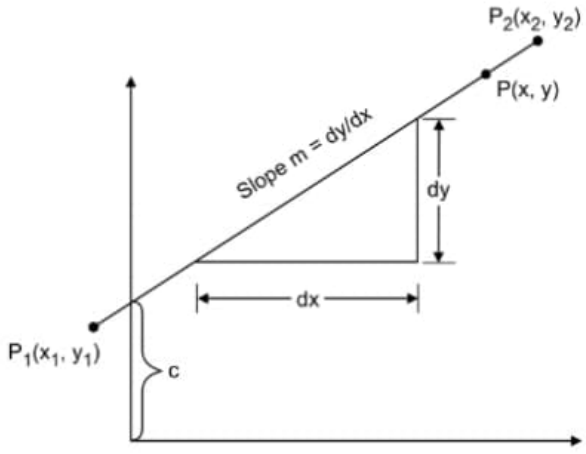


Fig. 2.1.1 : Representation of line

Let c represents the Y-intercept of the line. Given the x coordinate, value of corresponding y coordinate of any point is computed as follow :

$$y = mx + c$$

Lines with different slopes are shown in the Fig. 2.1.2.

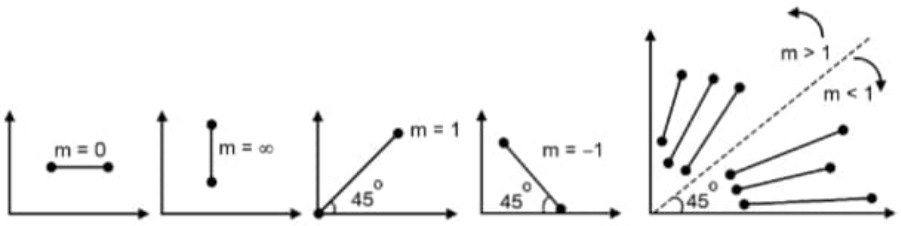


Fig. 2.1.2 : Lines with different slope

2.1.2.1 Characteristics of Ideal Line

Q.	Discuss the characteristics of ideal line.
Q.	Enlist the properties of ideal line.

- When we draw a line on paper, it looks smooth and continuous. But due to the grid structure of monitor, when the line is rasterized, it looks zigzag except horizontal, vertical and diagonal line.
- Distance between pixels also changes with orientation, and hence the brightness.

An ideal line should have the following properties :

- It should interpolate both the endpoints.
- The brightness of the line should be independent of the orientation of the line.
- It should appear straight and smooth.
- It should be drawn quickly.
- All properties are satisfied when we draw lines on paper. But when we rasterize it on the monitor, most of the properties do not hold.
- On the monitor screen, vertical and horizontal lines are displayed with maximum intensity; the brightness of inclined line changes with orientation.

Fig. 2.1.3 : Line with different intensities and orientations

2.1.2.2 Line Representation

Q. Mention the different ways to represent the line.

The line can be represented using various ways. The most common representations are as follow :

1. Implicit representation : $ax + by + c = 0$

   (x, y) is the point on line and a, b, c are the line coefficient

2. Explicit representation : $y = mx + c$

   (x, y) is the point on line, m is the slope of the line and c is the Y-intercept

3. Parametric representation :

$$x = x_1 + (x_2 - x_1) \cdot t$$

$$y = y_1 + (y_2 - y_1) \cdot t$$

   Where, $0 \leq t \leq 1$. (x_1, y_1) and (x_2, y_2) are endpoints of line and (x, y) is any point on line for given some value of t.

   For $t = 0$, $(x, y) = (x_1, y_1)$ and for $t = 1$ $(x, y) = (x_2, y_2)$. For any other value of t, we will get some intermediate point on the line.

2.2 Line Drawing Algorithms

W-18

Q. State two line drawing algorithms. **(W-18, 2 Marks)**

Line generation or **scan conversion of line** is the process of turning *on* or *off* the pixel along the line path.

In this section, we will discuss following incremental line drawing algorithms.

- Digital Differential Analyzer
- Bresenham's Algorithm

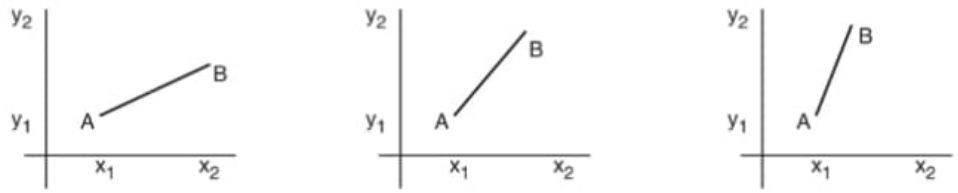
In incremental approach, line starts from the first end point and some fix increment value is added to current pixel to get the coordinates of next pixel. This process is repeated until line interpolates second endpoint.

2.2.1 Digital Differential Analyzer (DDA) Algorithm

W-18

Q. Explain and write steps for DDA line drawing algorithm. **(W-18, 4 Marks)**

- Digital Differential Analyzer (DDA) is a simple, incremental line scan converting algorithm.
- In the DDA algorithm, horizontal or vertical displacement is set to unit interval and the corresponding displacement for other direction is calculated using slope.
- If the line makes an angle less than 45° with X-axis (i.e. $m < 1$), increment in the X direction is set to 1 and corresponding Y is computed.
- And if line makes an angle greater than 45° with X-axis (i.e. $m > 1$), increment in the Y direction is set to 1 and corresponding X is computed.
- For line with slope $m = 1$, increment in both direction is set to 1.
- Fig. 2.2.1 shows the lines with different orientations.



(a) Line with slope |m| < 1 **(b) Line with slope |m| = 1** **(c) Line with slope |m| > 1**

Fig. 2.2.1 : Lines with different orientations

- Almost all recent monitors support integer coordinate system. Only integer movement is possible in any direction. It is not possible to draw a pixel at position (2.3, 5.6), the system either rounds up or round down coordinate values and then display the point.
- We will derive the equation to draw a line in the *first quadrant*.
- Another assumption is that we will process the line from *left to right*. It is intuitive to derive a line in any other quadrant just by changing the sign of increment values.
- Let's consider that Δx and Δy are the increments in x and y direction, respectively. Value of x and y coordinates at k + 1 location is given by,

$$\text{New value} = \text{Old value} + \text{Increment}$$

$$x_{k+1} = x_k + \Delta x$$

$$y_{k+1} = y_k + \Delta y$$

We will discuss three cases.

Case - I : Slope is less than 1 (|m| < 1)

When slope is less than 1, Δx is set to unit interval, i.e., $\Delta x = 1$ and corresponding y coordinate is computed.

From equation of slope,

$$m = \frac{\Delta y}{\Delta x} = \frac{\Delta y}{1}$$

$$\therefore \Delta y = m$$

So coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k + 1$$

$$y_{k+1} = y_k + \Delta y = y_k + m$$

Value of m is real number, so y_{k+1} may be real number. Hence, every time after adding m to y_k, answer is rounded to nearest integer. If line is processed from right to left, value of x and y decreases in every iteration. So, Δx is set to –1 and Δy is set to –m.

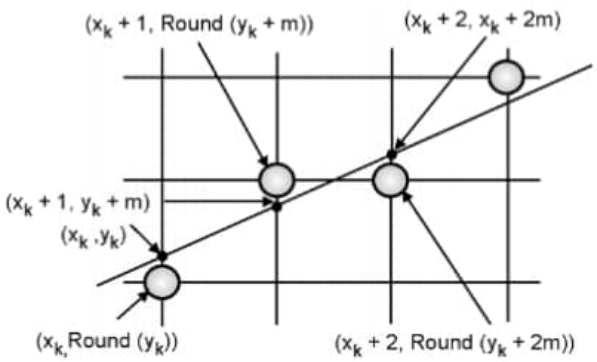


Fig. 2.2.2 : Line with |m| < 1

Coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k - 1$$

$$y_{k+1} = y_k + \Delta y = y_k - m$$

Case - II : Slope is greater than 1 (|m| > 1)

When slope is greater than 1, Δy is set to unit interval, i.e., $\Delta y = 1$ and corresponding x coordinate is computed.

From equation of slope,

$$m = \frac{\Delta y}{\Delta x} = \frac{1}{\Delta x}$$

$$\therefore \Delta x = \frac{1}{m}$$

Coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k + \frac{1}{m}$$

$$y_{k+1} = y_k + \Delta y = y_k + 1$$

If line is processed from right to left, value of x and y decreases in every iteration. Hence, Δx is set to $-\frac{1}{m}$ and Δy is set to – 1.

Coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k - \frac{1}{m}$$

$$y_{k+1} = y_k + \Delta y = y_k - 1$$

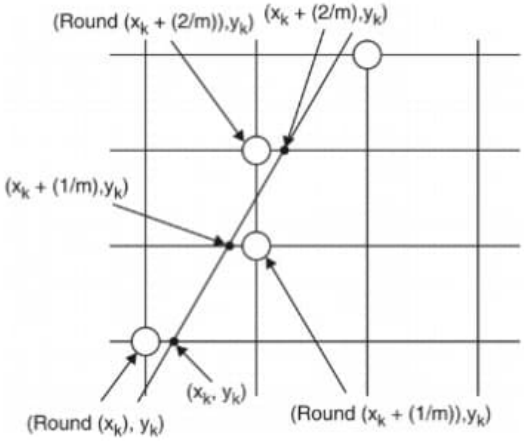


Fig. 2.2.3 : Line with |m| > 1

Case - III : Slope is 1 (|m| = 1)

When slope of line is 1, i.e. line makes an angle of 45° with X axis, we increment both coordinate value by 1. So, $\Delta x = \Delta y = 1$.

Coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k + 1$$

$$y_{k+1} = y_k + \Delta y = y_k + 1$$

If line is processed from right to left, in that case value of x and y decreases in every iteration. So, Δx and Δy are set to – 1.

Coordinate value for next pixel is given as,

$$x_{k+1} = x_k + \Delta x = x_k - 1$$

$$y_{k+1} = y_k + \Delta y = y_k - 1$$

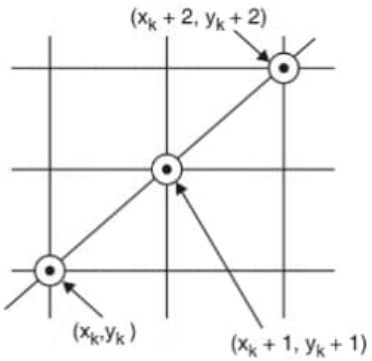


Fig. 2.2.4 : Line with |m| = 1

Table 2.2.1 shows the sign and value of x and y increments to draw a line in any quadrant.

Table 2.2.1 : Generalization of DDA line drawing algorithm

Quadrant	Magnitude of Slope	If processed from left to right		If processed from right to left	
		Δx	Δy	Δx	Δy
1	$\|m\| < 1$	1	m	− 1	− m
	$\|m\| > 1$	1/m	1	− 1/m	− 1
2	$\|m\| < 1$	1	− m	− 1	m
	$\|m\| > 1$	− 1/m	− 1	1/m	1
3	$\|m\| < 1$	1	m	− 1	− m
	$\|m\| > 1$	− 1/m	1	1/m	− 1
4	$\|m\| < 1$	1	− m	− 1	m
	$\|m\| > 1$	1/m	− 1	− 1/m	1

Advantage :

Q. What are the advantages of DDA line drawing algorithm?

1. It is simple.
2. It is easy to understand.
3. It eliminates multiplications involved in explicit line drawing equation, $y = mx + c$.
4. DDA is quite faster.
5. DDA is more efficient than an implicit line drawing algorithm.
6. It does not require any special skill to implement it.

Disadvantage :

Q. What are the disadvantages of DDA line drawing algorithm?

1. It involves floating point operation for each pixel.
2. It performs rounding off operation for each pixel.
3. Rounding off error is accumulated in each iteration and calculated pixel position may drift away from the actual position due to cumulative rounding off error.
4. It is slower.

Steps of DDA line drawing algorithm :

Q. Write the steps of DDA line drawing algorithm.

Note : These steps are to draw a line in first quadrant – growing from left to right or right to left only. Generalized line algorithm can be derived with the help of increment and sign as shown in Table 2.2.1.

Step 1 : Read two endpoints (x_1, y_1) and (x_2, y_2)

Step 2 : Compute horizontal and vertical differences as,

$$dx = x_2 - x_1$$

$$dy = y_2 - y_1$$

Step 3 : Set number of steps to N = max(|dx|, |dy|)

Step 4 : Repeat step 5 to step 8 N times.

Step 5 : If |dx| > |dy|and $x_1 < x_2$, set increments in X and Y direction to 1 and m, respectively.

Step6 : If |dx| > |dy|and $x_1 \geq x_2$, set increments in X and Y direction to – 1 and – m, respectively.

Step 7 : If |dx| < |dy|and $y_1 < y_2$, set increments in X and Y direction to 1/m and 1, respectively.

Step 8 : If |dx| < |dy|and $y_1 \geq y_2$, set increments in X and Y direction to – 1/m and – 1, respectively.

DDA line drawing Algorithm :

Q. Write an algorithm for DDA line drawing.

Algorithm to draw a line in first quadrant, growing from left to right is stated as follows :

```
Algorithm DDA_LINE (x1, y1, x2, y2)
        dx ← x2 – x1
        dy← y2 – y1
        x←  x1,
        y ←y1
        m ←  dy/dx
    if  abs(m) < 1 then
        num_of_pixels← abs (dx)
    else
        num_of_pixels←abs (dy)
    end

        Δx  ←  dx/num_of_pixels
        Δy  ←  dy/num_of_pixels

        putPixel (x, y)
for     x   ←  x1 to x2 do
        x   ←  x + Δx
        y   ←  y + Δy
        putPixel (x, Round (y))
end
```

This algorithm draws a line from left to right in first quadrant.

Example 2.2.1 : Calculate the pixel coordinates of line AB using DDA algorithm. Where A = (0, 0) and B = (4, 6).

Solution :

Let's take, $(x_1, y_1) = (0, 0)$ and $(x_2, y_2) = (4, 6)$

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{6-0}{4-0} = \frac{6}{4} = 1.5$$

Line is in first quadrant, it is growing from left to right and slope $|m| > 1$, so set $\Delta x = 1/m = 0.67$ and $\Delta y = 1$. So,

$$x_{k+1} = x_k + \Delta x = x_k + 0.67$$

$$y_{k+1} = y_k + \Delta y = y_k + 1$$

Table P. 2.2.1 shows the calculation for required pixels.

Table P. 2.2.1 : Calculation for required pixels

k	(x_k, y_k)	$x_{k+1} = x_k + 0.67$	$y_{k+1} = y_k + 1$	Round (x_{k+1})
0	(0, 0)	0 + 0.67 = 0.67	0 + 1 = 1	1
1	(1, 1)	0.67 + 0.67 = 1.34	1 + 1 = 2	1
2	(1, 2)	1.34 + 0.67 = 2.01	2 + 1 = 3	2
3	(2, 3)	2.01 + 0.67 = 2.68	3 + 1 = 4	3
4	(3, 4)	2.68 + 0.67 = 3.35	4 + 1 = 5	3
5	(3, 5)	3.35 + 0.67 = 4.02	5 + 1 = 6	4
6	(4, 6)	-	-	-

In Table P. 2.2.1, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.1.

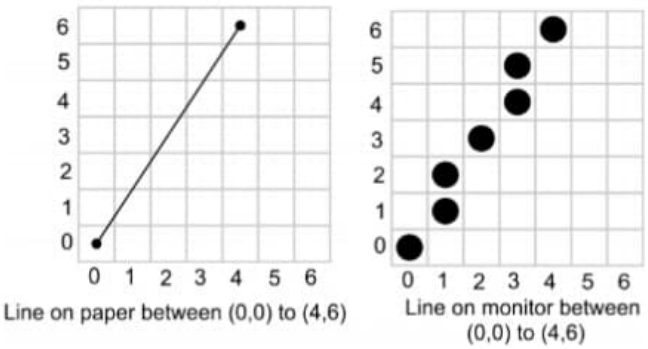


Fig. P. 2.2.1 : DDA line between points (0, 0) and (4, 6)

Example 2.2.2 : Draw a line from point (2, 2) to (10, 7) using DDA line drawing algorithm.

Solution :

Let's take $(x_1, y_1) = (2, 2)$ and $(x_2, y_2) = (10, 7)$

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-2}{10-2} = \frac{5}{8} = 0.625$$

Line is in first quadrant, it is growing from left to right and slope $|m| < 1$, so set $\Delta x = 1$ and $\Delta y = m = 0.625$. So,

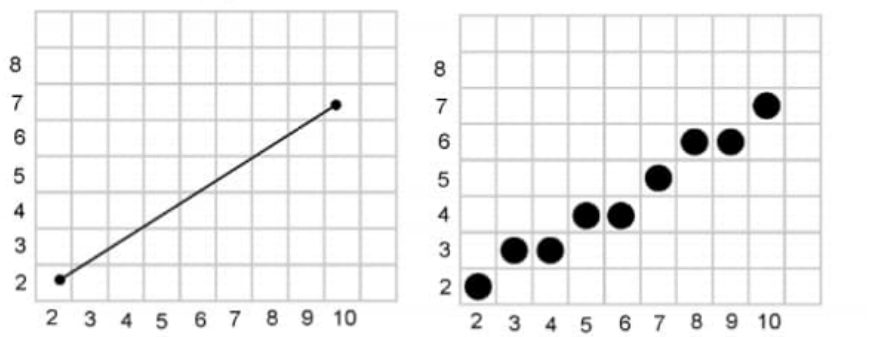
$$x_{k+1} = x_k + \Delta x = x_k + 1$$
$$y_{k+1} = y_k + \Delta y = y_k + 0.625$$

Table P. 2.2.2 shows the calculation for required pixels.

Table P. 2.2.2 : Calculation for required pixels

k	(x_k, y_k)	$x_{k+1} = x_k + 1$	$y_{k+1} = y_k + 0.625$	Round (y_{k+1})
0	(2, 2)	2 + 1 = 3	2 + 0.625 = 2.625	3
1	(3, 3)	3 + 1 = 4	2.625 + 0.625 = 3.25	3
2	(4, 3)	4 + 1 = 5	3.25 + 0.625 = 3.875	4
3	(5, 4)	5 + 1 = 6	3.875 + 0.625 = 4.5	5
4	(6, 5)	6 + 1 = 7	4.5 + 0.625 = 5.125	5
5	(7, 5)	7 + 1 = 8	5.125 + 0.625 = 5.75	6
6	(8, 6)	8 + 1 = 9	5.75 + 0.625 = 6.375	6
7	(9, 6)	9 + 1 = 10	6.375 + 0.625 = 7.000	7
8	(10, 7)	-	-	-

In Table P. 2.2.2, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.2.



The line on paper between (2,2) to (10,7) The line on the monitor between (2,2) to (10,7)

Fig. P. 2.2.2 : DDA line between points (2, 2) and (10, 7)

Example 2.2.3 : Draw a line from A(10, 2) to B(6, 8) using DDA line drawing algorithm.

Solution :

Let's take $(x_1, y_1) = (10, 2)$ and $(x_2, y_2) = (6, 8)$

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1}$$
$$= \frac{8-2}{6-10} = \frac{6}{-4} = -1.5$$

The line is in the first quadrant, it is growing from **right to left** and slope m < 1. It is intuitive from the line coordinates that when line proceeds from point A to point B, its x coordinate decreases and y coordinate increases, so set

$$\Delta y = 1 \text{ and } \Delta x = \frac{1}{m} = -0.67$$

$$x_{k+1} = x_k + \Delta x = x_k - 0.67$$

$$y_{k+1} = y_k + \Delta y = y_k + 1$$

Table P. 2.2.3 shows the calculation for required pixels.

Table P. 2.2.3 : Calculation for required pixels

k	(x_k, y_k)	$x_{k+1} = x_k - 0.67$	$y_{k+1} = y_k + 1$	Round (x_{k+1})
0	(10, 2)	10 – 0.67 = 9.33	2 + 1 = 3	9
1	(9, 3)	9.33 – 0.67 = 8.67	3 + 1 = 4	9
2	(9, 4)	8.67 – 0.67 = 8.00	4 + 1 = 5	8
3	(8, 5)	8.00 – 0.67 = 7.33	5 + 1 = 6	7
4	(7, 6)	7.33 – 0.67 = 6.67	6 + 1 = 7	7
5	(7, 7)	6.67 – 0.67 = 6.00	7 + 1 = 8	6
6	(6, 8)	-	-	-

In Table P. 2.2.3 , column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.3.

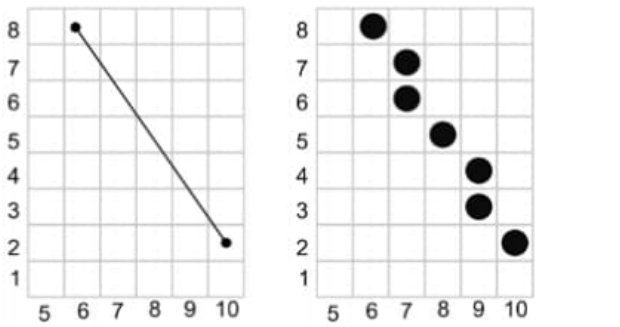


Fig. P. 2.2.3 : DDA line between points (10, 2) and (6, 8)

Example 2.2.4 : Draw a line from A(20, 10) to B(30, 18) using DDA line drawing algorithm.

Solution :

$$\text{Let's take } (x_1, y_1) = (20, 10) \text{ and } (x_2, y_2) = (30, 18)$$

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{18 - 10}{30 - 20} = \frac{8}{10} = 0.8$$

Line is in first quadrant, it is growing from left to right and slope $|m| < 1$, so set $\Delta x = 1$ and $\Delta y = m = 0.8$.

$$x_{k+1} = x_k + \Delta x = x_k + 1$$

$$y_{k+1} = y_k + \Delta y = y_k + 0.8$$

Table P. 2.2.4 : Calculation for required pixels

K	(x_k, y_k)	$x_{k+1} = x_k + 1$	$y_{k+1} = y_k + 0.8$	Round (y_{k+1})
0	(20, 10)	20 + 1 = 21	10.0 + 0.8 = 10.8	11
1	(21, 11)	21 + 1 = 22	10.8 + 0.8 = 11.6	12
2	(22, 12)	22 + 1 = 23	11.6 + 0.8 = 12.4	12
3	(23, 12)	23 + 1 = 24	12.4 + 0.8 = 13.2	13
4	(24, 13)	24 + 1 = 25	13.2 + 0.8 = 14.0	14
5	(25, 14)	25 + 1 = 26	14.0 + 0.8 = 14.8	15
6	(26, 15)	26 + 1 = 27	14.8 + 0.8 = 15.6	16
7	(27, 16)	27 + 1 = 28	15.6 + 0.8 = 16.4	16
8	(28, 16)	28 + 1 = 29	16.4 + 0.8 = 17.2	17
9	(29, 17)	29 + 1 = 30	17.2 + 0.8 = 18.0	18
10	(30, 18)	-	-	-

In Table P. 2.2.4, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.4.

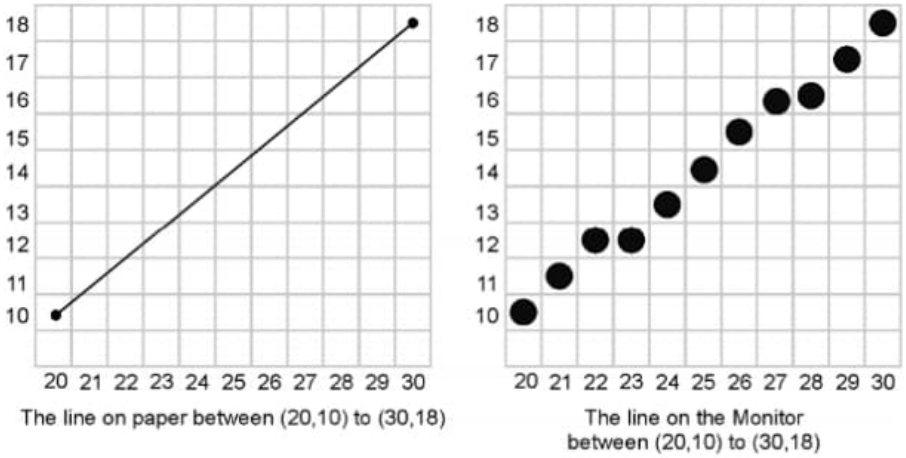


Fig. P. 2.2.4 : DDA line between points (20, 10) and (30, 18)

Example 2.2.5 : Draw a line from A(0, 0) to B(-5, -5) using DDA line drawing algorithm.

Solution :

Let's take (x_1, y_1) = (0, 0) and (x_2, y_2) = (-5, -5)

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-5-0}{-5-0} = \frac{-5}{-5} = 1$$

Line is in **third quadrant**, it is growing from **right to left** and slope |m| = 1, so set $\Delta x = -1$ and $\Delta y = -1$

$$x_{k+1} = x_k + \Delta x = x_k - 1$$

$$y_{k+1} = y_k + \Delta y = y_k - 1$$

Table P. 2.2.5 : Calculation for required pixels

K	(x_k, y_k)	$x_{k+1} = x_k - 1$	$y_{k+1} = y_k - 1$	Round (x_{k+1}, y_{k+1})
0	(0, 0)	0 – 1 = – 1	0 – 1 = – 1	(– 1, – 1)
1	(– 1, – 1)	– 1 – 1 = – 2	– 1 – 1 = – 2	(– 2, – 2)
2	(– 2, – 2)	– 2 – 1 = – 3	– 2 – 1 = – 3	(– 3, – 3)
3	(– 3, – 3)	– 3 – 1 = – 4	– 3 – 1 = – 4	(– 4, – 4)
4	(– 4, – 4)	– 4 – 1 = – 5	– 4 – 1 = – 5	(– 5, – 5)
5	(– 5, – 5)	-	-	-

In Table P. 2.2.5, column 2, i.e. (x_k, y_k) are the points to be displayed.

2.2.2 Bresenham's Line Drawing Algorithm

Q. Explain with the help of illustration, Bresenham's line drawing algorithm.

Q. Explain Bresenham's line drawing algorithm (Do not give Pseudo-code).

Q. Write Bresenham's line drawing algorithm.

- DDA algorithm is simple but not efficient.
- Bresenham has proposed an incremental approach to compute the next pixel position along a straight line. Given the endpoints of the line (x_0, y_0) and (x_1, y_1), we will derive the incremental formula.
- This approach selects the pixel which is close to the ideal line. It uses only integer addition and subtraction operations.
- **Assumption :** Line is in the first octant and it is growing from left to right and $|m| < 1$.

  The slope of the line is given as $m = \frac{y_1 - y_0}{x_1 - x_0}$.

- Fig. 2.2.5 shows the segment of the line. The line has a slope less than 1. By sampling in x-direction, we shall compute y coordinate that best matches with the ideal line. In Fig. 2.2.5, (15, 20) is the known pixel value, next pixel can be drawn on (16, 20) or (16, 21).
- Thus, for the line with slope $|m| < 1$, if current pixel is (x_k, y_k), then next pixel on line would be at $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$.
- For each $(x_k + 1)$ position, algorithm determines in which scan line pixel should be plotted - in y_k or y_{k+1}?

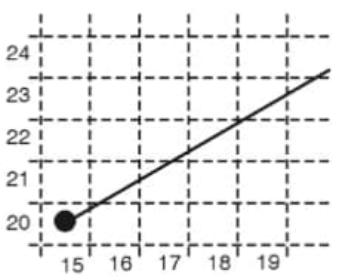


Fig. 2.2.5

- The computed y coordinate for (x_k+1) position is let's say y. Distance between y and y_k is d_1 and distance between (y_k+1) and y is d_2. (Refer Fig. 2.2.6)
- From Fig. 2.2.6, it is clear that $d_2 < d_1$ implies line is close to scanline (y_k+1), so we shall plot pixel (x_k+1, y_k+1). And if $d_{1c} < d_2$, line is close to scanline y_k and hence next pixel remains on a same scan line, and we shall plot pixel (x_k+1, y_k).

- Thus, the choice of next pixel depends on the sign of $d_1 - d_2$. If it is positive, plot (x_k+1, y_k) – pixel in East (E) direction, else plot (x_k+1, y_k+1) – Pixel in North-East (NE) direction. This computation needs to be performed recursively.
- Implicit representation of line is $y = mx + c$. At sampling position $x_k + 1$, corresponding y is computed as,

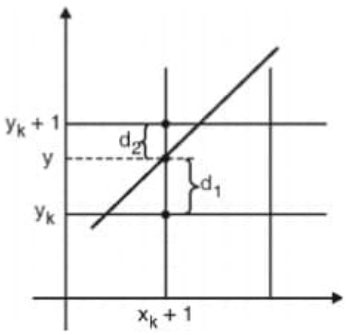


Fig. 2.2.6

$$y = m(x_k + 1) + c = m \cdot x_k + m + c$$

From Fig. 2.2.6,

$$d_1 = y - y_k = m \cdot x_k + m + c - y_k$$

$$d_2 = y_{k+1} - y = y_k + 1 - (m \cdot x_k + m + c)$$

($\because$ for next scan line, $y_{k+1} = y_k + 1$)

$$d_1 - d_2 = (m.x_k + m + c - y_k) - (y_k + 1 - m \cdot x_k - m - c)$$

$$= m.x_k + m + c - y_k - y_k - 1 + m \cdot x_k + m + c$$

$$= 2m \cdot x_k + 2m - 2y_k + 2c - 1$$

- To simplify the computation, multiply it by Δx,

Decision parameter $p_k = \Delta x (d_1 - d_2) = \Delta x (2m \cdot x_k + 2m - 2y_k + 2c - 1)$

$$= 2\Delta x \cdot m \cdot x_k + 2\Delta x \cdot m - 2\Delta x \cdot y_k + 2\Delta x \cdot c - \Delta x$$

Put $m = \Delta y / \Delta x$ in above equation and solve it, we get

$$= 2\Delta y \cdot x_k + 2\Delta y - 2\Delta x \cdot y_k + 2\Delta x \cdot c - \Delta x$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + B$$

Where, $B = 2\Delta y + \Delta x (2c - 1)$. Value of B does not depend on pixel position, so in subsequent computation of p_k, B would be eliminated.

- Decision parameter at step k + 1 would be,

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + B$$

- Subtracting p_{k+1} and p_k, we get incremental parameter value,

$$p_{k+1} - p_k = (2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + B) - (2\Delta y \cdot x_k - 2\Delta x \cdot y_k + B)$$

$$= 2\Delta y \cdot (x_k + 1) - 2\Delta x \cdot y_{k+1} - 2\Delta y \cdot x_k + 2\Delta x \cdot y_k$$

$$= 2\Delta y \cdot x_k + 2\Delta y - 2\Delta x \cdot y_{k+1} - 2\Delta y \cdot x_k + 2\Delta x \cdot y_k$$

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x (y_{k+1} - y_k)$$

- Depending on sign of p_k, $(y_{k+1} - y_k)$ will be either 0 or 1.

Initial decision parameter :

The line starts from the end point (x_0, y_0). Implicit equation of line is : $y = mx + c$. The initial point is always on line. By putting first end point (x_0, y_0) in the implicit representation of the line, we get

$$c = y_0 - mx_0 = y_0 - (\Delta y/\Delta x) \cdot x_0$$

Put this value in equation (1).

$$\text{Initial decision parameter, } p_0 = 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + 2\Delta y + 2\Delta x (y_0 - (\Delta y/\Delta x) \cdot x_0) - \Delta x$$

$$= 2\Delta y \cdot x_0 - 2\Delta x \cdot y_0 + 2\Delta y + 2\Delta x \cdot y_0 - 2\Delta y \cdot x_0 - \Delta x$$

$$p_0 = 2\Delta y - \Delta x$$

Using these decision parameter values, we can compute the pixels along the line as follow.

Steps of Bresenham's line drawing algorithm :

Q. Mention the steps for Bresenham's line drawing algorithm

Following steps should be performed to draw a line (with $|m| < 1$) using Bresenham's algorithm.

Step 1 : Read two endpoints (x_0, y_0) and (x_1, y_1)

Step 2 : Plot the first pixel (x_0, y_0)

Step 3 : Compute the constants:

$$\Delta x = x_1 - x_0,\ \Delta y = y_1 - y_0,\ \text{initial decision parameter } p = 2\Delta y - \Delta x$$

Step 4 : if $p \geq 0$ then

$$x = x + 1$$

$$y = y + 1$$

$$\text{Set } p = p + 2(\Delta y - \Delta x)$$

//Select pixel in North-East (NE) direction

else

$$x = x + 1$$

$$\text{Set } p = p + 2\Delta y$$

//Select pixel in East (E) direction

Step 5 : Repeat step 4 Δx times

Note : For line with $|m|<1$, interchange the role of x and y, i.e. increment y in each iteration and compute corresponding x.

Q. Enlist advantages and disadvantages of Bresenham's line drawing algorithm.

Q. State merits and demerits of Bresenham's line drawing algorithm.

Advantages :

1. Involves only integer calculation.
2. It is faster than DDA.
3. Multiplication by two can be implemented in hardware using shift register.

Disadvantages :

1. Bresenham's line drawing approach does not consider the anti-aliasing.
2. It may not produce a smooth line.

Algorithm for Bresenham's Line drawing method :

Q. Write an algorithm for Bresenham's Line drawing method.

Algorithm :

Algorithm for Bresenham's line drawing approach (for $|m| < 1$) is described below.

```
Algorithm  BRESENHAM_LINE(x0, y0, x1, y1)
           Δx ←x1 – x0
           Δy ←y1 – y0
           p0← 2Δy - Δx
    for    k← 0 to Δx do
    if     pk< 0 then
           putPixel (xk + 1, yk)        // Select pixel in East direction
           pk + 1← pk + 2Δy
    else
           putPixel (xk + 1, yk + 1)    // Select pixel in North-East direction
           pk + 1← pk + (2Δy - 2Δx)
    end
end
```

Example 2.2.6 : Calculate the pixel coordinates of line PQ using Bresneham's algorithm where, P = (20, 20) and Q = (10, 12).

Solution :

Let us assume (x_1, y_1) = (20, 20) and (x_2, y_2) = (10, 12)

$\Delta x = dx = |20 - 10| = 10$, $\Delta y = dy = |20 - 12| = 8$

Initial decision parameter, $p_0 = 2dy - dx = 16 - 10 = 6$

$\Delta E = 2dy = 16$ (Increment if E pixel is selected)

$\Delta NE = 2(dy - dx) = -4$ (Increment if NE pixel is selected)

Table P. 2.2.6

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	-	-	10	12	$p_k = p_0 = 2dy - dx = 6$
1	(10, 12)	$p_k \geq 0$, so select NE	10 + 1 = 11	12 + 1 = 13	$p_k = p_k + \Delta NE = 6 - 4 = 2$
2	(11, 13)	$p_k \geq 0$, so select NE	11 + 1 = 12	13 + 1 = 14	$p_k = p_k + \Delta NE = 2 - 4 = -2$
3	(12, 14)	$p_k < 0$, so select E	12 + 1 = 13	14 + 0 = 14	$p_k = p_k + \Delta E = -2 + 16 = 14$
4	(13, 14)	$p_k \geq 0$, so select NE	13 + 1 = 14	14 + 1 = 15	$p_k = p_k + \Delta NE = 14 - 4 = 10$
5	(14, 15)	$p_k \geq 0$, so select NE	14 + 1 = 15	15 + 1 = 16	$p_k = p_k + \Delta NE = 10 - 4 = 6$
6	(15, 16)	$p_k \geq 0$, so select NE	15 + 1 = 16	16 + 1 = 17	$p_k = p_k + \Delta NE = 6 - 4 = 2$
7	(16, 17)	$p_k \geq 0$, so select NE	16 + 1 = 17	17 + 1 = 18	$p_k = p_k + \Delta NE = 2 - 4 = -2$
8	(17, 18)	$p_k < 0$, so select E	17 + 1 = 18	18 + 0 = 18	$p_k = p_k + \Delta E = -2 + 16 = 14$
9	(18, 18)	$p_k \geq 0$, so select NE	18 + 1 = 19	19 + 0 = 19	$p_k = p_k + \Delta NE = 14 - 4 = 10$
10	(19, 19)	$p_k \geq 0$, so select NE	19 + 1 = 20	19 + 1 = 20	$p_k = p_k + \Delta NE = 10 - 4 = 6$
11	(20, 20)	-	-	-	-

In Table P. 2.2.6, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.6.

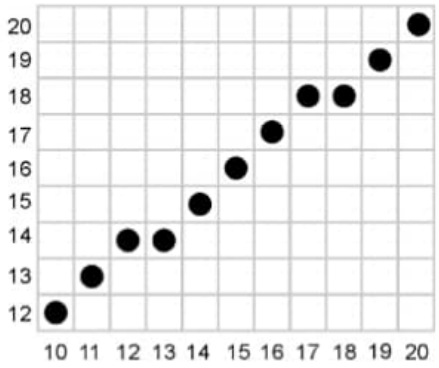


Fig. P. 2.2.6 : Line between (20, 20) to (10, 12) using Bresenham's line drawing approach

Example 2.2.7 : Consider line from (4,4) to (12,9). Use Bresenhaum's algorithm to rasterize this line. **(W-18, 4 Marks)**

Solution :

Let us assume (x_1, y_1) = $(4, 4)$ and $(x_2, y_2) = (12, 9)$

$dx = |x_2 - x_1| = |12 - 4| = 8$, $dy = |y_2 - y_1| = |9 - 4| = 5$

Initial decision parameter, $p_0 = 2dy - dx = 10 - 8 = 2$

$\Delta E = 2dy = 10$ (Increment if E pixel is selected)

$\Delta NE = 2\,(dy - dx) = 2(5 - 8) = -6$ (Increment if NE pixel is selected)

Table P. 2.2.7

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	-	-	4	4	$p_k = p_0 = 2dy - dx = 2$
1	(4, 4)	$p_k \geq 0$, so select NE	4 + 1 = 5	4 + 1 = 5	$p_k = p_k + \Delta NE = 2 - 6 = -4$
2	(5, 5)	$p_k < 0$, so select E	5 + 1 = 6	5 + 0 = 5	$p_k = p_k + \Delta E = -4 + 10 = 6$
3	(6, 5)	$p_k \geq 0$, so select NE	6 + 1 = 7	5 + 1 = 6	$p_k = p_k + \Delta NE = 6 - 6 = 0$
4	(7, 6)	$p_k \geq 0$, so select NE	7 + 1 = 8	6 + 1 = 7	$p_k = p_k + \Delta NE = 0 - 6 = -6$
5	(8, 7)	$p_k < 0$, so select E	8 + 1 = 9	7 + 0 = 7	$p_k = p_k + \Delta E = -6 + 10 = 4$
6	(9, 7)	$p_k \geq 0$, so select NE	9 + 1 = 10	7 + 1 = 8	$p_k = p_k + \Delta NE = 4 - 6 = -2$
7	(10, 8)	$p_k < 0$, so select E	10 + 1 = 11	8 + 0 = 8	$p_k = p_k + \Delta E = -2 + 10 = 8$
8	(11, 8)	$p_k \geq 0$, so select NE	11 + 1 = 12	8 + 1 = 9	$p_k = p_k + \Delta NE = 8 - 6 = 2$
9	(12, 9)	-	-	-	-

In Table P. 2.2.7, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 2.2.7.

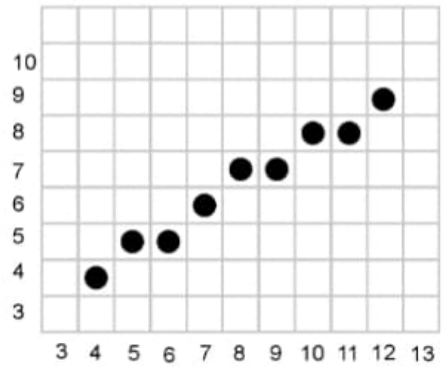


Fig. P. 2.2.7 : Line between (4, 4) to (12, 9) using Bresenham's line drawing approach

Example 2.2.8 : Write a function Bresenham's line(x_1, y_1, x_2, y_2, type) so it draws line with specified type. If type=0, solid line, type=1 dashed line and type=2 dotted line.

Solution :

The solid line plots pixels continuously. Dotted line plots alternate pixel. And we will consider the length of the dash in dashed line 5 pixels. So first we plot 5 pixels followed by a dot and this process is repeated till the end of the line.

```
Algorithm  BRESENHAM_LINE (x1, y1, x2, y2, type)
           Δx←x2 – x1
           Δy ←y2 – y1
           d ← 2Δy – Δx          // Initial decision parameter
           ΔE ← 2Δy
           ΔNE ← 2(Δy – Δx)
           x ←x1
           y ←y1
           Count ← 0
           flag ← 1

           putPixel (x, y)
    while  (x <x2) do
    if      d < 0 then
           d ←d + ΔE
    else
           d ← d + ΔNE
           y ← y + 1
    end
           x ← x + 1
    if     type = = 0 then       // solid Line
           putPixel (x, y)
    end
    if      type == 1 then                 // Dashed Line
```

```
            if Count < 5 then          // Draw five pixels and skip one to draw a dotted line
            putPixel (x, y)
            Count ← Count + 1
        end

        if  Count == 5 then
            Count ← 0
        end
    end

    if      type == 2 then             // Dotted Line
        if  x%2 == 0 then              // Skip every odd pixels
            putPixel (x, y)
        end
    end
end
```

2.2.3 DDA vs. Bresenham's Line Drawing Algorithm

Q. Compare DDA and Bresenham's line drawing algorithm.

Q. Differentiate: DDA vs. Bresenham.

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

Sr. No.	DDA Algorithm	Bresenham's Algorithm
1.	Involves floating point calculation.	Purely based on integer calculation.
2.	Involves costly operations like multiplication and division.	Involves cheaper operations like addition and subtraction.
3.	Due to floating point operation, it is slower.	It is faster as it involves only integer calculation.
4.	Less accurate.	More accurate.
5.	DDA performs rounding off operation of each pixel.	Bresenham's algorithm does not perform a rounding operation.
6.	Expensive due to extensive multiplication and division operations.	Less expensive as it computes the points on line using addition and subtraction.

2.3 Circle Generation Algorithms

- The circle is another very useful and important output primitive. Almost all graphics packages provide the functionality to plot the circle.
- Implicit equation of a circle centered at the origin is given by $x^2 + y^2 = r^2$, where r is the radius of the circle and (x, y) is any pair of the coordinate on circle boundary which satisfies the above equation.

- If the circle is centered at the point (x_c, y_c), it should be translated to origin to plot the circle. The equation of the circle, in this case, would be, $(x - x_c)^2 + (y - y_c)^2 = r^2$.

2.3.1 Symmetry of Circle

- It is simple to derive an explicit equation to draw a circle using its implicit representation in form of $y = f(x)$.

$$\therefore \quad y^2 = r^2 - x^2$$

$$\therefore \quad y = \pm\sqrt{r^2 - x^2}$$

- Varying value of x from 0 to r, we can get the corresponding value of y. This is a simple but inefficient way of plotting circle. This method involves square and square root operation for every pixel.
- Apart from this, pixels on circle would not be equispaced. Points near to x = 0 would be dense and pixels at x = r would have a large gap between them. (As shown in Fig. 2.3.1)
- However, we can reduce the computation by 1/8 if we use 8-way symmetry property of circle. We only need to compute pixel of arc from x = 0 to x = y (first octant).

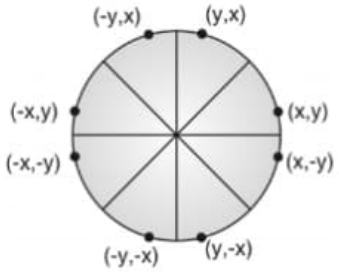


Fig. 2.3.1 : 8-Way symmetricity of circle

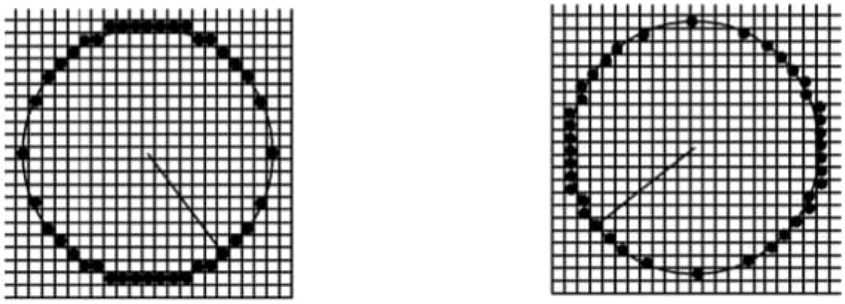
- Rest of all pixels in the remaining seven octants can be generated just by changing sign and/or coordinate positions. Fig. 2.3.1 describes the 8-way symmetry property of the circle.

Algorithm :

```
Algorithm   INCREMENTAL_CIRCLE(r)
            x ← r
            y ← 0
while       x ≠ y do
            EightWaySymmetry (x, y)
            y ← |√r2 – x2|
            x ← x – 1
end
            EightWaySymmetry (x, y)
            putPixel (x, y)
            putPixel (x, – y)
            putPixel (– x, y)
            putPixel (– x, – y)
            putPixel (y, x)
            putPixel (y, – x)
            putPixel (– y, x)
            putPixel (– y, – x)
```

2.3.2 Polar Circle Drawing Algorithm

- Serious draw back of incremental approach is that it does not create points equally spaced on circle boundary. When radius is sampled in x direction and y is computed, it creates dense points near y = r and sparse points at y = 0 (refer Fig. 2.3.2(a)).
- Similarly, when radius is sampled in y direction and x is computed, it creates dense points near x = r and sparse points at x = 0 (refer Fig. 2.3.2(b)).



(a) Incrementing x and computing y **(b) Incrementing y and computing x**

Fig. 2.3.2 : Circles using an incremental approach

- Polar representation overcomes this problem by computing (x, y) coordinates using the trigonometric relationship.
- Let's consider the radius of the circle is r. Line joining origin and any point on circle border make an angle θ with
- X-axis. Using trigonometry rule,

$$\cos\theta = \frac{x}{r}$$

$$\sin\theta = \frac{y}{r}$$

So, $x = r\cos\theta$ and $y = r\sin\theta$.

- Value of θ controls the number of points on circle boundary. Value of θ varies from 0 to 360°. Larger increment creates fewer points and smaller increment produces more points.

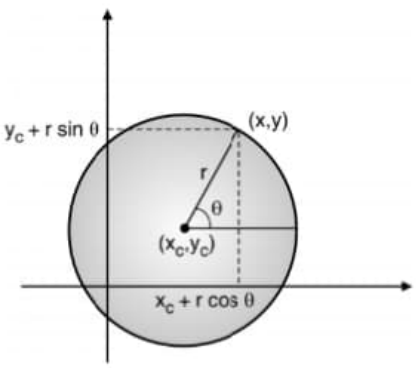


Fig. 2.3.3 : Polar representation of point (x, y)

- To create a smooth arc, we should set the value of θ in such a way so that it dynamically updated with the value of r.
- Experimental results show that the value of θ increment should be 1/r. With increment 1/r, we can create a smooth circle with an equally spaced point for any radius.
- Fig. 2.3.4 shows the circle generated using polar representation.

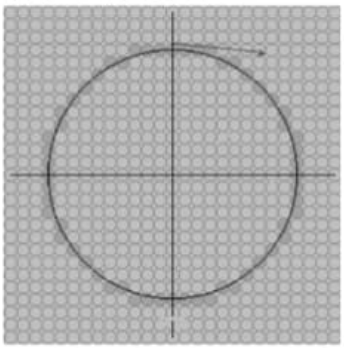


Fig. 2.3.4 : Circle using the polar equation

Q. List advantages and disadvantages of polar circle drawing method.

Advantages :

1. Easy to compute.
2. Creates a circle with uniformly distributed points on circle boundary.

Disadvantages :

1. Computationally expensive.
2. sin and cos trigonometric series is expanded for each pair of coordinates.
3. Two round operations are performed per pixel.
4. Pseudocode is shown to generate a circle using polar representation.

Algorithm Polar circle drawing :

Q. Write an algorithm for polar circle drawing.

```
Algorithm  POLAR_CIRCLE(r)
           thetaInc← 1/r
     for   theta ← 0 to 360 do
           x ← r*cos(theta)
           y ← r*sin(theta)
           theta ←theta + thetaInc
           putPixel (ROUND(x), ROUND(y))
     end
```

2.3.3 Bresenham's Circle Drawing Algorithms

W-18

Q. Derive the expression for decision parameter used in Bresenham's circle drawing algorithm. **(W-18, 6 Marks)**

- Bresenham has proposed integer calculation based circle drawing algorithm. This approach is simple and faster.
- The circle is eight-way symmetrical shape, so we will derive the formula for only one octant, circle in remaining 7 octants is produced by its symmetry property.
- If (x_k, y_k) is the current pixel, as shown in Fig. 2.3.5 there are two choices for the next pixel:
  1. East pixel E, i.e. (x_{k+1}, y_k)
  2. South East pixel SE, i.e. (x_{k+1}, y_{k-1})
- Explicit representation of the circle is

$$f(x, y) = x^2 + y^2 - r^2$$

  - If (x, y) is on circle, f (x, y) = 0
  - If (x, y) is an outside circle, f (x, y) > 0
  - If (x, y) is inside the circle, f (x, y) < 0

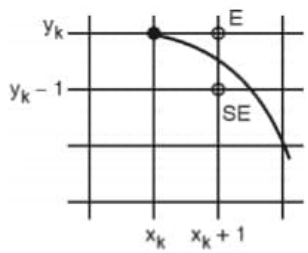


Fig. 2.3.5

– For point E and SE, equations of the circle would be,

$$f(E) = f(x_k + 1, y_k) = (x_k + 1)^2 + (y_k)^2 - r^2$$

$$f(SE) = f(x_k + 1, y_k - 1) = (x_k + 1)^2 + (y_k - 1)^2 - r^2$$

Based on relationship between circle arc, point E and point SE, we have five different cases to choose the next pixel form current pixel (x_k, y_k).

All five cases are depicted in Fig. 2.3.6.

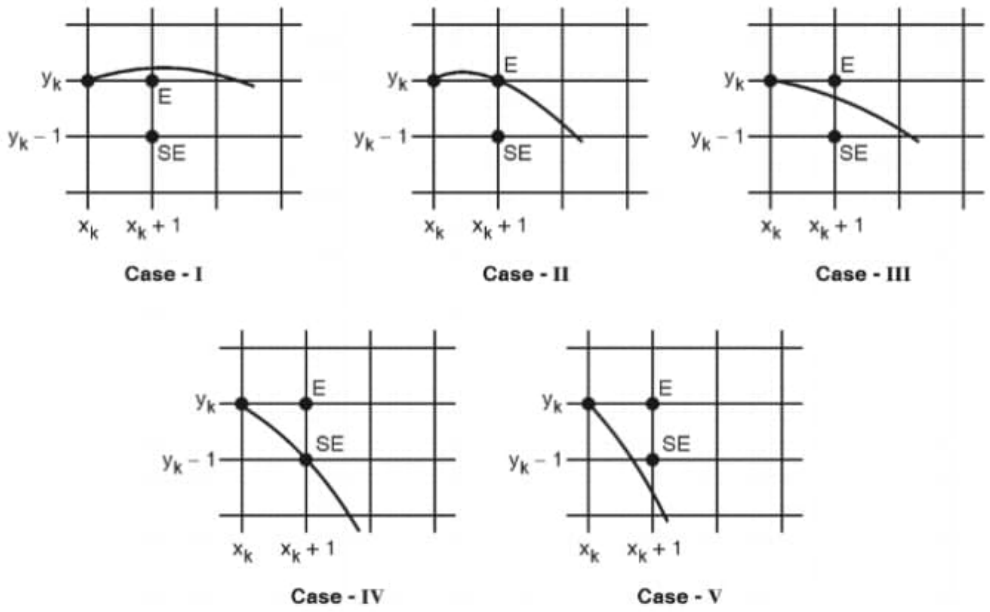


Fig. 2.3.6

According to the relation between are E and SE, f (E) and f (SE) would be positive, zero or negative. Table 2.3.1 sign for all five cases.

Table 2.3.1

Case	I	II	III	IV	V
f (E)	– ve	0	+ ve	+ ve	+ ve
f (SE)	– ve	– ve	– ve	0	+ ve
sum	– ve	– ve	– ve, 0, + ve	+ ve	+ ve

– Case I and case V are trivial. Both points are inside or outside the circle in respective cases.

– If both points are inside the circle (Case - I), the next pixel would be E (x_{k+1}, y_k), because arc is close to E.

– If both points are outside the circle, next pixel would be SE (x_{k+1}, y_{k-1}), because arc is close to SE.

– Cases II and IV are also simple. In case to II, f (E) = 0. So circle passes through the point (x_{k+1}, y_k).

– In case IV, f (SE) = 0, so circle passes through point (x_{k+1}, y_{k-1}).

- From Table 2.3.1 and above discussed four cases, it is simple to note that if the sum is negative, select East pixel, if the sum is positive, then select South East pixel.
- In case III, the sum has all three possible values. So according to sign, we shall select E or SE as the next pixel. Let us derive the decision parameter for subsequent pixel choice.
- If we start plotting circle from initial point $(x_0, y_0) = (0, r)$ then

$$f(E) = f(x_0 + 1, y_0) = (0+1)^2 + (r)^2 - r^2 = 1 + r^2 - r^2 = 1$$

$$f(SE) = f(x_0 + 1, y_0 - 1) = (0+1)^2 + (r-1)^2 - r^2$$

$$= 1 + r^2 - 2r + 1 - r^2 = 2 - 2r$$

$$\text{Initial decision parameter, } S = f(E) + f(SE) = 1 + 2 - 2r$$

$$S = 3 - 2r$$

- From (0, r), we select (1, r) or (1, r – 1) depending on the sign of sum.
- Let us generalize this decision parameter for an arbitrary point (x_k, y_k) on the circle. Consider p_k is the current pixel.

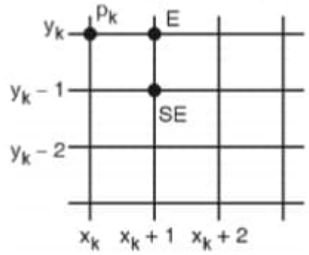


Fig. 2.3.7

$$S_{old} = f(E) + f(SE)$$

$$= f(x_k + 1, y_k) + f(x_k + 1, y_k - 1)$$

$$S_{old} = (x_k + 1)^2 + (y_k)^2 - r^2 + (x_k + 1)^2 + (y_k - 1)^2 - r^2$$

1. If E pixel is selected as next pixel :

If $S_{old} < 0$, we shall select E $(x_k + 1, y_k)$ as next pixel on arc. Consider E as current pixel, we have two choices from E, i.e., $(x_k + 2, y_k)$ or $(x_k + 2, y_k - 1)$.

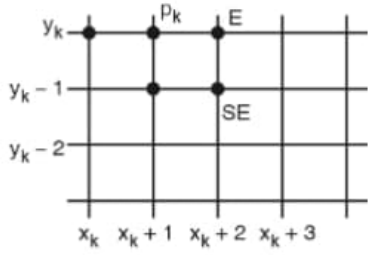


Fig. 2.3.8

$$S_{new} = f(E) + f(SE) = f(x_k + 2, y_k) + f(x_k + 2, y_k - 1)$$

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

$$= (x_k^2 + 4x_k + 4) + y_k^2 - r^2 + (x_k^2 + 4x + 4) + (y_k - 1)^2 - r^2$$

$$= (x_k^2 + 2x_k + 1) + 2x_k + 3 + y_k^2 - r^2 + (x_k^2 + 2x_k + 1) + 2x_k + 3 + (y_k - 1)^2 - r^2$$

$$= [(x_k + 1)^2 + (y_k)^2 - r^2 + (x_k + 1)^2 + (y_k - 1)^2 - r^2] + 4x_k + 6$$

$$S_{new} = S_{old} + 4x_k + 6$$

From point $p_k (x_k + 1, y_k)$, select pixel E $(x_k + 2, y_k)$ if $S_{new} \leq 0$, otherwise select SE $(x_k + 2, y_k - 1)$.

2. If SE is selected as next pixel :

In Fig. 2.3.9, if $S_{old} > 0$, we shall select SE as next pixel. In next iteration, consider SE as current pixel, we have two choices from current pixel $(x_k + 1, y_k - 1)$ i.e. $(x_k + 2, y_k - 1)$ or $(x_k + 2, y_k - 2)$.

From Fig. 2.3.9

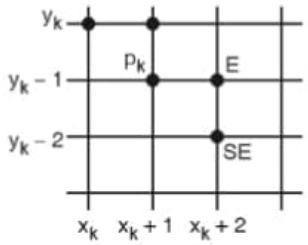


Fig. 2.3.9

$$S_{new} = f(E) + f(SE)$$

$$= f(x_k + 2, y_k - 1) + f(x_k + 2, y_k - 2)$$

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

$$= (x_k^2 + 4x_k + 4) + (y_k^2 - 2y_k + 1) - r^2 + (x_k^2 + 4x_k + 4) + (y_k^2 - 4y_k + 4) - r^2$$

$$= [(x_k^2 + 2x_k + 1) + 2x_k + 3] + [y_k^2 - 2y_k + 1] - r^2 + [(x_k^2 + 2x_k + 1) + 2x_k + 3] + [(y_k^2 - 2y_k + 1) - 2y_k + 3] - r^2$$

$$S_{new} = \left[(x_k^2 + 2x_k + 1) + y_k^2 - r^2 \right] + \left[(x_k^2 + 2x_k + 1) + (y_k^2 - 2y_k + 1) - r^2 \right] + 4x_k - 4y_k + 10$$

$$S_{new} = S_{old} + 4(x_k - y_k) + 10$$

Steps of Bresenham's circle generation algorithm :

Q. Mention the steps for Bresenham's circle generation algorithm.

Step 1 : Read radius r and circle center (x_c, y_c)

Step 2 : Plot the initial pixel $(x, y) = (x_0, y_0) = (0, r)$

Step 3 : Compute initial decision parameter $S = 3 - 2r$

Step 4 : Repeat step 4 to step 6 till $x \leq y$

If $S < 0$ then

$$x = x + 1$$

$$\text{Set } S = S + 4x + 6$$

else

$$x = x + 1$$

$$y = y - 1$$

Set $S = S + 4(x - y) + 10$

Step 5 : Compute remaining seven symmetry points

Step 6 : Plot $(x_c + x, y_c + y)$

Bresenham's circle drawing algorithm :

Q. Write an algorithm for Bresenham's circle drawing algorithm.

Bresenham's line drawing algorithm is described below :

```
Algorithm  BRESENHAM_CIRCLE (xc, yc, r)
           x ← 0
           y ← r
           S← 3 – 2r
           EightWaySymmetry(x, y)
    while  (x ≤ y) do
           x ←x + 1
    if     S< 0 then
           S    ←    S + 4x + 6
    else
           y    ←    y– 1
           S    ←    S + 4 (x – y) + 10
    end
           EightWaySymmetry (x, y)
end
```

Example 2.3.1 : Plot a circle using Bresenham's algorithm for r = 3 and center at (0, 0).

Solution :

Here r = 3, $x_c = 0$, $y_c = 0$

First point to plot is (0, r) = (0, 3)

Using symmetry property of circle, we can easily compute the remaining points, which will be (0, –3), (3, 0) and (–3, 0).

Let's compute the remaining points on circle.

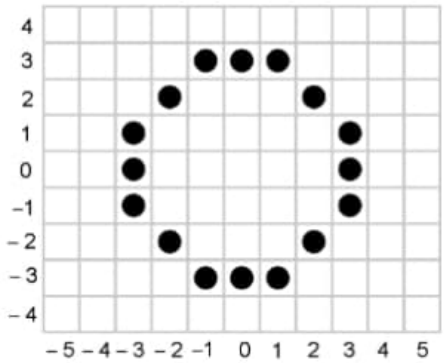
Initial decision parameter r = 3 – 2r = 3 – 6 = –3

Table P. 2.3.1

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	-	-	0	3	S = 3 – 2r = 3 – 6 = –3
1	(0, 3)	S< 0, so select E	0 + 1 = 1	3 + 0 = 3	$S = S + 4x_k + 6 = -3 + 0 + 6 = 3$
2	(1, 3)	S> 0, so select SE	1 + 1 = 2	3–1 = 2	x = y so stop
3	(2, 2)	-	-	-	-

Fig. P. 2.3.1 shows the final points on circle.

X	Y
0	3
1	3
2	2



(a) Points computed in one octant **(b) Points generated using 9-way symmetry**

Fig. P. 2.3.1 : Circle generation for given data

2.4 Polygons

Many times basic primitives like point, line or triangles are not sufficient to describe the arbitrary shape. Real world objects are often complex. The best way to describe them is to use polygons. In this section, we will study different types of polygons and the methods to fill them.

2.4.1 Types of Polygons

W-18

Q.	List types of Polygon.	**(W-18, 2 Marks)**
Q.	Name with an example, different types of polygons.	
Q.	What is a polygon? What are the different types of polygons?	

- **Polygon** is a closed figure made up of connected lines. Polygon has many sides and is represented by line segments. Line segments forming the boundary of the polygon are called **edges** of polygon and the endpoints of edges are known as **vertices** of the polygon.
- Triangle is the polygon with a minimum number of edges. Polygon is always closed figure; hence less than three sides cannot form the polygon. The circle is closed figure but it does not have edges, so it is not a polygon. Few other examples of polygons are shown in the Fig. 2.4.1.

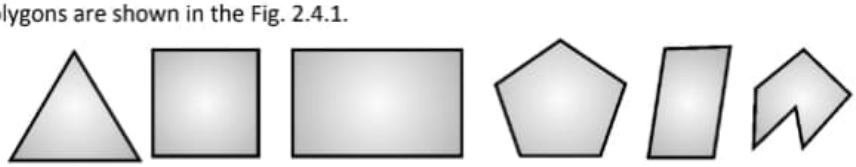


Fig. 2.4.1 : Polygon with different shapes and sides

We can classify polygon in one of the following categories :

- Convex polygon
- Concave polygon
- Complex polygon

2.4.1.1 Convex Polygon

- In a convex polygon, the angle between any consecutive pair of edges is always less than 180^{o}.
- For a convex polygon, if we select any random two points inside the polygon and we join them through a line; all the points on the line are certainly inside the polygon.
- Examples of the convex polygon are shown in the Fig. 2.4.2.

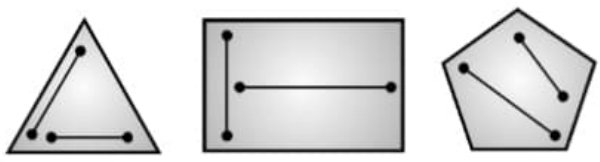


Fig. 2.4.2 : Example of convex polygons

2.4.1.2 Concave Polygon

- In a concave polygon, the angle between at least one pair of consecutive edges is greater than 180°.
- For a concave polygon, there always exists a pair of vertices inside the polygon, when joined through a line segment, part of line segment falls outside the polygon.
- Examples of the concave polygon are shown in the Fig. 2.4.3.

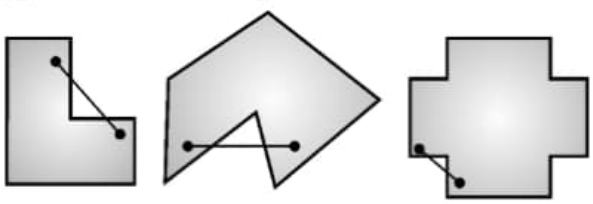


Fig. 2.4.3 : Examples of concave polygons

2.4.1.3 Complex Polygons

- Polygons which are neither convex nor concave are called complex polygon. Complex polygons are self-intersecting or overlapping.
- Examples of the complex polygon are shown in the Fig. 2.4.4

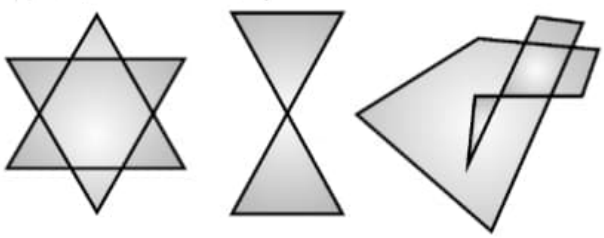


Fig. 2.4.4 : Examples of complex polygons

2.4.2 Inside-Outside Test

Q.	Explain two methods for testing whether the point is inside the polygon or not.
Q.	Explain the different methods for testing a pixel inside a polygon.

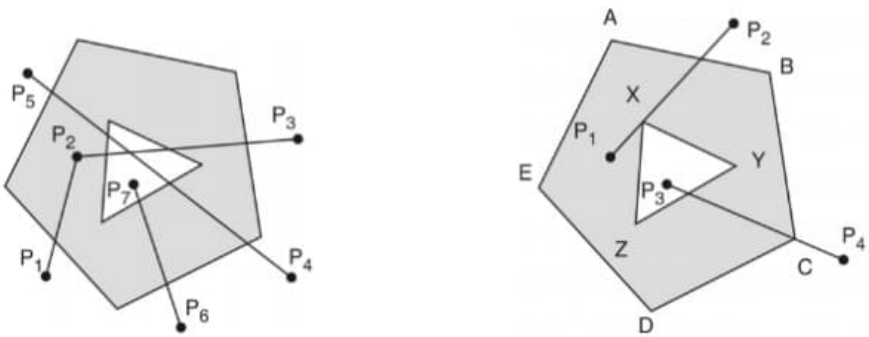
Deciding membership of pixel is very important in polygon filling. Accurate detection of whether the point is inside or outside of polygon is crucial. We will discuss two methods to decide if a pixel is inside the polygon.

- Even-odd method
- Winding number method

2.4.2.1 Even–Odd Method

Q. Describe the Even-Odd test method

- This method constructs a line from a given point to the known point outside the polygon and checks the intersection of line with polygon boundary.
- Let's check the visibility of point P_2 (Refer Fig. 2.4.5(a)). Draw a line from P_2 to known exterior point to a polygon in any direction and count number of intersections of line with polygon edges.
- If count is odd, then point is inside polygon otherwise it is outside of the polygon.
- Line P_2P_1 intersects polygon edges only once, while line P_2P_3 intersects polygon edge thrice, in both the cases count is odd, means the point is inside the polygon.
- Consider the point P_4, draw a line to known exterior point P_5, it intersects polygon boundary four times, so P_4 is considered as an exterior point. Same is true for point P_7.



(a) Simple case **(b) Complex case**

Fig. 2.4.5 : Odd-even test

- This rule is very simple, but the problem arises when the line crosses the common vertex shared by two adjacent edges.
- Consider the Fig. 2.4.5(b), line P_3P_4 intersects the outer boundary of the polygon at common vertex shared by edges BC and CD. So even though P_3P_4 intersects three polygon edges (YZ, BC and CD), P_3 is an exterior point.
- The solution to this problem is to count intersection as a single intersection, if both edges are on the opposite side of the line. For example, CD and BC in case of line P_3P_4.
- If both edges are on the same side of line segment, then count it as two intersections.
- For example, XY and XZ are on the same side of line P_1P_2, so count it as two intersection points.
- It is easy to understand from Fig. 2.4.6, where to count once (odd count) and where to count twice (even count).

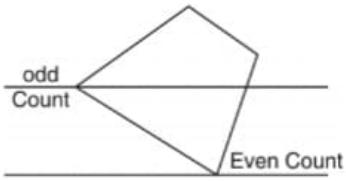


Fig. 2.4.6 : Remedy to shared edge

2.4.2.2 Winding Number Method

Q. Explain the winding number method.

- Winding number method is another way to define the visibility of pixel.
- Like even-odd method, draw a straight line from point to be examined to any known exterior point.
- Visit outer most boundary of the polygon in anti-clockwise direction and inner boundary (if exists) in clockwise direction.
- Find the intersection points of line with polygon boundaries and check how many times polygon edge crosses the line while traversing from left to right and how many times it crosses the line while traversing from right to left.
- Take the difference of these two counts, if the difference is zero, the point is outside the polygon, otherwise, it is considered as an inner point.
- Consider the point P_2, (Refer Fig. 2.4.7) which is joined with exterior point P_1.
- Polygon edge AB intersects P_1P_2 while traversing from left to right. So point P_2 is the interior point.
- Edge YZ intersect line P_6P_7 from right to left and edge BC intersects it while traversing from left to right. The difference of intersection count is zero and hence P_7 is considered as an outside pixel.

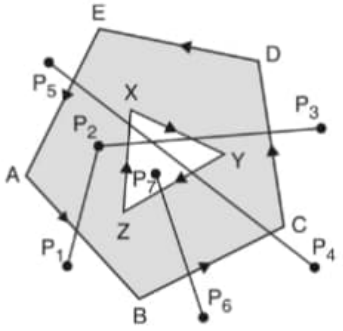


Fig. 2.4.7 : Winding number method

- Similarly edges EA and YZ intersect line P_4P_5 while traversing from right to left and edges ZX and BC intersects P_4P_5 while traversing from left to right. The difference is zero, so point P_4 is outside the polygon.

2.5 Polygon Filling

W-18

Q. List various polygon filling algorithms. **(W-18, 2 Marks)**

Polygon filling algorithms fill the background of polygons. Most popular algorithms of polygon filling are listed and discussed in the following section.

- Pixel level algorithms
  - Seed fill algorithms
    - (i) Boundary fill algorithm
    - (ii) Flood fill algorithm
  - Edge fill algorithm
  - Fence fill algorithm
- Geometry level algorithms
  - Scan line polygon filling algorithm

2.5.1 Pixel Connectivity

- Connectivity defines the relationship between pixel under consideration and its neighbor pixels.
- Most common and useful types of connectivity are shown in Fig. 2.5.1.
- Pixel is called four connected if it has four neighbors and it is called 8-connected if it has eight neighbors.
- Fig. 2.5.1 defines the 4-connected and 8-connected component.

(a) 4 Connected **(b) 8 Connected**

Fig. 2.5.1 : Pixel connectivity

2.5.2 Seed Fill Algorithms

Scan line algorithm fills the visible span on each scan line. Seed fill algorithms select interior pixel of the polygon as a seed and grow gradually. Algorithms examines the neighbours of each pixel, if it is not filled with specified color or boundary color then fill it with the givencolor. This process is recursively called until all interior pixels are filled with the specifiedcolor. We will discuss two approaches of the seed fill algorithm.

2.5.2.1 Boundary Fill Algorithm

W-18

Q. Explain the boundary fill algorithm with pseudo-code. Also mention its limitations, if any. **(W-18, 6 Marks)**

- Boundary fill algorithm starts with some interior pixel of polygon, called *seed pixel* and keep filling neighbor pixels in outward directions until the boundary color is encountered.
- Boundary fill algorithm starts with interior point (x, y), fill color and boundary color.
- This approach h retrieves the color of the current pixel and compares it with fill color and boundary color.
- If the color of the current pixel is neither fillcolor nor boundary color, then fill it with the fill color and make recursive call, otherwise skip the pixel under observation.
- Neighbour pixels are approached using 4-connectivity or 8-connectivity.
- Sometimes 4-connectivity fails to paint the entire region. But 8-connectivity is very time and memory intensive.
- This method is useful in interactive painting packages, where selection of interior pixel can be done very easily using input device like mouse.
- To create a solid region, set the fill color to boundary color, so that boundary and interior region becomes undistinguishable after filling.
- Recursively this algorithm checks all pixels in given polygon and fill them if not already filled.

Q. Write an algorithm for boundary fill.

Algorithm :

– Following pseudo code describes how boundary fill algorithm fills the region.

```
Algorithm  BOUNDARY_FILL (x, y, fillColor, boundaryColor)
           // filColor: Color to be filled in polygon
           //boundaryColor: Color of polygon boundary

           current←getColor (x, y)        // Retrieve the color of current pixel
     if    current ≠fillColor&& current≠boundaryColorthenthen
           putPixel (x, y, fillColor)
           BOUNDARY_FILL (x + 1, y, fillColor, boundaryColor)
           BOUNDARY_FILL (x – 1, y, fillColor, boundaryColor)
           BOUNDARY_FILL (x, y + 1, fillColor, boundaryColor)
           BOUNDARY_FILL (x, y – 1, fillColor, boundaryColor)
end
```

– This algorithm may not work properly if some of the pixels are already filled, because algorithm returns when the current pixel is either boundary pixel or it is filled.

– To avoid this, we shall set the color of all interior pixels to background color.

Q. State merits and demerits of boundary fill algorithm

Advantages :

– Simple.

– Easy to implement.

– Works for any type of polygons.

Disadvantages :

– Require extensive stacking due to recursion.

– Does not work if the boundary is specified with multiple colors.

– Need to specify seed point contained within the polygon.

– Polygon should be filled with the backgroundcolor.

– Uses extensive stacking.

– Require more memory.

– Not suitable for the large polygon.

– Filling the region with 4 and 8 connectivity.

– Filling of polygon depends on types of connectivity employed. The 4-connected approach does four recursive calls for each of its four nearest neighbors.

– The 8-connected approach does eight calls. The 4-connected approach requires less memory but sometimes fails to fill certain parts of the polygon.

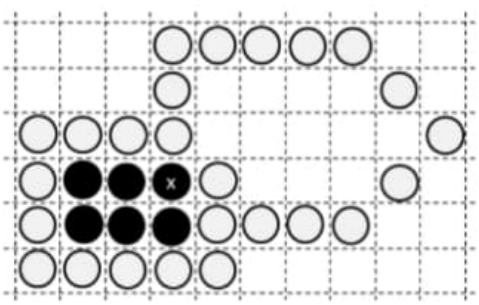


Fig. 2.5.2

- In Fig. 2.5.2, if the 4-connected approach is used then it terminates after reaching to pixel marked with x. It cannot jump into another part of the polygon. But if the 8-connected approach is used, the algorithm can jump diagonally too. Hence it would be able to fill both the regions of the polygon.

Example 2.5.1 : Which algorithm cannot be used to fill region R2 which is bounded by Blue color and Red color boundary, justify.

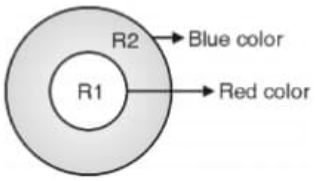


Fig. P. 2.5.1

Solution :

- Recursive call of boundary fill algorithm returns when a current pixel is either already filled or it has boundary color. It takes fill color and *one* boundary color as an argument.
- Boundary fill algorithm can either take blue or red color as an argument. Hence, for the region with multiple boundary color cannot be filled using boundary fill algorithm.
- Whereas, the flood fill algorithm fills pixel if it has a backgroundcolor. It does not check the boundary color. Hence, it can manage any number of boundary colors.

2.5.2.2 Flood Fill Algorithm

Q. Explain the flood fill algorithm with algorithm.

- Many realistic objects have different colored boundary. Boundary fill algorithm cannot fill the polygon with multi-colored boundary.
- Flood fill can fill the polygon with multiple boundary color.
- Flood fill algorithm is recursive in nature.
- The algorithm starts with the interior pixel, fill color and old color. If the color of the current pixel is the same as old color then it fills the pixel with fill color and examine its neighbors.
- Like boundary fill algorithm, if the interior pixel is already filled with some other color, flood fill algorithm fails. To handle this problem, we shall first paint all interior pixels with background (old) color.

Q. Write algorithm to fill the polygon area using flood fill method.

Algorithm :

```
Algorithm  FLOOD_FILL (x, y, fillColor, oldColor)
           current ←getColor (x, y)
    if     current ≠oldColorthenthen
           putPixel (x, y, fillColor)
           FLOOD_FILL (x + 1, y, fillColor, oldColor)
           FLOOD_FILL (x – 1, y, fillColor, oldColor)
           FLOOD_FILL (x, y + 1, fillColor, old,Color)
           FLOOD_FILL (x, y – 1, fillColor, oldColor)
    end
```

Advantages :

- Simple.
- Easy to implement.
- Boundary fill algorithm cannot handle the object with multi color boundary, whereas flood fill can.

Disadvantages :

- Require extensive stacking.
- Slow in nature
- Not suitable for large polygon

Optimization :

- Boundary fill and flood fill, both are the recursive approaches. On every call, they make another four calls to the neighbours of the current pixel, this creates extensive stacking. The system runs out of memory if polygon to be filled is very large.

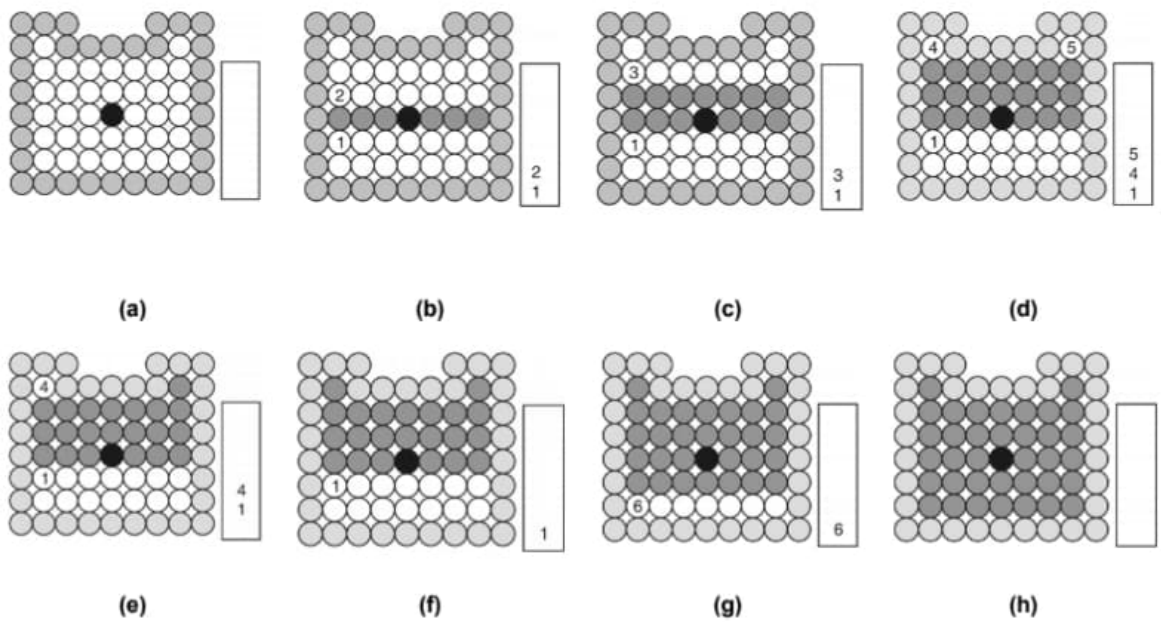


Fig. 2.5.3 : Optimized seed fill approach

- We will discuss a more efficient method which will minimize the recursive calls. Instead of pushing all four neighbors on the stack, just push neighbor scan line numbers on the stack and fill the entire span of the current scan line.
- Following a sequence of figure explains the concept. In Fig. 2.5.3(a), random seed pixel is selected. The span of the current scan line is filled up and addresses of its upper and lower scan line are pushed on to the stack. The procedure always pops topmost scan line number from the stack, fills it horizontally and pushes the above and below scan line numbers if they are not already processed. This approach greatly simplifies the processing and reduces memory requirement.

2.5.3 Scan Line Polygon Filling Algorithms

- This method computes the visible span of each scan line within polygon boundaries. It works with convex, concave, self-intersecting and polygon with holes.
- The scan line intersects polygon boundary at a, b, c and d (Refer Fig. 2.5.4). We shall set appropriate values of all intersection points to fill the polygon properly.
- Midpoint line algorithm can be used to determine the intersection point of the scan line with the edges.
- This approach selects the pixel which is closer to the edge, irrespective of on which side of polygon it lies. If the edge is shared between two polygons, and the outer pixel is selected for drawing, this would look odd if a neighbor polygon has a differentcolor. So even if the exterior pixel is closer to the edge, we should reject it for better rasterization.

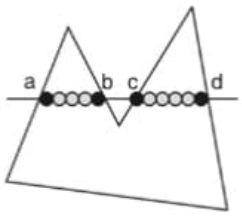


Fig. 2.5.4 : Visible span of the scan line

- A simple solution to this problem is to round up the intersection pixel coordinate if scan line is entering in polygon region, and round down if it is leaving the polygon.
- Scan line polygon filling approach works as follow :
  - Find intersections of scan line with polygon edges.
  - Sort all intersections on their x coordinate.
  - Fill the span using a pair of sorted intersection points.
- We use parity rule to determine the intersection of scan line with shared vertex as a single point or two points. Horizontal edge is the special case. Parity changes on every pixel for horizontal edge and every alternative pixel is illuminated. To avoid this effect, horizontal edges are not considered for processing.

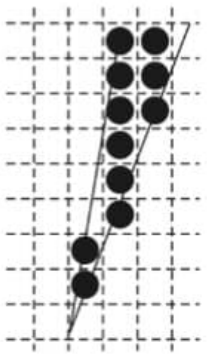


Fig. 2.5.5 :Sliver

- Another problem with this approach is a sliver. When the angle between adjacent edges is very small, consecutive scan line produces the same span, which results in aliasing effect. This is called a **sliver**. Fig. 2.5.5 demonstrates sliver.
- Scan line approach takes advantage of edge coherence. Most of the time, edges intersecting scan line k are also intersected by scan line k + 1. If the current intersection point is (x_k, y_k), the intersection point on the next scan line is computed as,

$$y_k + 1 = y_k + 1$$

$$x_k + 1 = x_k + \frac{1}{m}$$

- This computation is identical to a line drawing algorithm.
- This approach uses two data structures, called Edge Table (ET) and Active Edge Table (AET). Edge table maintains the following information for each edge in a polygon, except horizontal edges.

Y_{MAX}	X_{MIN}	$\frac{1}{m}$	*ptr

- Bucket sort algorithm is used to build an edge table. Each entry in the edge table maintains edge list in increasing order of their x coordinate. Each bucket holds the above mentioned three values of a particular edge.

After setting up the edge table, scan line algorithm works as follow :

1. Set Y_{scan} to smallest y for which there is an entry in ET.
2. AET is initially empty.
3. Repeat following steps until both, AET and ET are empty :
   (i) Move edges from ET to AET whose $y_{min} = Y_{scan}$.
   (ii) Fill the visible span.
   (iii) Remove edges with $y_{max} = y_{scan}$.
   (iv) Increment y by 1.
   (v) For each edges in AET, update x coordinate.
   (vi) Sort all edges on x.

– This algorithm optimizes the computation by using edge coherence to compute x and scan line coherence to compute visible span. Sorting in step 3(vi) operates on a small number of edges and on the almost sorted list, so bubble sort or insertion sort would be a good choice to sort them in roughly O(N) time.

2.5.4 Seed Fill vs. Scan Line Algorithm

Sr. No.	Seed fill algorithm	Scan line algorithm
1.	Simple to implement	Very complex in nature
2.	Operates in screen space	Operates in screen and / or object space
3.	Require system call to get pixel intensity	It is device independent
4.	Need to specify seed point contained within the polygon	No need to specify seed point
5.	Need extensive stacking	No need of stacking
6.	Generally used in graphics packages	Typically used in interactive rendering
7.	Not suitable for Z-Buffer	Suitable for Z-Buffer

Example 2.5.2 : Determine the visible span for each scan line for given polygon using scan line polygon filling algorithm.

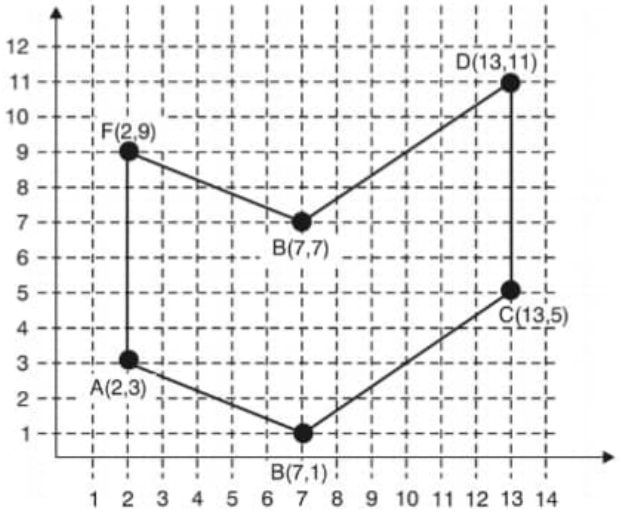


Fig. P. 2.5.2

Solution :

Edge table for a given polygon is shown in Fig. P. 2.5.2(a).

Let us compute the visible span for each scan line.

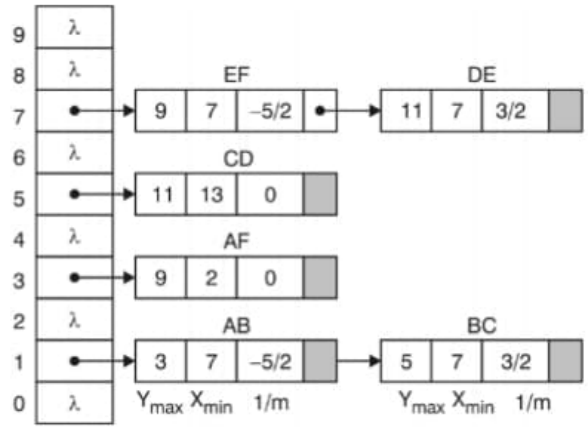


Fig. P. 2.5.2(a)

Scan line Y = 1 : AB and BC edges are added

- **Edge AB :** $Y_{scan}<Y_{max}(AB)$, So continue

  $X_{AB} = 7$, So $(X_{AB}, Y_{SCAN}) = (7, 1)$

- **Edge BC :** $Y_{scan}<Y_{max}(BC)$, So continue

  $X_{BC} = 7$, So $(X_{BC}, Y_{SCAN}) = (7, 1)$

  Draw pixels from (7, 1) to (7, 1)

Scan line Y = 2 : No new Edges are added

- **Edge AB :** $Y_{scan}<Y_{max}(AB)$, So continue

  $X_{AB} = X_{AB} + (1/m) = 7 - 2.5$

  $= 4.5$, So $(X_{AB}, Y_{SCAN}) = (5, 2)$

- **Edge BC :** $Y_{scan}<Y_{max}(BC)$, So continue

  $X_{BC} = X_{BC} + (1/m) = 7 + 1.5 = 8.5$, So $(X_{BC}, Y_{SCAN}) = (8, 2)$

  Draw pixels from (5, 2) to (8, 2)

Scan line Y = 3 : AF Edge is added

- **Edge AB :** $Y_{scan} \geq Y_{max(AB)}$, So remove AB
- **Edge BC :** $Y_{scan}<Y_{max(BC)}$, So continue

  $X_{BC} = X_{BC} + (1/m) = 8.5 + 1.5 = 10$, So $(X_{BC}, Y_{SCAN}) = (10, 3)$

- **Edge AF :** $Y_{scan}<Y_{max(AF)}$, So continue

  $X_{AF} = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 3)$

  Draw pixels from (2, 3) to (10, 3)

Scan line Y = 4 : No New Edges are added

- **Edge BC :** $Y_{scan}<Y_{max(BC)}$, So continue

  $X_{BC} = X_{BC} + (1/m) = 10 + 1.5 = 11.5$, So $(X_{BC}, Y_{SCAN}) = (11, 4)$

- **Edge AF :** $Y_{scan}<Y_{max(AF)}$, So continue

  $X_{AF} = X_{AF} + (1/m) = 2 + 0 = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 4)$

  Draw pixels from (2, 4) to (11, 4)

Scan line Y = 5 : CD Edge is added

- **Edge BC :** $Y_{scan} \geq Y_{max(BC)}$, So remove BC
- **Edge AF :** $Y_{scan}<Y_{max(AF)}$, So continue

  $X_{AF} = X_{AF} + (1/m) = 2 + 0 = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 5)$

- **Edge CD :** $Y_{scan}<Y_{max(CD)}$, So continue

  $X_{CD} = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 5)$

  Draw pixels from (2, 5) to (13, 5)

Scan line Y = 6 : No New Edges are added

- **Edge AF :** $Y_{scan} < Y_{max(AF)}$, So continue

  $X_{AF} = X_{AF} + (1/m) = 2 + 0 = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 6)$

- **Edge CD :** $Y_{scan} < Y_{max(CD)}$, So continue

  $X_{CD} = X_{CD} + (1/m) = 13 + 0 = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 6)$

  Draw pixels from (2, 6) to (13, 6)

Scan line Y = 7 : EF and DE Edges are added

- **Edge AF :** $Y_{scan} < Y_{max(AF)}$, So continue

  $X_{AF} = X_{AF} + (1/m) = 2 + 0 = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 7)$

- **Edge CD :** $Y_{scan} < Y_{max(CD)}$, So continue

  $X_{CD} = X_{CD} + (1/m) = 13 + 0 = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 7)$

  EF and DE Edges are added.

- **Edge EF :** $Y_{scan} < Y_{max(EF)}$, So continue

  $X_{EF} = 7$, So $(X_{EF}, Y_{SCAN}) = (7, 7)$

- **Edge DE :** $Y_{scan} < Y_{max(DE)}$, So continue

  $X_{DE} = 7$, So $(X_{DE}, Y_{SCAN}) = (7, 7)$

  Draw pixels from (2, 7) to (7, 7) and (7, 7) to (13, 7)

Scan line Y = 8 : No New Edges are added

- **Edge AF :** $Y_{scan} < Y_{max(AF)}$, So continue

  $X_{AF} = X_{AF} + (1/m) = 2 + 0 = 2$, So $(X_{AF}, Y_{SCAN}) = (2, 8)$

- **Edge CD :** $Y_{scan} < Y_{max(CD)}$, So continue

  $X_{CD} = X_{CD} + (1/m) = 13 + 0 = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 8)$

- **Edge EF :** $Y_{scan} < Y_{max(EF)}$, So continue

  $X_{EF} = X_{EF} + (1/m) = 7 - 2.5 = 4.5$, So$(X_{EF}, Y_{SCAN}) = (4, 8)$

- **Edge DE :** $Y_{scan} < Y_{max(DE)}$, So continue

  $X_{DE} = X_{DE} + (1/m) = 7 + 1.5 = 8.5$, So $(X_{DE}, Y_{SCAN}) = (9, 8)$

  Draw pixels from (2, 8) to (4, 8) and (9, 8) to (13, 8)

Scan line Y = 9 : No New Edges are added

- **Edge AF :** $Y_{scan} \geq Y_{max(AF)}$, So Remove AF

- **Edge CD :** $Y_{scan} < Y_{max(CD)}$, So continue

  $X_{CD} = X_{CD} + (1/m) = 13 + 0 = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 9)$

- **Edge EF :** $Y_{scan} \geq Y_{max(EF)}$, So remove EF

- **Edge DE :** $Y_{scan} < Y_{max(DE)}$, So continue

  $X_{DE} = X_{DE} + (1/m) = 8.5 + 1.5 = 10$, So $(X_{DE}, Y_{SCAN}) = (10, 9)$

  Draw pixels from (10, 9) to (13, 9)

Scan line Y = 10 : No New Edges are added

- **Edge CD :** $Y_{scan} < Y_{max(CD)}$, So continue

  $X_{CD} = X_{CD} + (1/m) = 13 + 0 = 13$, So $(X_{CD}, Y_{SCAN}) = (13, 10)$

- **Edge DE :** $Y_{scan} < Y_{max(DE)}$, So continue

  $X_{DE} = X_{DE} + (1/m) = 10 + 1.5 = 11.5$, So $(X_{DE}, Y_{SCAN}) = (12, 10)$

  Draw pixels from (12, 10) to (13, 10)

Scan line Y = 11 : No New Edges are added

- **Edge CD :** $Y_{scan} \geq Y_{max(CD)}$, So remove CD
- **Edge DE :** $Y_{scan} < Y_{max(DE)}$, So remove DE

  AET is empty, so STOP

Scan Line	Visible Span
$Y_{SCAN} = 1$	(7, 1) to (7, 1)
$Y_{SCAN} = 2$	(5, 2) to (8, 2)
$Y_{SCAN} = 3$	(2, 3) to (10, 3)
$Y_{SCAN} = 4$	(2, 4) to (11, 4)
$Y_{SCAN} = 5$	(2, 5) to (13, 5)
$Y_{SCAN} = 6$	(2, 6) to (13, 6)
$Y_{SCAN} = 7$	(2, 7) to (7, 7) and (7, 7) to (13, 7)
$Y_{SCAN} = 8$	(2, 8) to (4, 8) and (9, 8) to (13, 8)
$Y_{SCAN} = 9$	(10, 9) to (13, 9)
$Y_{SCAN} = 10$	(12, 10) to (13, 10)

2.6 Scan Conversion

Q. What do you mean by scan conversion ?

- **Scan conversion** is a process of computing pixel locations of primitives and displaying it on the monitor screen.
- Geometric shapes are often defined by parameters. For example, line is defined by its endpoints; circle is defined by its centre and radius.
- Application program takes these parameters as input and generates all the pixels representing the shape. This process is called scan conversion.
- All the points on line can be generated using DDA or Bresenham's algorithm for given endpoints.
- Most of the monitors and television sets support raster scan display. In raster scan devices, picture definition is stored as a discrete pixel value. The entire scene is defined by a set of intensity values for each pixel on the screen.
- Computed pixel values may be fraction, so it needs to round off to show it on monitor screen.
- As shown in Fig. 2.6.1, computed pixel values (10.8, 10.8), (12.6, 11.7) and (13.3, 10.4) are rounded to nearest integer values (11, 11), (13, 12) and (13, 10).

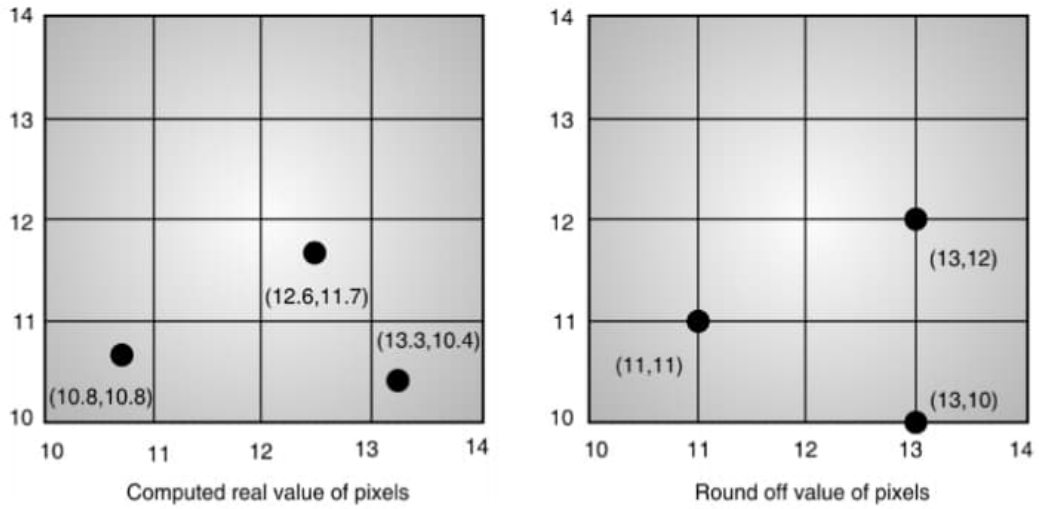


Fig. 2.6.1 : Conversion from actual value to rounded value

2.7 Frame Buffer

W-18

Q.	Define Frame Buffer.	**(W-18, 2 Marks)**
Q.	Describe Frame buffer display in computer graphics.	

- We can consider the screen picture as a two-dimensional grid of pixels. Raster system displays the picture by drawing pixel in a row by row from left to right. Picture definition is stored in a memory called the **refresh buffer** or **frame buffer**.
- Scene to be displayed is first loaded into a frame buffer in the form of intensity values. Ideally, the size of the frame buffer is the same as screen resolution. Intensity value from the top left location of the frame buffer is retrieved and painted at a top-left location on the screen. The second pixel of the same row is painted soon after that and this process continues.
- As shown in Fig. 2.7.1, frame buffer is part of display processor memory, which is connected with video controller.

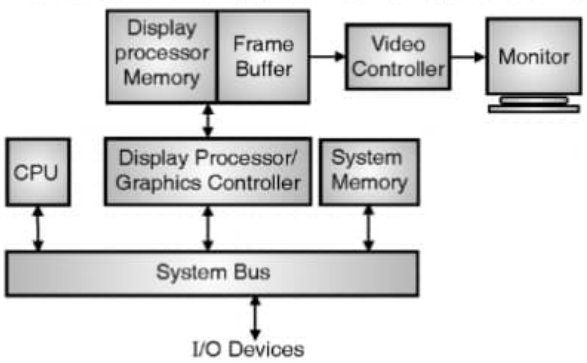


Fig. 2.7.1 : Position of frame buffer in raster scan display system

- Video controller reads the intensity value from frame buffer and gives command to CRT to paint it on monitor screen.
- Frame buffer in raster display stores individual pixels of the scene, whereas in case of random scan display, frame buffers stores command for drawing the scene.

- Suppose a system has resolution 1280 × 1280 and it supports 24 bits per pixel. The memory requirement for the frame buffer is :

$$\begin{aligned}
\text{Number of pixel} &= 1280 \times 1280 = 16, 38, 400 \\
\text{Number of bits} &= \text{Number of pixels} * \text{Number of bits/pixel} \\
&= 16, 38, 400 * 24 \\
&= 3, 93, 21, 600 \text{ bits} \\
&= 3, 93, 21, 600/8 = 49,15,200 \text{ Bytes} \quad (\because 1 \text{ Byte} = 8 \text{ bits}) \\
&= 49,15,200/1024 = 4800 \text{ KB} \quad (\because 1 \text{ KB} = 1024 \text{ Bytes}) \\
&= 4800/1024 = 4.69 \text{ MB} \quad (\because 1 \text{ MB} = 1024 \text{ KB})
\end{aligned}$$

- Thus, the system having screen resolution1280 × 1280 and supporting 24 bits per pixel require 4.69 MB of memory for the Frame buffer.
- The frame buffer is known as **bitmap** if it uses one bit per pixel. And it is known as **pixmap** if it uses multiple bits per pixel. A bitmap is used in the bi-level system while pixmap is used in a system supporting multiple colors.
- Sometimes the additional alpha channel is retained to adjust the transparency of a pixel.

2.8 Character Generation Methods

W-18

Q. Explain various character generation methods with suitable diagrams.

Q. Explain stroke method and Bitmap method with an example. **(W-18, 4 Marks)**

- Character generation is possible through hardware and software. Hardware can produce characters quickly but it is less flexible due to hardware limitations. Software generation is a bit slower but provide great flexibility. There are basically three methods of character generation.
  - Stroke or vector method
  - Bitmap or dot matrix method
  - Starbust method

2.8.1 Stroke Method

Q. Explain the stroke method for character generation.

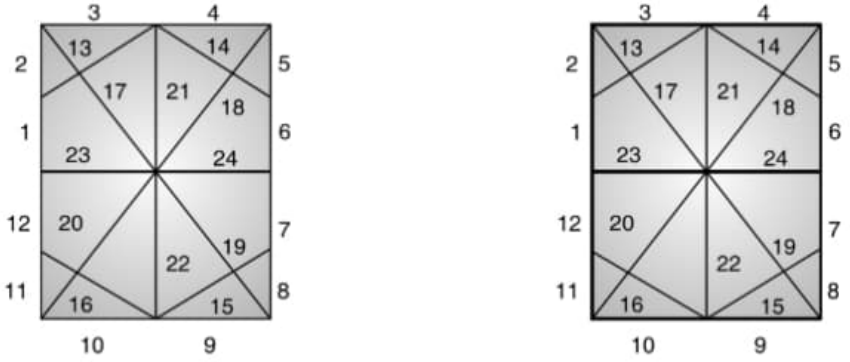
- This is also known as *vector character generation* method.
- In this method, basic primitive shapes like line, curves are used to generate the characters.
- Arguments to the character generation procedure decide the scale, weight and orientation of characters.
- Character is generally stored as a sequence of commands with its start and end points. We can even change the size of the characters.
- This method is device dependent.
- Characters generated using this method scales well. According to the size of character, algorithm scales the line, curve etc. primitives.

Fig. 2.8.1 : Character representation using stroke method

2.8.2 Starburst Method

Q. Explain the starbust method for character generation.

- Character is generated using fix line patterns.
- Normally 24 lines are sufficient to generate all characters. Out of 24 lines, only required lines are illuminated.
- These 24 lines are encoded using 24 bits to show which lines should be illuminated.
- Due to its characteristic appearance, this method is called star bust method. The particular bit is set to 1 or 0 to decide the visibility of that line segment.



(a) Star bust pattern to encode characters

(b) Encoding patter for E : (MSB indicates 24th bit)110000000000111100001111

Fig. 2.8.2

Encoding for character E is shown in the following table :

Bit position	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bit value	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1

Disadvantages :

- Each character is encoded using 24 bit so it needs more memory.
- Software interpretation of 24 bit is required for each character.
- Renders curvy shaped characters poorly due to straight line support only.

2.8.3 Bitmap Method

Q. Explain the working principle of bitmap character generation method.

- This method is also known as the **dot matrix method.**

- Characters are stored in two-dimensional array of dots. That's why this method is called dot matrix or bitmap method.
- Depending on the size of the font, array size varies. Most preferred sizes of the array are 7×5 and 9 × 7, where each character is stored in said size of the array.
- It is easy to retrieve and display character using this method. This method is faster but it's not much scalable due to fixed array size.
- Rescaling the font may create aliasing effect. However, such representation needs more memory because each variation must be stored as rescaling is not a good option to resize font.

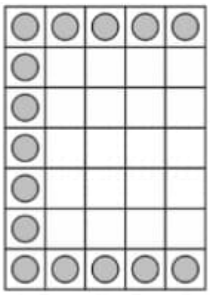


Fig. 2.8.3 : Character representation using the bitmap method

2.9 Lab Programs

Program 2.9.1 : Write a program to draw following graphics object using built in C functions :

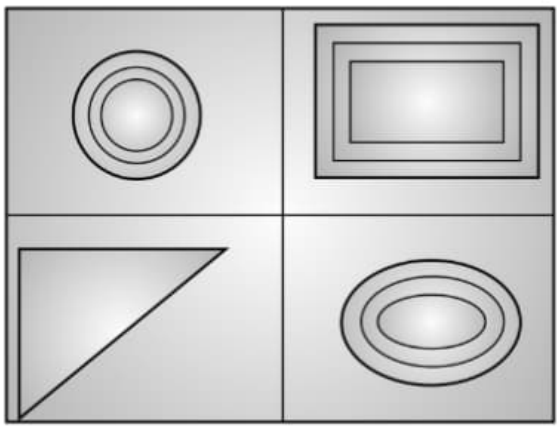
1. Pixel, 2. Lines, 3. Circles 4. Rectangles 5. Ellipse **(Lab 1)**

Solution :

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void main()
{
    clrscr();
    // Request to auto dection of graphics driver
    Int gdriver = DETECT;
    Int gmode;

    // Initialize graphics mode
    initgraph(&gdriver, &gmode, "");

    // Get supported resolution of graphcis adapter (xmax, ymax)
    int xmax = getmaxx();
    int ymax = getmaxy();

    // Divide screen into 4 parts
    line(0, ymax/2, xmax, ymax/2);          // Horizontal line
    line(xmax/2, 0, xmax/2, ymax);          // Vertical line

    // Draw pixel in center of the screen
    putpixel(xmax/2, ymax/2, RED);
```

```
    // Draw lines in first cell
    setcolor(RED);
    line(20, 10, 20, 200);
    setcolor(YELLOW);
    line(30, 200, 250, 200);
    setcolor(GREEN);
    line(25, 25, 240, 190);

    // Draw Ellpise in second cell
    setcolor(6);
    ellipse(480, 120, 0, 360, 100, 70);
    setcolor(7);
    ellipse(480, 120, 0, 360, 80, 50);
    setcolor(10);
    ellipse(480, 120, 0, 360, 60, 30);

    // Draw circle in third cell
    setcolor(15);
    circle(150, 350, 50);
    setcolor(14);
    circle(150, 350, 40);
    setcolor(13);
    circle(150, 350, 30);
    // Draw rectanle in forurth cell
    setcolor(2);
    rectangle(350, 280, 600, 450);
    setcolor(3);
    rectangle(370, 300, 580, 430);
    setcolor(4);
    rectangle(390, 320, 560, 410);

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

Output :



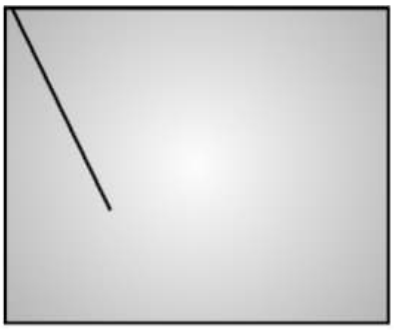
Program 2.9.2 : Implement DDA algorithm to draw a line. **(Lab 2)**

Solution :

```
// Assumption: Line is in first quadrant, growing from left to right and m < 1
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>

void main()
{
    clrscr();
    int gdriver = DETECT, gmode;
    int x1 = 10, x2 = 100, y1 = 10, y2 = 200;
    int dx, dy, x;
    float m, y;

    initgraph(&gdriver, &gmode, "");
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    m = float(dy) / (float)dx;

    x = x1;
    y = y1;
    putpixel(x, floor(y), 15);

    if(x1 > x2)
```

```
    {
        printf("Line is not growing from left to right");
    }
    else if(y1 > y2)
    {
        printf("Slope is greater than 1");
    }
    else
    {
        for(x = x1; x <= x2; x++)
        {
            y = y + m;
            putpixel(x, floor(y), 15);
        }
    }
    getch();
    closegraph();
}
```

Output :



Program 2.9.3 : Implement Bresenham's algorithm to draw a line. **(Lab 3)**

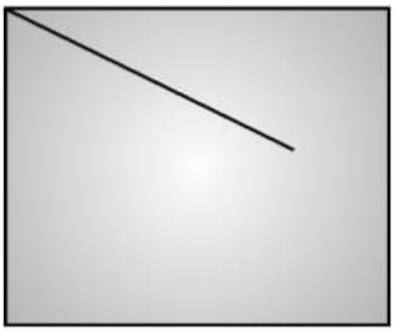
Solution :

```
// Assumption: Line is in first quadrant, growing from left to right and m < 1
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>

void main()
{
```

```
clrscr();
int gdriver = DETECT, gmode;
int x1 = 10, x2 = 300, y1 = 10, y2 = 150;
int dx, dy, x, y, k, p;
float m;

initgraph(&gdriver, &gmode, "");
dx = abs(x2 - x1);
dy = abs(y2 - y1);
m = float(dy) / (float)dx;

x = x1;
y = y1;

if(x1 > x2)
{
    printf("Line is not growing from left to right");
}
else if(y1 > y2)
{
    printf("Slope is greater than 1");
}
else
{
    p = 2*dy - dx;
    for(k = 0; k < dx; k++)
    {
        if(p < 0)
        {
            x = x + 1;
            putpixel(x, y,15);
            p = p + 2*dy;

        }
        else
        {
            x = x + 1;
            y = y + 1;
```

```
                putpixel(x, y, 15);
                p = p + 2*(dy- dx);
            }
        }
    }
    getch();
    closegraph();
}
```

Output :



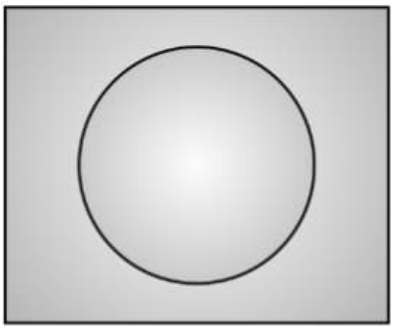
Program 2.9.4 : Implement Bresenham's algorithm to draw a circle. **(Lab 4)**

Solution :

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

int xc, yc;
void EightWaySymmetry(int, int);

void main()
{
    clrscr();
    int gdriver = DETECT, gmode;
    int r = 100, S, x, y;

    initgraph(&gdriver, &gmode, "");

    int xmax = getmaxx();
    int ymax = getmaxy();
```

```
    xc = floor(xmax/2);
    yc = floor(ymax/2);

    x = 0;
    y = r;
    EightWaySymmetry(x, y);
    S = 3 - 2*r;
    while(x <= y)
    {
        x = x + 1;
        if (S < 0)
        {
            S = S + (4*x + 6);
        }
        else
        {
            S = S + (4*(x - y) + 10);
            y = y - 1;
        }
        EightWaySymmetry(x, y);
    }

    getch();
    closegraph();
}
void EightWaySymmetry(int x, int y)
{
    int c = 10;
    putpixel(xc + x, yc + y, c);
    putpixel(xc + x, yc - y, c);
    putpixel(xc - x, yc + y, c);
    putpixel(xc - x, yc - y, c);
    putpixel(xc + y, yc + x, c);
    putpixel(xc + y, yc - x, c);
    putpixel(xc - y, yc + x, c);
    putpixel(xc - y, yc - x, c);
}
```

Output :



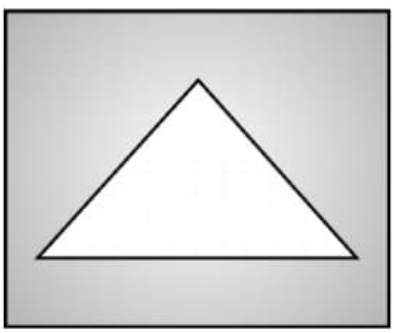
Program 2.9.5 : Write a program to fill polygon using Flood fill. **(Lab 5)**

Solution :

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

void FloodFill(int, int, int, int);
void main()
{
    clrscr();
    int gdriver = DETECT, gmode;
    int fillColor = 4, oldColor = 0, r = 25;
    initgraph(&gdriver, &gmode, "");
    int xmax = getmaxx();
    int ymax = getmaxy();
    int midx = floor(xmax/2);
    int midy = floor(ymax/2);
    circle(midx, midy, r);
    FloodFill(midx, midy, fillColor, oldColor);
    getch();
    closegraph();
}
void FloodFill(int x, int y, intfillColor, intoldColor)
{
    int current;
    current = getpixel(x, y);
    if(current == oldColor)
    {
        putpixel(x, y, fillColor);
```

```
        delay(3);
        FloodFill(x, y + 1, fillColor, oldColor);
        FloodFill(x, y - 1, fillColor, oldColor);
        FloodFill(x + 1, y, fillColor, oldColor);
        FloodFill(x - 1, y, fillColor, oldColor);
    }
}
```

Output :



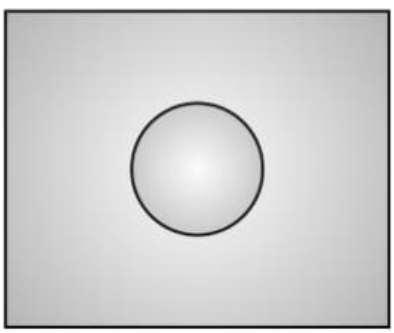
Program 2.9.6 : Write a program to fill polygon using Boundary fill. **(Lab 6)**

Solution :

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

void BoundaryFill(int, int, int, int);
void main()
{
    clrscr();
    int gdriver = DETECT, gmode;
    int fillColor = 4, boundaryColor = 15, r = 30;
    initgraph(&gdriver, &gmode, "");

    int xmax = getmaxx();
    int ymax = getmaxy();
    int midx = floor(xmax/2);
    int midy = floor(ymax/2);

    circle(midx, midy, r);
```

```
    BoundaryFill(midx, midy, fillColor, boundaryColor);

    getch();
    closegraph();
}
void BoundaryFill(int x, int y, int fillColor, int boundaryColor)
{
    int current;
    current = getpixel(x, y);

    if(current != fillColor&& current != boundaryColor)
    {
        putpixel(x, y, fillColor);
        delay(5);
        BoundaryFill(x, y + 1, fillColor, boundaryColor);
        BoundaryFill(x, y - 1, fillColor, boundaryColor);
        BoundaryFill(x + 1, y, fillColor, boundaryColor);
        BoundaryFill(x - 1, y, fillColor, boundaryColor);
    }
}
```

Output :



Review Questions

Topic : Basics Concepts in line drawing

Q. 1 Define following terms : Point, Rasterization.

Q. 2 How to draw a line using slope-intercept formula?

Q. 3 State the characteristics of ideal line.

Q. 4 Enlist three methods of line representation.

Q. 5 What do you mean by scan conversion of line?

Topic : Digital Differential Analyzer (DDA) Line Drawing Algorithm

Q. 1 Explain DDA line drawing algorithm.

Q. 2 Enlist merits and demerits of DDA line drawing algorithm.

Q. 3 Mention the steps to draw line with |m| > 1 using DDA line drawing algorithm.

Q. 4 Write an algorithm for DDA line drawing algorithm.

Q. 5 Calculate the pixel coordinates of line AB using DDA algorithm. Where A = (0, 0) and B = (5, 8).

Q. 6 Draw a line from A(12, 3) to B(7, 9) using DDA line drawing algorithm.

Q. 7 Write C code to scan convert the line using DDA algorithm.

Topic : Bresenham's Line Drawing Algorithm

Q. 1 Explain Bresenham's line drawing algorithm. Derive decision parameters.

Q. 2 Mention the steps to draw line using Bresenham's method.

Q. 3 State advantages and disadvantages of Bresenham's line drawing approach.

Q. 4 Write a function Bresenhamline (x1, y1, x2, y2, type) so it draws line with specified type. If type = 0, solid line, type=1 dashed line and type=2 dotted line.

Q. 5 Calculate the pixel coordinates of line PQ using Bresenham's algorithm where, P = (20, 20) and Q = (10, 12)

Q. 6 Write an algorithm for Bresenham's line drawing algorithm.

Q. 7 Differentiate: DDA vs. Bresenham's line drawing algorithm.

Q. 8 Write C code to scan convert the line using Bresenham's algorithm.

Topic : Circle Generation Algorithms

Q. 1 Write a note on 8-way symmetry of circle.

Q. 2 What is the limitation of incremental circle drawing algorithm? How to overcome it using polar circle drawing method?

Q. 3 How to draw a circle using polar equation?

Q. 4 State pros and cons of polar circle generation method.

Q. 5 Write an algorithm to scan converts the circle using polar equation.

Q. 6 Derive equations of decision parameter for Bresenham's circle drawing algorithm

Q. 7 Write the steps for Bresenham's circle generation algorithm

Q. 8 Write an algorithm for Bresenham's circle drawing algorithm

Q. 9 Plot a circle using Bresenham's algorithm for r = 5 and center at (0, 0)

Q. 10 Write a program to scan convert the circle using incremental approach

Q. 11 Write a program to scan convert the circle using polar representation

Q. 12 Write a program to scan converts the circle Bresenham's circle generation algorithm.

Topic : Polygon Filling Algorithms

Q. 1 What is polygon? Explain various types of polygon.

Q. 2 Explain inside-outside test to check the membership of pixel with respect to given polygon

Q. 3 Describe odd-even method with suitable example

Q. 4 Explain winding number rule method.

Q. 5 Describe 4-connectivity and 8-connectivity of the pixel.

Q. 6 Explain boundary fill algorithm.

Q. 7 Write an algorithm for boundary fill.

Q. 8 Write C code for boundary fill algorithm.

Q. 9 Explain flood fill algorithm.

Q. 10 Write an algorithm for flood fill.

Q. 11 Write C code for flood fill algorithm.

Q. 12 State the advantages and disadvantages of seed fill algorithms.

Q. 13 How to optimize the filling method for flood fill and boundary fill?

Q. 14 Explain scan line polygon filling algorithm.

Q. 15 Differentiate: Seed fill vs. Scan line algorithm.

Topic : Scan Conversion and Frame Buffer

Q. 1 What do you mean by scan conversion?

Q. 2 Describe the role of frame buffer in computer graphics.

Q. 3 Define terms: Frame buffer, Bitmap, Pixmap.

Q. 4 Find the size of frame buffer for a system having resolution 3000 x 4000 and it supports 8 bits per pixel.

Topic : Character Generation

Q. 1 Explain various character generation methods.

Q. 2 Explain stroke method. Write its good characteristics.

Q. 3 Discuss starbust method with its limitations.

Q. 4 Write a note on bitmap method and state its advantage and limitation.

Chapter Ends...

❑❑❑

2D Transformation

Syllabus

Two Dimensional Transformations : Translation, Scaling, Rotation, Reflection, Shearing, Matrix Representations and Homogeneous Coordinates : Translation, Scaling, Rotation, Reflection, Shearing, Composite Transformations : Rotation About and Arbitrary Point

3.1 Introduction

W-18

Q. What is a transformation ?

Q. List out basic transformations techniques. **(W -18, 1 Mark)**

- In the real world, the user frequently rearranges objects. A graphics system should allow users to change the way by which the object appears.
- The process of changing the scale, shape or position of the object is called **transformation**.
- Transformation is a very common operation used in graphics applications.
- Translation, rotation and scaling are the basic transformation operations.
- Reflection and shearing are other common transformation operations.
- Translation, rotation and reflection are also known as **rigid body transformations** because they do not alter the shape of the object. They just reposition the object from one location to another location.
- Scaling and shearing deform the shape of the object under consideration, so they are not rigid body transformation.
- The main objective of two-dimensional transformations is to simulate movement and manipulate 2D objects in the XY plane. This can be achieved in two ways.

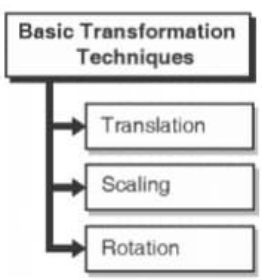


Fig. 3.1.1 : Basic transformation operations

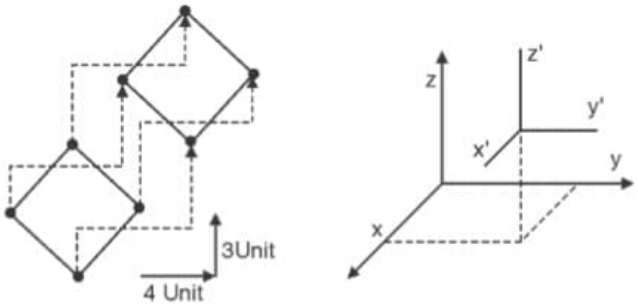
3.1.1 Geometric Transformation

Q. Explain the geometric and coordinate transformation.

- Geometric transformation is a common operation in almost all graphics packages.
- In which, the object itself is rotated, scaled or moved to get the desired effect.
- Viewing coordinates are stationary. Fig. 3.1.2(a) shows the displacement of the original object by translation vector (4, 3).

3.1.2 Coordinate Transformation

- In coordinate transformation, the object remains stationary and the coordinate system is changed to get the desired effect.
- Fig. 3.1.2(b) shows the displacement of the original coordinate system (x, y, z) to (x′, y′, z′).
- To view all the surfaces of the book, we rotate and translate the book itself. But, if we want to see different sides of the building, coordinate system (our eye) has to be transformed, the building remains stationary.



(a) Geometric Transformation **(b) Coordinate Transformation**

Fig. 3.1.2 : Types of transformation

Terminology :

The following terminology will be used for the rest of the discussion.

- P = Set of points of the object which has to be transformed.
- P′ = Set of points of the object which has been transformed.
- M = Transformation matrix.

Transformation is a function of object coordinates with various operations as another argument.

$$\therefore P' = f(P, M)$$

Where, M is a transformation matrix to be applied on a set of points P. More common representation of transformation operation is $P' = M \cdot P$.

3.2 Matrix Operations

Transformation and matrix operations go hand in hand. Let us first discuss various representation, property and operations on matrix.

3.2.1 Representation of Matrix

The matrix can be represented in two ways.

3.2.1.1 Column Measure Representation

In this representation, point coordinates are represented in the column.

For example,

$$A = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\therefore \quad x' = ax + cy$$

$$y' = bx + dy$$

In column major representation, during transformation, M is pre-multiplied to original points of an object.

3.2.1.2 Row Measure Representation

In this representation, point coordinates are represented in row. For example,

$$B = A^T = [x \ y] = [2 \ 3]$$

$$P'^T = P^T \cdot M^T$$

$$[x' \ y'] = [x \ y] \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\therefore \quad x' = ax + cy$$

$$y' = bx + dy$$

In row major representation, during transformation, M is *post-multiplied* to original points of an object.

We can multiply two matrices A and B only if dimensions of A are p × q and dimensions of B are q × r. For matrix multiplication, it is necessary to have a number of columns in the first matrix equivalent to the number of rows in the second matrix.

$$A_{p \times q} \times B_{q \times r} = C_{p \times r}$$

If number of columns in A and number of rows in B are not same, then matrix dimensions are not compatible and multiplication is not possible.

3.2.2 Matrix Properties

1. **Associative :** Matrix multiplication is associative. i.e.,

$$A.B.C = A \,.\, (B \,.\, C) = (A \,.\, B) \,.C$$

2. **Cumulative :** Matrix multiplication is not cumulative. i.e.,

$$A.B \neq B.A$$

3.2.3 Determinant of Matrix

Determinant of 2 × 2 matrix is calculated as below :

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$\text{Determinant } \Delta(A) = |A| = a_{11}a_{22} - a_{12}a_{21}$$

Determinant of 3 × 3 matrix is calculated as below :

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$\text{Determinant } \Delta(B) = |B|$$

$$= b_{11}(b_{22}b_{33} - b_{23}b_{32}) - b_{12}(b_{21}b_{33} - b_{23}b_{31}) + b_{13}(b_{21}b_{32} - b_{22}b_{31})$$

3.2.4 Multiplying Two Matrices

Let us assume that A and B are 2 × 2 square matrices to be multiplied and C is the resultant matrix.

If, $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

$$C = A \times B$$

$$\therefore \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Where,

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

For some random dimensional compatible, rectangular matrices A and B,

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

Where,

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13}B_{31}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} + A_{13}B_{32}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} + A_{23}B_{31}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} + A_{23}B_{32}$$

3.3 Translation

Q. Explain 2D Transformation. Derive transformation matrix for the same.

Translation is defined as a shifting of an object along straight path.

Translation does not alter the shape or size of the object. It just moves the entire object from one location to other location along a straight path.

This is achieved by first shifting object in the X direction and then in the Y direction by amount t_x and t_y, respectively.

The vector $T = [t_x \quad t_y]$, specifying the amount of shift in both the direction is called **shift vector** or **translation vector**.

Translation operation is formulated as,

$$x' = x + t_x$$
$$y' = y + t_y$$

Where (x, y) is the original point, $[t_x, t_y]$ is shift vector and (x', y') is translated point.

Equivalent matrix representation of this operation is,

$$P' = T + P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

If an object has more than one vertex, we can apply a transformation to individual vertex and then join them all to reconstruct the object.

Rigid body transformation moves object without deformation. Amount of shift is identical for all the points.

In Fig. 3.3.1, object O is translated by shift vector [6, 5]. O' is the translated object.

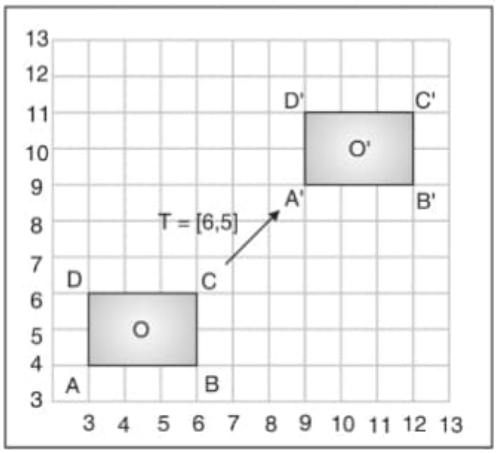


Fig. 3.3.1 : Translation operation

Example 3.3.1 : Translate a triangle with coordinates A(2,2), B(6,2) and C(4,4) with translation vector $[4\ \ 7]^T$.

Solution :

$$\text{Here translation vector, } T = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

Translation operation is defined as,

$$P' = T + P$$

$$\therefore A' = T + A = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

$$B' = T + B = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$$

$$C' = T + C = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 8 \\ 11 \end{bmatrix}$$

After translation, triangle A(2,2), B(6,2), C(4,4) will move to A' (6,9), B'(10,9), C'(8,11).

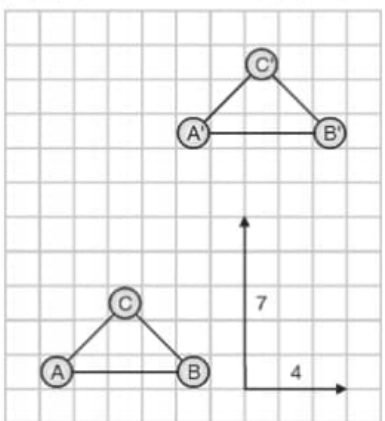


Fig. P. 3.3.1 : Translation of triangle ABC by shift vector $[4\ \ 7]^T$

Original Coordinates	Transformed Coordinates
A(2, 2)	A'(6, 9)
B(6, 2)	B'(10, 9)
C(4, 4)	C'(8, 11)

3.4 Scaling

W-18

Q. Explain scaling transformation with respect to 2D. **(W-18, 3 Marks)**

Q. What is scaling ? Derive 2D transformation matrix for scaling with respect to origin and a reference point

- Translation and rotation change the position of an object; they do not deform the shape. Scaling may alter the shape of an object.
- Scaling can be performed with respect to origin or some reference point.

3.4.1 Scaling with respect to Origin

Q. Derive the transformation matrix for scaling with respect to origin.

Scaling is carried out by multiplying (x, y) coordinates with corresponding scaling parameters S_x and S_y respectively. So,

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

We can represent the scaling operation in matrix form as,

$$P' = S.\ P$$

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

S_x, S_y are any positive numeric values.

Note : for 2 x 2 matrix representation, translation is an additive operation, whereas scaling and rotation are multiplicative.

Various cases of scaling :

$S_x = S_y = 1$ $\Rightarrow$ No scaling

$S_x = S_y \neq 1$ $\Rightarrow$ Uniform scaling

$S_x \neq S_y$ $\Rightarrow$ Non Uniform scaling

$S_x = S_y < 1$ $\Rightarrow$ Uniform compression

$S_x = S_y > 1$ $\Rightarrow$ Uniform explanation

$S_x > 1$ $\Rightarrow$ X coordinates of the object will move away from the origin

$S_x < 1$ $\Rightarrow$ X coordinates of the object will move near to the origin

$S_y > 1$ $\Rightarrow$ Y coordinates of the object will move away from the origin

$S_y < 1$ $\Rightarrow$ Y coordinates of the object will move near to the origin

– As shown in Fig. 3.4.1, line AB has coordinates A(1, 1) and B(1, 4). With $S_x = 3$, $S_y = 2$, the line goes away from the origin and we get the A'(3, 2) and B'(3, 8).

$$A' = S \cdot A = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$B' = S \cdot B = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

– In a similar way, if we select scaling factor less than one, object comes closer to the origin. With $S_x = S_y = 0.5$, line CD with endpoints C(10, 4) and D(10, 8) comes at C'(5, 2) and D'(5, 4).

$$C' = S \cdot C = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 10 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$D' = S \cdot D = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

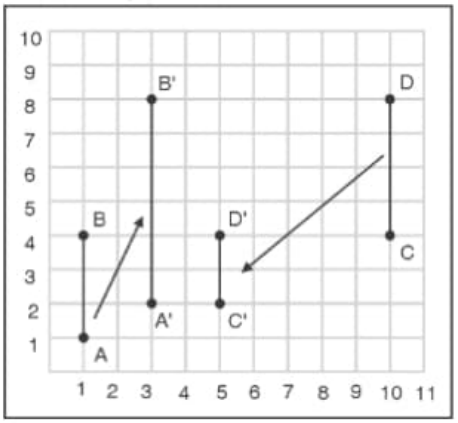


Fig. 3.4.1 : Demonstration of scaling

Note : These scaling formulas are only applicable if scaling is with respect to the origin.

3.4.2 Scaling with respect to Reference Point

Q. Derive the transformation matrix for scaling with respect to reference point.

In this section, we will derive the transformation matrix to scale the object with respect to some reference point

Let (x_r, y_r) be the reference point, with respect to which object is to be scaled.

Following steps should be performed to carry out the desired operation.

1. Translate the reference point to the origin

$$T = \begin{bmatrix} -x_r \\ -y_r \end{bmatrix}$$

2. Apply scaling on translated object

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

3. Translate reference point back to its actual location

$$T^{-1} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

The sequence of operations is shown in Fig. 3.4.2.

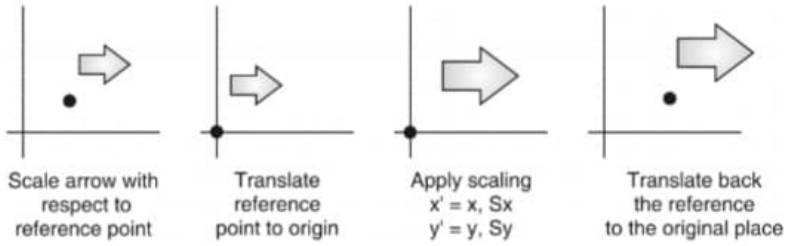


Fig. 3.4.2 : Sequence of transformation operations for scaling with respect to a reference point

So, the transformed coordinates of an object are given by,

$$P' = [T^{-1} + [S[T + P]]]$$

$$P' = \left\{\begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}\right\} \cdot \left\{\begin{bmatrix} -x_r \\ -y_r \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}\right\}\right\}$$

$$= \left\{\begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix}\right\}\right\} = \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \begin{bmatrix} (x - x_r) S_x \\ (y - y_r) S_y \end{bmatrix}$$

$$x' = x_r + (x - x_r)S_x = S_x \cdot x + x_r(1 - S_x)$$

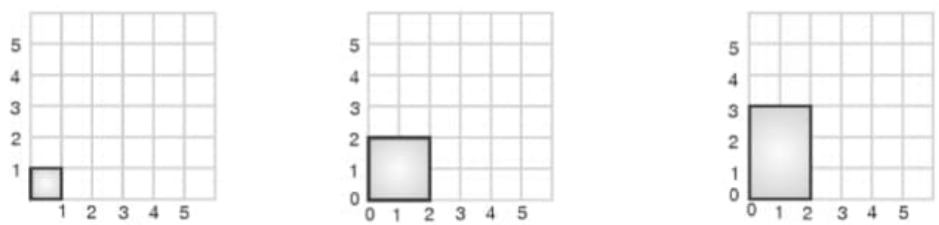
$$y' = y_r + (y - y_r)S_y = S_y \cdot y + y_r(1 - S_y)$$

Where, $x_r (1 - S_x)$ and $y_r (1 - S_y)$ are constant for all points

3.4.3 Uniform vs. Non Uniform Scaling

Q. Explain uniform and non-uniform scaling with suitable example.

- If scaling parameters are same for both the directions, it is called **uniform scaling**, otherwise it is called **non uniform scaling**.
- Uniform scaling only scale up or scale down the object size; it does not change the shape of the object. In the case of uniform scaling, square remains square and circle remains circle after scaling operation, only size of object alters.
- Non-uniform scaling may alter the shape of the object. In the case on non-uniform scaling, the circle may turn out to an ellipse, or square may turn out to a rectangle.
- Consider the unit square with vertices A(0, 0), B(1, 0), C(1, 1) and D(0, 1). If we apply uniform scaling on this unit square with $s_x = s_y = 2$, we get A'(0, 0), B'(2, 0), C'(2, 2) and D'(0, 2). And if we apply non uniform scaling with $S_x = 2$ and $S_y = 3$, we get A"(0, 0), B"(2, 0), C"(2, 3) and D"(0, 3). Fig. 3.4.3 illustrates the concept of uniform and non-uniform scaling.
- In case of uniform scaling, square remains square, whereas in case of non-uniform scaling square turns into triangle.



(a) Unit square **(b) Uniform scaling with $S_x = S_y = 2$** **(c) Non uniform scaling with $S_x = 2$, $S_y = 3$**

Fig. 3.4.3 : Uniform and non-uniform scaling

Example 3.4.1 : Scale a square with coordinates A(2,2) B(4,2) C(4,4) D(2,4), Where; $S_x = 3$ and $S_y = 2$.

Solution :

Transformation sequence for scaling is defined by,

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

$$\therefore A' = S.A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$B' = S.B = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 4 \end{bmatrix}$$

$$C' = S.C = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix}$$

$$D' = S.D = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

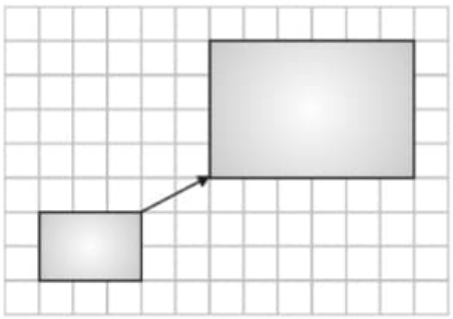


Fig. P. 3.4.1 : Scaling of object by scaling parameters $S = [3 \quad 2]^T$

Original Coordinates	Transformed Coordinates
A(2, 2)	A'(6, 4)
B(4, 2)	B'(12, 4)
C(4, 4)	C'(12, 8)
D(2, 4)	D'(6, 8)

Example 3.4.2 : Scale a triangle with vertices A(2,2), B(6,2) and C(4,4) with respect to a reference point (3,4) with $S_x = 2$ and $S_y = 3$.

Solution :

Here, Scaling is with respect to a reference point. We have to first translate reference point to origin then apply to scale and then translate reference point back to the actual location. Transformed coordinates of this sequence of operation is given by,

$$x' = S_x x + x_r(1 - S_x)$$

$$y' = S_y \cdot y + y_r(1 - S_y)$$

$x_r(1 - S_x)$ and $y_r(1 - S_y)$ are constant for all points.

$$x_r(1 - S_x) = 3(1 - 2) = -3$$

$$y_r(1 - S_y) = 4(1 - 3) = -8$$

$$A' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$B' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 12 \\ 6 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 9 \\ -2 \end{bmatrix}$$

$$C' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(2, 2)	A'(1, – 2)
B(6, 2)	B'(9, – 2)
C(4, 4)	C'(5, 4)

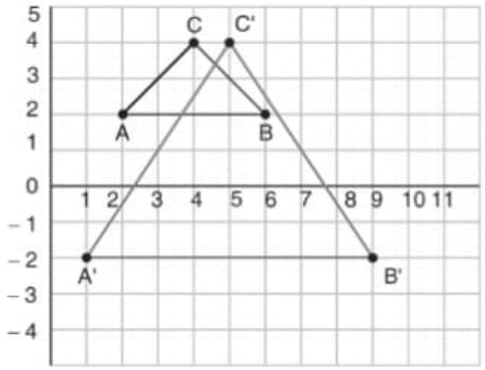


Fig. P. 3.4.2 : Output of scaling operation

Example 3.4.3 : Apply scaling to unit square with a bottom left corner at (3, 4) with $S_x = 2$ and $S_y = 0.5$.

Solution :

Assume that A is the corner with coordinates (3, 4). So remaining vertices of the unit square will be B (4, 4), C (4, 5) and D (3,5).

$$\text{Scaling matrix, } S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Let us assume P and P' are original and transformed coordinates of an object respectively.

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

$$= \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 3 & 4 & 4 & 3 \\ 4 & 4 & 5 & 5 \end{bmatrix} = \begin{bmatrix} 6 & 8 & 8 & 6 \\ 2 & 2 & 2.5 & 2.5 \end{bmatrix}$$

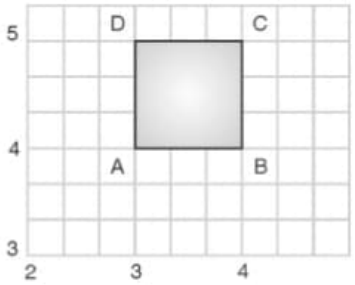


Fig. P. 3.4.3

Original Coordinate	Transformed Coordinate
A(3, 4)	A'(6, 2)
B(4, 4)	B'(8, 2)
C(4, 5)	C'(8, 2.5)
D(3, 5)	D'(6, 2.5)

3.5 Rotation

Q. Derive the transformation matrices for rotation with respect to origin and reference point.

- 2D **rotation** is described by repositioning all the points of an object *along a circular path* in the 2D plane.
- To perform rotation, we need a rotation angle θ and the reference point (x_r, y_r) with respect to which object is to be rotated. The reference point is also called as pivot point.
- If rotation is performed in an anti-clockwise direction, the value of the angle is considered positive, otherwise, it is negative. Rotation directions are shown in Fig. 3.5.1.

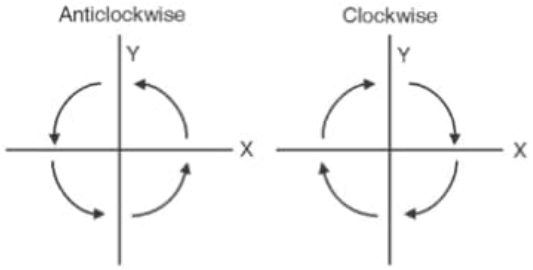


Fig. 3.5.1 : Anticlockwise and clockwise rotation direction

– Rotation can be performed with respect to origin or with respect to some reference point. We will discuss both the cases here.

3.5.1 Rotation with respect to Origin

Q. Derive 2D transformation matrix for rotation with respect to origin.

– Let's derive the transformation matrix for *rotation about the origin*. As shown in Fig. 3.5.2, P(x, y) is the original point, which is to be rotated in XY plane by the angle θ in an anticlockwise direction. P'(x', y') is the rotated point.

– P is rotated with respect to the origin, and distance of P from the origin is let's say *r*.

– Let us, consider the angle of P with X axis is ϕand is rotated by angle θ in anti-clockwise direction, so point P' makes an angle (ϕ + θ) with X-axis.

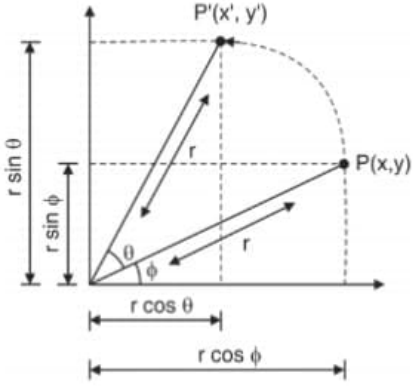


Fig. 3.5.2 : Rotational geometry of point P

From Fig. 3.5.2, Using trigonometric rule, for Point P,

$$\sin\phi = \frac{y}{r}$$

$$\cos\phi = \frac{x}{r}$$

$$\therefore\ x = r\cos\phi \qquad \ldots(3.5.1)$$

$$y = r\sin\phi \qquad \ldots(3.5.2)$$

For point P',

$$\cos(\phi+\theta) = \frac{x'}{r}$$

$$\sin(\phi+\theta) = \frac{y'}{r}$$

$$\text{So, } x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta \qquad \ldots(3.5.3)$$

$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\sin\theta \qquad \ldots(3.5.4)$$

By putting value of Equations (3.5.1) and (3.5.2) in Equation (3.5.3) and Equation (3.5.4), we get

$$x' = x\cos\theta - y\sin\theta \qquad \ldots(3.5.5)$$

$$y' = x\sin\theta + y\cos\theta \qquad \ldots(3.5.6)$$

Matrix representation of above equation is written as:

$$P' = R.P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \qquad \ldots(3.5.7)$$

Where, $R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ represents rotation matrix for rotation about origin

Inverse rotation is defined by considering negative value of θ

$$R^{-1} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

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

$$= R^{T}$$

Inverse rotation matrix is equal to transpose of the original matrix. Such a matrix is called the **orthogonal matrix**.

Properties of pure rotation matrix :

- The determinant is always 1.
- The matrix should be orthogonal.

Note :

– If reference point is not specified, rotation is by default about the origin.

– If rotation direction is not specified, by default it is anti-clockwise.

3.5.2 Rotation with Respect to Reference Point

Q. Derive 2D transformation matrix for rotation with respect to reference point.

Matrix sequence of Equation (3.5.7) is only valid for rotation about the origin. If we want to rotate any point or object with respect to some reference point (x_r, y_r), then we should perform the following steps.

1. Translate the reference point (x_r, y_r) to origin : Translation vector is given as,

$$T = \begin{bmatrix} -x_r \\ -y_r \end{bmatrix}$$

2. Apply rotation by given angle θ. The rotation matrix is given as,

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

3. Translate reference point back to its actual location.

$$T^{-1} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

The sequence of operations is shown in Fig. 3.5.3.

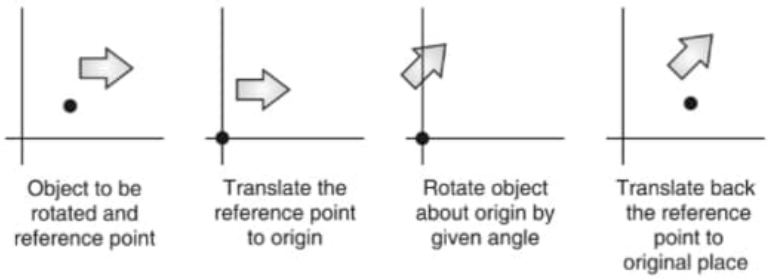


Fig. 3.5.3 : Transformation sequence for rotation with respect to reference point

Sequence of operations is defined as :

$$P' = [T^{-1} + [R[T + P]]]$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \left\{ \begin{bmatrix} -x_r \\ -y_r \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \right\} \right\} \right\}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix} \right\} \right\}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \begin{bmatrix} (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{bmatrix}$$

So,

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

Example 3.5.1 : Rotate a point P(3, 2) around the origin in counterclockwise direction by 90°.

Solution :

Here, the direction of rotation is anticlockwise

$$\text{So, } \theta = +90^\circ$$

$$\therefore P' = R.P = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

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

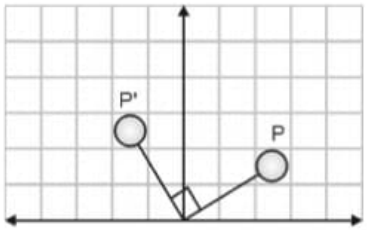


Fig. P. 3.5.1 : Rotation of point P(2, 3) by 90° in counterclockwise direction

Thus, point P(3, 2) becomes P'(– 2, 3) after rotation by 90°in anti-clock wise direction.

Original Coordinates	Transformed Coordinates
A(3, 2)	A' (– 2, 3)

Example 3.5.2 : Rotate a point A(3, 2) by 90° in anticlockwise direction with respect to reference point B(1, 2).

Solution :

Here, Rotation direction is anticlockwise, so $\theta = +90^\circ$

We have to rotate a point around the reference point

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\therefore x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

$$x' = 1 + (3 - 1)\cos 90^\circ - (2 - 2)\sin 90^\circ$$

$$= 1 + (2)\,0 - (0)1$$

$$\therefore x' = 1$$

$$y' = 2 + (3 - 1)\sin 90^\circ + (2 - 2)\cos 90^\circ = 2 + (2)\,1 + (0)\,0$$

$$\therefore y' = 4$$

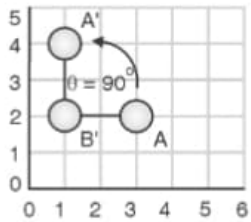


Fig. P. 3.5.2 : Output of said rotation operation

Thus, point A(3,2) becomes A'(1, 3) after rotation by 90°in anti-clockwise direction with respect to reference point B(1, 2).

Original Coordinates	Transformed Coordinates
A(3, 2)	A'(1, 4)

3.6 Matrix Representation and Homogeneous Coordinates

Q. Explain in brief : Homogeneous coordinate.

Q. What is the need of homogenous coordinate in transformation?

- In real-world applications, it is very common that the object goes through sequences of different geometric transformations.
- The object may require translation, rotation, scaling, shearing or reflection to be applied any number of times in any order.
- In previous sections, we saw that translation operation is additive while scaling and rotation are multiplicative. So all three basic transformations can be represented in generalized form as,

$$P' = M_1 + M_2P \quad \ldots(3.6.1)$$

- For translation, M_2 is the identity matrix (multiplicative identity).

  So, $P'=M_1 + P$ represents translation matrix

  And for rotation and scaling, M_1 is zero matrices (additive identity),

  So, $P' = M_2P$, Where M_2 represents rotation or scaling matrix
- We will reformulate this matrix representation of Equation (3.6.1) to combine additive and multiplicative matrices in a single matrix by extending 2 × 2 representation to 3 × 3 using homogeneous representation.
- In the homogeneous coordinate system, if we multiply the point by some constant, it represents the same point.
- Any Cartesian coordinate (x, y) can be represented as (x_h, y_h, h) in a homogeneous system, such that

$$x = \frac{x_h}{h} \text{ and } y = \frac{y_h}{h}$$

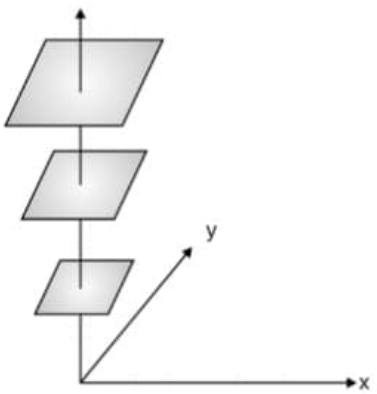


Fig. 3.6.1 : Representation of point in the homogeneous coordinate system

- Point (x, y) in Cartesian coordinate has infinite representation in homogeneous coordinates.
- For example, (x, y) = (x, y, 1) = (2x, 2y, 2) = ... = (x_h, y_h, h), all correspond to the same representation of (x, y).
- Homogeneous coordinate introduces the third dimension but it doesn't alter the value of (x, y).
- Fig. 3.6.1 explains how we can represent same points at different levels.

Properties :

Q. List the properties of the homogeneous coordinate representation.

1. Any 2D point in the homogeneous coordinate system is represented by a triplet (x, y, h), where x, y and h are not all zero. (0, 0, 0) does not represent any point. Origin is represented as (0, 0, 1).
2. In homogeneous coordinate systems, two points are identical, if one point is derived by multiplying some constant to the second point.
3. If h is not zero, then point (x_h, y_h, h) in a homogenous coordinate system is represented as (x_h/h, y_h/h) in the Euclidean/Cartesian coordinate system.
4. If h is 0, point represented is at infinity.

- Homogenous representation allows us to write all geometric transformation equations in matrix multiplication form.
- It brings uniformity in operation. Implementation of transformation operation in programming language becomes easier with this representation as all the operations are performed using matrix multiplication operations only.
- Basic transformation operations matrix using homogeneous coordinate are shown below.

3.6.1 Translation

Q. Give matrix representation for 2D translation.

Q. Give the homogeneous representation for translation, scaling and rotation.

Translation operation in homogenous system can be represented as follow :

$$P' = T.P$$

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

$$x' = x + t_x$$

$$y' = t + t_y$$

$$1 = 1$$

With homogeneous representation, translation operation is converted to multiplicative operation.

3.6.2 Scaling

W-18

Q. Give matrix representation for 2D scaling. **(W-18, 2 Marks)**

Scaling operation in homogenous system can be represented as follow :

$$P' = S.P$$

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

$$x' = x \cdot S_x$$

$$y' = y \cdot S_y$$

$$1 = 1$$

3.6.3 Rotation

Q. Give matrix representation for 2D rotation.

Rotation operation in homogenous system can be represented as follow :

$$P' = R.P$$

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

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

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

$$1 = 1$$

3.7 Composite Transformation

Q. What is composite transformation?

Composite transformation is any sequence of basic transformation operations.

We can generate a composite transformation matrix by multiplying matrices of basic transformation operations like translation, scaling, rotation, reflection, shearing etc.

For column measure representation matrices are placed from right to left.

Let's say $M_1, M_2, \ldots, M_n$ are the transformation matrices of some transformation operation in the same order, so for column measure representation of transformed coordinates is given as,

$$P' = M_n \ldots . M_2 . M_1 . P$$

Where, M_i indicates i^{th} operation in sequence.

For row measure representation, transformed coordinates are computed as follow :

$$P'^T = P^T . M_1^T . M_2^T \ldots \ldots . M_n^T$$

3.7.1 Successive Transformations

1. Translation :

Q. Prove that two successive translations are additive.

Let us translate the point(x, y) by $[t_{x_1}\ t_{y_1}]^T$ followed by $[t_{x_2}\ t_{y_2}]^T$. Here translation by $[t_{x_1}\ t_{y_1}]^T$ is the first operation, let's call the transformation matrix M_1, which is followed by translation vector $[t_{x_2}\ t_{y_2}]^T$, call it M_2.

So transformed coordinates are given as,

$$P' = M_2 \cdot M_1 \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x_2} \\ 0 & 1 & t_{y_2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x_1} \\ 0 & 1 & t_{y_1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x_1} + t_{x_2} \\ 0 & 1 & t_{y_1} + t_{y_2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Thus, two successive transformations are additive.

2. Rotation :

Q. Prove that two successive rotations are additive.

Let us rotate the point (x, y) by the angle θ_1 followed by angle θ_2. Here rotation by angle θ_1 is the first operation, let's call the transformation matrix M_1, which is followed by rotation with θ_2 angle, call the transformation matrix for this rotation M_2. So transformed coordinates P' are given as,

$$P' = M_2 \cdot M_1 \cdot P$$

$$= R(\theta_2) \cdot R(\theta_1) \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) & -\sin(\theta_1)\cos(\theta_2) - \sin(\theta_2)\cos(\theta_1) & 0 \\ \sin(\theta_1)\cos(\theta_2) + \sin(\theta_2)\cos(\theta_1) & \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$P' = R(\theta_2 + \theta_1) \cdot P$

$\therefore$ Successive rotation is additive.

3. Scaling :

Q. Prove that two successive scaling are multiplicative

By applying two successive scaling with scaling parameters $S_1 = [S_{x_1}\ S_{y_1}]^T$ and $S_2 = [S_{x_2}\ S_{y_2}]^T$ on point $[x\ y]^T$, we get

$$P' = S_2 \cdot S_1 \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_2}S_{x_1} & 0 & 0 \\ 0 & S_{y_2}S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\therefore x' = x \cdot (S_{x_2} S_{x_1})$$

$$y' = y \cdot (S_{y_2} S_{y_1})$$

Thus, two successive scaling is multiplicative.

3.7.2 General Pivot Point Rotation

Q. Find out the composite transformation matrix to rotate a given 2D object by an amount θ about the given point (x_r, y_r).

Q. Derive transformation matrix for general pivot point rotation.

Q. Write a short note on : General pivot point rotation.

We have already discussed rotation of object with respect to some pivot point (x_r, y_r). In this section, we will derive generalized 3 × 3 transformation matrix for rotation with respect to a reference (pivot) point.

General pivot point rotation can be performed by a sequence of the following steps :

1. Translate the pivot point to the origin.
2. Rotate the object.
3. Translate back the pivot point to its original place.

If M is the final transformation matrix, then

$$M = T^{-1} \cdot R \cdot T$$

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

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

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Above matrix represents the transformation matrix for rotation about a pivot point in homogenous coordinate.

Example 3.7.1 : Write 3 × 3 transformation matrix for each of the following rotation about the origin (a). Counterclockwise rotation by 180° , (b) Clockwise rotation by 90°.

Solution :

(a) 3 × 3 transformation matrix for counterclockwise rotation by 180° about the origin.

$$M_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(180°) & -\sin(180°) & 0 \\ \sin(180°) & \cos(180°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) 2 × 2 transformation matrix for clockwise rotation by 90°, So $\theta = -90°$

$$M_2 = \begin{bmatrix} \cos(-90°) & -\sin(-90°) & 0 \\ \sin(-90°) & \cos(-90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3.7.2 : Rotate a triangle defined by A (0,0), B (6,0) & C (3,3) by 90° about origin in anti-clockwise direction. **(W-18, 6 Marks)**

Solution :

Here, rotation is to be performed with respect to origin by 90^0 in anticlockwise rotation. So $\theta = +90°$.

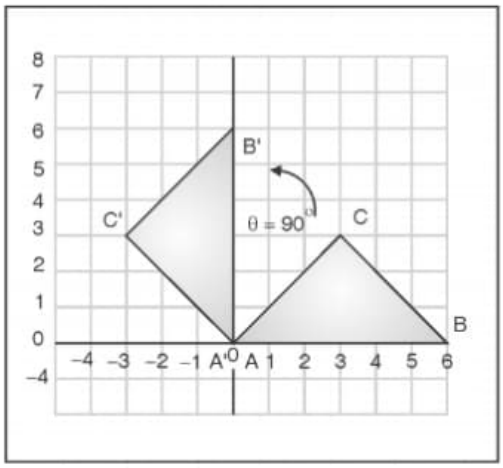


Fig. P. 3.7.2 : Output of said rotation operation

The transformed coordinates of triangle is computed as,

$$P' = R \cdot P$$

$$= \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 \\ \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -3 \\ 0 & 6 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	A' (0, 0)
B (6, 1)	B' (0, 6)
C (3, 3)	C' (-3, 3)

Example 3.7.3 : Consider the square A(1,0), B (0,0), C (0,1), D (1,1). Rotate the square ABCD by 45° anticlockwise about point A (1,0). **(W-18, 4 Marks)**

Solution :

Square is to be rotated about a reference point A(1, 0) by 45°. The desired operation is achieved by performing following steps:

1. Translate reference point A(1, 0) to origin
2. Perform rotation by 45° in anticlockwise direction
3. Inverse translation of point A

The transformation matrix for this sequence is given by $M = T^{-1}1. R_{(\theta=45°)} . T$

$$P' = M . P$$

$$P' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.709 & -0.709 & 0 \\ 0.709 & 0.709 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Multiply first two and last two matrices,

$$= \begin{bmatrix} 0.709 & -0.709 & 1 \\ 0.709 & 0.709 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0.291 & -0.418 & 0.291 \\ 0 & -0.709 & 0 & 0.709 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (1, 0)	A′ (1, 0)
B (0, 0)	B′ (0.291, -0.709)
C (0, 1)	C′ (-0.418, 0)
D (1, 1)	D′ (0.291, 0.709)

Example 3.7.4 : Perform 45° rotation of a triangle A(0, 0), B(1, 1) and C(5, 2). Find transformed coordinates after rotation, (a) About origin, (b) About P(– 1, – 1).

Solution :

Triangle A (0, 0), B (1, 1), C (5, 2) is to be rotated by 45°.

(a) Rotation with respect to the origin.

$$P' = M.P = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{3}{\sqrt{2}} \\ 0 & \sqrt{2} & \frac{7}{\sqrt{2}} \\ 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	A′ (0, 0)
B (1, 1)	B′ (0, $\sqrt{2}$)
C (5, 2)	C′ ($\frac{3}{\sqrt{2}}, \frac{7}{\sqrt{2}}$)

(b) Rotation with respect to P (– 1, – 1)

Rotated co-ordinates with respect to some reference point (x_r, y_r) is given by,

$$x' = x_r + (x - x_r) \cdot \cos\theta - (y - y_r) \cdot \sin\theta$$

$$y' = y_r + (x - x_r) \cdot \sin\theta + (y - y_r) \cdot \cos\theta$$

Let's consider $A'\left(x_1', y_1'\right)$, $B'\left(x_2', y_2'\right)$ and $C'\left(x_3', y_3'\right)$

$$x_1' = -1 + (0+1) \cdot \frac{1}{\sqrt{2}} - (0+1) \cdot \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} = -1$$

$$y_1' = -1 + (0+1) \cdot \frac{1}{\sqrt{2}} + (0+1) \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} = -1 + \sqrt{2}$$

$$x_2' = -1 + (1+1) \cdot \frac{1}{\sqrt{2}} - (1+1) \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{2}{\sqrt{2}} - \frac{2}{\sqrt{2}} = -1$$

$$y_2' = -1 + (1+1) \cdot \frac{1}{\sqrt{2}} + (1+1) \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{2}{\sqrt{2}} + \frac{2}{\sqrt{2}} = -1 + 2\sqrt{2}$$

$$x_3' = -1 + (5+1) \cdot \frac{1}{\sqrt{2}} - (2+1) \cdot \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{6}{\sqrt{2}} - \frac{3}{\sqrt{2}} = -1 + \frac{3}{\sqrt{2}}$$

$$y_3' = -1 + (5+1) \frac{1}{\sqrt{2}} + (2+1) \cdot \frac{1}{\sqrt{2}}$$

$$= -1 + \frac{6}{\sqrt{2}} + \frac{3}{\sqrt{2}} = -1 + \frac{9}{\sqrt{2}}$$

$$\therefore \quad A' = \left(-1, -1 + \sqrt{2}\right), \quad B' = \left(-1, -1 + 2\sqrt{2}\right), \quad C' = \left(-1 + \frac{3}{\sqrt{2}}, -1 + \frac{9}{\sqrt{2}}\right)$$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	$A'\left(-1, -1 + \sqrt{2}\right)$
B (1, 1)	$B'\left(-1, -1 + 2\sqrt{2}\right)$
C (5, 2)	$C'\left(-1 + \frac{3}{\sqrt{2}}, -1 + \frac{9}{\sqrt{2}}\right)$

Example 3.7.5 : Consider a triangle with vertices A(1,1), B(5,2) and C(3,4). Find out the transformation matrix which rotates given triangle about point C(3,4) by an angle 300 clockwise. Also, find the coordinates of rotated triangle.

Solution :

The given triangle is A(1, 1), B(5, 2), (3, 4). This triangle is to be rotated about C(3, 4) by 30° in a clockwise direction,

So, $\theta = -30°$

Transformation sequence would be :

1. Translate C(3, 4) to the origin.
2. Rotate by 30° in a clockwise direction.
3. Inverse translation.

$\therefore$ Transformation matrix, $M = T^{-1} \cdot R_{(\theta = -30)} \cdot T$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-30°) & -\sin(-30°) & 0 \\ \sin(-30°) & \cos(-30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{-4-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{3-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{2-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{11-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates, $P' = M \cdot P = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{2-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{11-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 5 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$

$$P' = M \cdot P = \begin{bmatrix} \frac{3-2\sqrt{3}}{2} & \frac{2\sqrt{3}+4}{2} & 3 \\ \frac{10-3\sqrt{3}}{2} & \frac{6-2\sqrt{3}}{2} & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (1, 1)	$A'\left(\frac{3-2\sqrt{3}}{2}, \frac{10-3\sqrt{3}}{2}\right)$
B (5, 2)	$B'\left(\frac{2\sqrt{3}+4}{2}, \frac{6-2\sqrt{3}}{2}\right)$
C (3, 4)	C' (3, 4)

3.7.3 General Pivot Point Scaling

Q. Derive transformation matrix for general pivot point scaling.

Q. Write a short note on : General pivot point scaling.

This operation is similar to general pivot point rotation. We will derive a generalized matrix to represent scaling with respect to reference (pivot) point (x_r, y_r). The sequence of operation would be :

1. Translate pivot point to the origin.
2. Scale the object by given scaling parameters.
3. Translate back the pivot point to the original place.

Let, M be the final composite transformation matrix, then

$$M = T^{-1}.S.T$$

$$= \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_r & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & -S_r x_r \\ 0 & S_y & -S_y y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & x_r(1-S_x) \\ 0 & 1 & y_r(1-S_y) \\ 0 & 0 & 1 \end{bmatrix}$$

Above matrix represents the transformation matrix for scaling about pivot point in homogenous coordinate.

Example 3.7.6 : Consider a square with a left-bottom corner at (2, 2) and right-top corner at (6, 6). Find out the transformation matrix which makes its size half such that its centre remains the same.

Solution :

Let us consider the vertices of a square are A (2, 2), B (6, 2), C (6, 6) and D (2, 6) center of the square is R (4, 4).

If we perform scaling with respect to R, then center remains the same. To reduce the size by half, we should set

$$S_x = S_y = 0.5$$

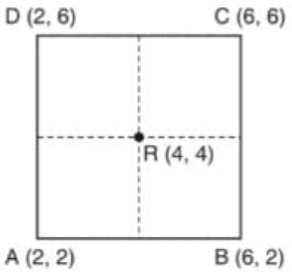


Fig. P. 3.7.6

Required transformation matrix would be a sequence of following operations :

1. Translate reference point to the origin
2. Perform scaling
3. Inverse translation

$\therefore$ Required transformation matrix, $M = T^{-1} . S . T$

$$= \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -4 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 4 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & -2 \\ 0 & 0.5 & -2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 2 \\ 0 & 0.5 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

3.8 Reflection

Q. What is reflection ? Derive transformation matrices for various cases of reflection.

- Almost all graphics packages provide functions for basic transformation operations like translation, scaling and rotation. Some packages provide the facility of additional functions like reflection and shearing.
- **Reflection** is the transformation which produces a mirror image of an object about a given axis.
- Reflection is achieved by rotating an object by 180° around reference axis, perpendicular to the XY plane.
- Reflection does not alter the shape and size of the object, so this is a rigid body transformation.

Let us discuss the various cases of reflection.

3.8.1 Reflection about X-Axis (Y = 0 Line)

Q. Derive transformation matrix for reflection about X axis.

Q. Write a short note on: Reflection about X axis.

Reflection about X-axis is also known as a reflection about the Y = 0 line.

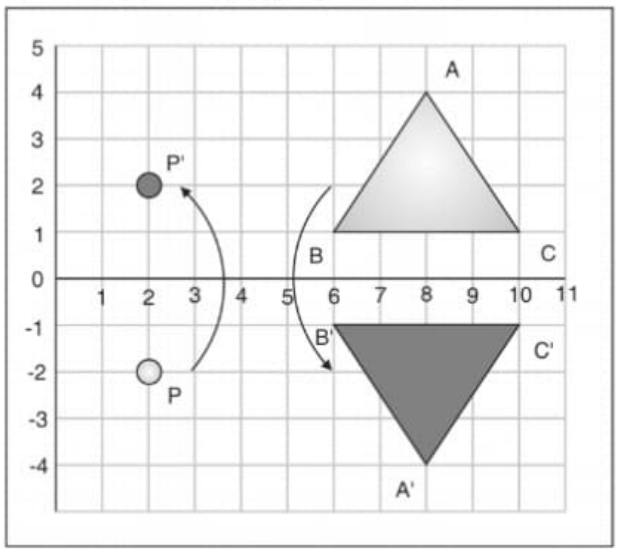


Fig. 3.8.1 : Reflection about the X axis

Reflection about X-axis only flips y-coordinate, x coordinate remains as it is. Point P(2, -2) becomes P'(2, 2) after reflection about X axis.

Reflection about X-axis is depicted in Fig. 3.8.1.

From Fig. 3.8.1, it is clear that,

$$\therefore x' = x$$

$$y' = -y$$

Transformation matrix for reflection about X axis is given as,

$$Ref_{(y=0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.8.2 Reflection About Y Axis (X = 0 Line)

Q. Derive transformation matrix for reflection about Y axis.

Q. Write a short note on: Reflection about Y axis.

Reflection about Y axis is also known as reflection about X = 0 line. Point P(-2, -3) becomes P'(2, -3) after reflection about Y axis.

Reflection about Y-axis is shown in Fig. 3.8.2

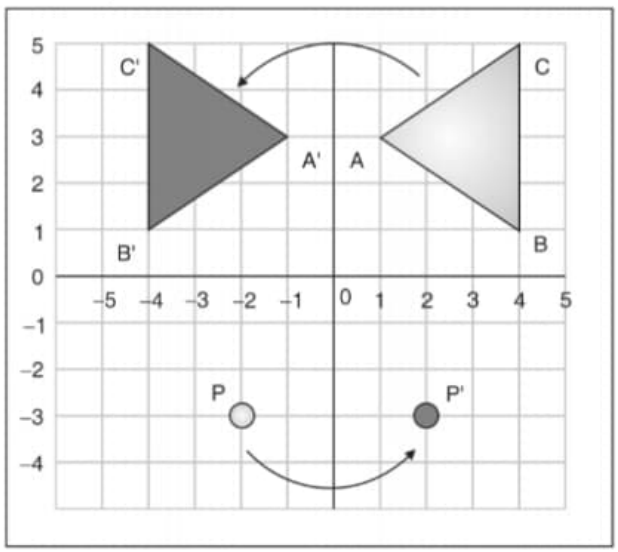


Fig. 3.8.2 : Reflection about the Y axis

Reflection about Y-axis only flips x-coordinate, y coordinate remains as it is.

$$\therefore x' = -x$$

$$y' = y$$

Transformation matrix for reflection about Y axis is given as,

$$Ref_{(x=0)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.8.3 Reflection about X = Y Axis

Q. Derive transformation matrix for reflection about X = Y axis.

Q. Write a short note on: Reflection about X = Y axis.

When point (x, y) is reflected about a line X = Y, it becomes (y, x). As shown in Fig. 3.8.3, P(-1, -4) became P'(-4, -1) after the reflection.

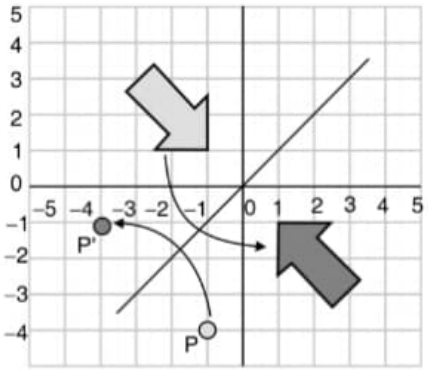


Fig. 3.8.3 : Reflection about X = Y axis

This operation swaps x and y coordinates.

$$\therefore \quad x' = y$$

$$y' = x$$

Transformation matrix for reflection about Y = X line is given as,

$$Ref_{(x=y)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.8.4 Reflection about X = – Y Axis

Q. Derive transformation matrix for reflection about X = – Y axis.

Q. Write a short note on: Reflection about X = -Y axis.

When point (x, y) is reflected about a line X = – Y, it becomes (– y, – x). As shown in Fig. 3.8.4,

P(1, 4) became P'(– 4, – 1) after the reflection.

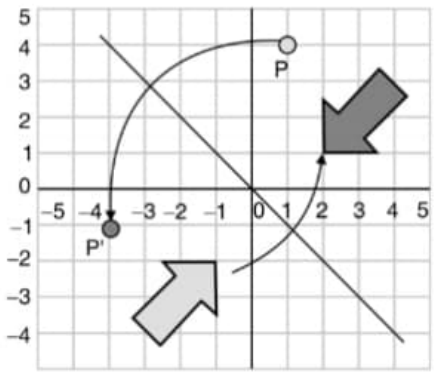


Fig. 3.8.4 : Reflection about X = – Y axis

This operation will swap x and y coordinates and also change their sign.

$$\therefore \quad x' = -y$$

$$y' = -x$$

Transformation matrix for reflection about Y = – X line is given as,

$$Ref_{(x=-y)} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Last both operations can be derived by aligning line Y = X or Y = – X with either of the principal axis and taking reflection about that particular axis followed by inverse rotation.

3.8.5 Reflection about Origin

Q. Derive transformation matrix for reflection about origin.

Q. Write a short note on Reflection about origin.

Rotation about the origin is carried out by flipping both, x and y coordinates. This is actually 180° rotation about an axis that is perpendicular to the XY plane and that passes through the coordinate origin.

This operation flips both the coordinates.

$$\therefore \quad x' = -x$$

$$y' = -y$$

Transformation matrix for reflection about origin is given as,

$$Ref_{(origin)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.8.6 Reflection about Any Line y = mx + c

Q. Derive transformation matrix for reflection about any arbitrary line.

Q. How to perform reflection about a line y = mx + c.

- Reflection about any arbitrary line y = mx + c can be accomplished by combination of translation, rotation and reflection.
- First translate the line such that it passes through the origin.
- Then rotate the line so that it aligns with one of the principal axes and perform reflection about it.
- Finally, restore the line back to its actual position by performing inverse rotation and inverse translation.
- The composite transformation matrix for this sequence of operation is obtained by multiplying following matrices.

$$M = T^{-1} . R^{-1} . Ref . R.T$$

We can implement reflections with respect to the coordinate axes or coordinate origin as scaling transformations with negative scaling factors.

3.8.7 Reflection about a Line Parallel to X Axis

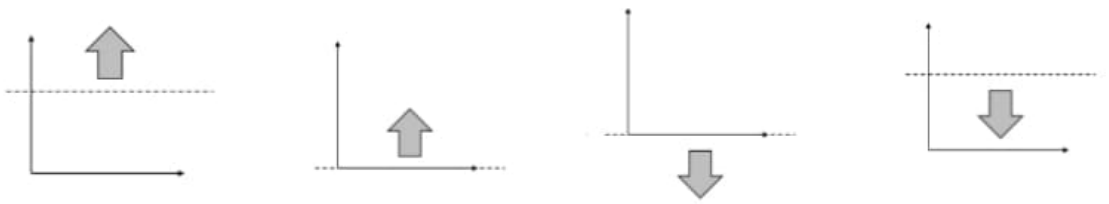
Reflection about a line parallel to X axis is carried out by following steps :

Step 1 : Translate the line such that it aligns with X axis (Refer Fig. 3.8.5(b)).

Step 2 : Perform the reflection about X axis (Refer Fig. 3.8.5(c)).

Step 3 : Translate back the line to its original position (Refer Fig. 3.8.5(d)).

The sequence of operation is depicted in Fig. 3.8.5.



(a) Grey arrow is to be reflected about dotted (reference) line

(b) Translate the reference line such that it aligns with the X axis

(c) Perform reflection about the X axis

(d) Translate back the reference line to its original position

Fig. 3.8.5 : Sequence of steps to reflect object about a reference line parallel to X-axis

Transformation matrix for this operation is given by,

$$M = T^{-1} . Ref_{(Y=0)} . T$$

3.8.8 Reflection about a Line Parallel to Y Axis

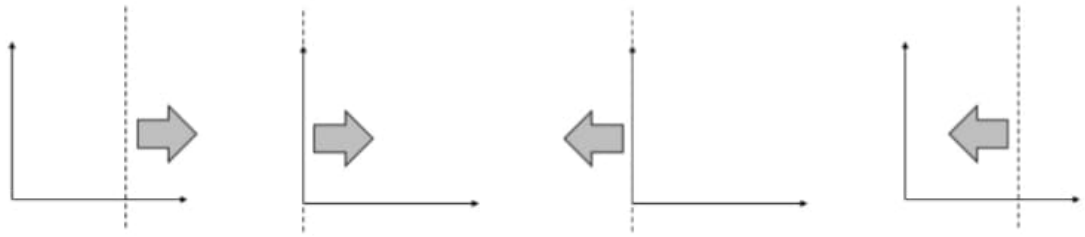
Reflection about a line parallel to Y axis is carried out by following steps :

Step 1 : Translate the line such that it aligns with Y axis (Refer Fig. 3.8.6(b)).

Step 2 : Perform the reflection about Y axis (Refer Fig. 3.8.6(c)).

Step 3 : Translate back the line to its original position (Refer Fig. 3.8.6(d)).

The sequence of operation is depicted in Fig. 3.8.6.



(a) Grey arrow is to be reflected about dotted (reference) line

(b) Translate the reference line such that it aligns with Y axis

(c) Perform reflection about the Y axis

(d) Translate back the reference line to its original position

Fig. 3.8.6 : Sequence of steps to reflect object about a reference line parallel to the Y axis

Transformation matrix for this operation is given by,

$$M = T^{-1} \cdot Ref_{(X=0)} \cdot T$$

Example 3.8.1 : Show that transformation matrix for a reflection about line y = x is equivalent to reflection about X-axis followed by counterclockwise rotation of 90°.

Solution :

Let's say M_1 is the transformation matrix for a reflection about line y = x

$$\therefore M_1 = Ref_{(y=x)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

And M_2 is the transformation matrix for reflection about X axis followed by counterclockwise rotation by 90°.

$$\therefore M_2 = R_{(\theta=90^\circ)} \cdot Ref_{(y=0)}$$

$$= \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

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

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

Here, $M_1 = M_2$, hence given statement is proved.

Example 3.8.2 : Prove with example two pure reflection transformation are applied successfully, the result is pure rotation.

Solution :

The determinant of a pure rotation matrix is always 1.

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

$$\Delta = \cos^2\theta + \sin^2\theta = 1$$

Let's perform two pure reflections and check if the resultant matrix has determinant 1. If it is so, it is pure rotation, otherwise not.

Consider following two pure reflection operation.

1. Reflection about X axis :

$$\therefore \quad M_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

2. Reflection about Y axis :

$$M_2 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Resultant transformation matrix $M = M_2 \cdot M_1$

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

$$\Delta = 1 - 0 = 1$$

Hence, it is proved that two successive pure reflections are equivalent to the pure rotation.

Example 3.8.3 : A triangle with vertices (2, 4), (6, 4), (4, 7) is to be reflected about the line with equation $Y = \frac{1}{2}(X + 4)$, give different steps explanation and matrix representation.

Solution :

Let us assume triangle vertices are A (2, 4), B (6, 4) and C (4, 7). Triangle ABC is to be reflected about line $Y = \frac{1}{2}(X + 4)$.

This line is not aligned with any of the principal axes. So we first align it with X-axis and then perform reflection.

$$Y = \frac{1}{2}(X + 4) = \frac{X}{2} + 2$$

Compare it with $y = mx + c$, $\therefore \quad m = \frac{1}{2}$ and $c = 2$

$$\tan\theta = m = \frac{1}{2} \therefore \theta = \tan^{-1}\left(\frac{1}{2}\right) = 85.24$$

We need to perform following steps :

Step 1 : Y intercept of line is c = 2. Translate line by [0 −2 1] such that line pass through origin :

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

Step 2 : Line is making angle $\theta = 85.24°$ with X-axis. So rotate line by 85.24° in clockwise direction to align it with X axis.

$$\therefore \quad R_{(\theta)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(-85.24) & -\sin(85.24) & 0 \\ \sin(-85.24) & \cos(85.24) & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.083 & 0.997 & 0 \\ -0.997 & 0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 3 : Perform reflection about X axis.

$$\therefore \quad Ref_{(y=0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 4 : Inverse rotation to move line in original orientation.

$$\therefore R_{(\theta)}^{-1} = R_{(-\theta)} = \begin{bmatrix} 0.083 & -0.997 & 0 \\ 0.997 & 0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Step 5 : Inverse translation to reposition line as its actual location.

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

$\therefore$ Composite transformation matrix M would be,

$$M = T^{-1} \cdot R_{(\theta)}^{-1} \cdot Ref_{(y=0)} \cdot R_{(\theta)} \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & -0.997 & 0 \\ 0.997 & 0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & 0.997 & 0 \\ -0.997 & 0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & -0.997 & 0 \\ 0.997 & 0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & 0.997 & -1.954 \\ -0.997 & 0.083 & -0.166 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying 2nd and 3rd matrices

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & 0.997 & 0 \\ 0.997 & -0.083 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.083 & 0.997 & -1.954 \\ -0.997 & 0.083 & -0.166 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -0.987 & 0.164 & -0.328 \\ 0.166 & 0.967 & -1.934 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -0.987 & 0.164 & -0.328 \\ 0.166 & 0.967 & 0.066 \\ 0 & 0 & 1 \end{bmatrix}$$

Reflected coordinates are computed as follows :

$$P' = M \cdot P$$

$$\text{Where } P = [A\ B\ C] = \begin{bmatrix} 2 & 6 & 4 \\ 4 & 4 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} -0.987 & 0.164 & -0.328 \\ 0.166 & 0.967 & 0.066 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 6 & 4 \\ 4 & 4 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1.646 & -5.594 & -3.128 \\ 4.266 & 4.930 & 7.499 \\ 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (2, 4)	A′ (-1.646, 4.266)
B (6, 4)	B′ (-5.594, 4.930)
C (4, 7)	C′ (-3.128, 7.499)

Example 3.8.4 : Find out composite transformation matrix to reflect a triangle with vertices A (– 2, 1), B (– 1, 2) and C (– 2, 2) about line y = x + 2. Also, find the coordinates of reflected object.

Solution:

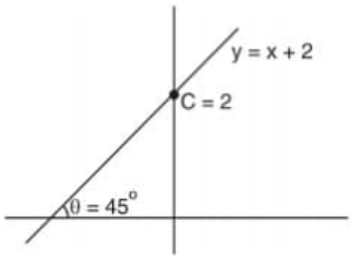
Vertices of the triangle are A (– 2, 1), B (– 1, 2) and C (– 2, 2). Triangle is to be reflected about line y = x + 2.

Compare y = x + 2 with y = mx + c

$$\therefore \quad \tan\theta = m = 1$$

$$\theta = \tan^{-1}(1) = 45°$$

$$\text{and } c = 2$$



To perform said operation, we should carry out following steps.

1. Translate line by (0, – 2) so that it passes through origin.
2. Rotate by 45° in clock wise direction such that line gets align with x axis.

$$\theta = -45°$$

3. Reflection about x axis.
4. Inverse rotation.
5. Inverse translation.

∴ Transformation matrix would be,

$$M = T^{-1} R_{(\theta)}^{-1} \cdot \text{Ref}_{(y=0)} \cdot R_{(\theta)}\, T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Put $\theta = -45°$

$$\therefore M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

Transformed coordinates are given by,

$$P' = MP$$

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

Original co-ordinates	Transformed co-ordinates
A (– 2, 1)	A′ (– 1, 0)
B (– 1, 2)	B′ (0, 1)
C (– 2, 2)	C′ (0, 0)

Example 3.8.5 : Prove that the reflection of a square ABCD [(2,2), (4,2), (4,4), (2,4)] about x-axis (y =0) and then rotation of the resulting square about 600 will not be the same if the order of transformation (first rotation and then reflection) is changed.

Solution :

Case-I : Reflection about x axis followed by rotation by $\theta = 60°$.

Composite transformation matrix of said sequence of operation is given by,

$$M_1 = R_{(\theta = 60°)} \cdot Ref_{(y=0)}$$

Transformed coordinates would be $P_1' = M_1 \cdot P$

$$P_1' = \begin{bmatrix} \cos(60°) & -\sin(60°) & 0 \\ \sin(60°) & \cos(60°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ 2 & 2 & 4 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & \frac{-\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ -2 & -2 & -4 & -4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P_1' = \begin{bmatrix} 1+\sqrt{3} & 2+\sqrt{3} & 2+2\sqrt{3} & 1+2\sqrt{3} \\ \sqrt{3}-1 & 2\sqrt{3}-1 & 2\sqrt{3}-2 & \sqrt{3}-2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Case II : Rotation by $\theta = 60°$ followed by reflection about X axis.

Transformed co-ordinates would be,

$$P_2' = Ref_{(y=0)} \cdot R_{(\theta=60°)} P$$

$$P_2' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & \frac{-\sqrt{3}}{2} & 0 \\ \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ 2 & 2 & 4 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{2} & \frac{-\sqrt{3}}{2} & 0 \\ \frac{-\sqrt{3}}{2} & \frac{-1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 4 & 2 \\ 2 & 2 & 4 & 4 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P_2' = \begin{bmatrix} 1-\sqrt{3} & 2-\sqrt{3} & 2-2\sqrt{3} & 1-2\sqrt{3} \\ -\sqrt{3}-1 & -2\sqrt{3}-1 & -2\sqrt{3}-2 & -\sqrt{3}-2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Here, $P_1' \neq P_2'$ so we can say that rotation followed by reflection is not the same as reflection followed by a rotation.

3.9 Shearing

Q. Derive transformation matrices for shearing in X and Y direction.

- Shearing is typically applied to soft objects like rubber, sheet, cloth etc.
- When we apply a horizontal force to the book which is seating upright, it bends in the direction of the force. It is a good example of shearing.
- Shearing deforms the shape of an object and hence it is not rigid body transformation.
- This deformation is proportional to shearing force and distance of the point of an object from the baseline.

3.9.1 X-direction Shearing

Q. How to perform shearing in X direction? Derive its transformation matrix.

- When force is applied in the horizontal direction on top of the book, pages slide over each other. Neither the base of book alters nor the height of any page change (internal layer).
- Effect of shearing in X direction is depicted in Fig. 3.9.1.

It is intuitive that as the height of the page increases from the base, its displacement would be more. So change in x-coordinate depends on two parameters :

1. Height of internal layer from the base.
2. The magnitude of force.

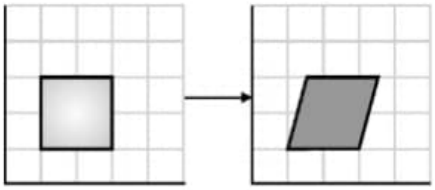


Fig. 3.9.1: Shape deformation after shearing in X direction

So, transformed coordinates are given by

$$x' = x + Sh_x \cdot y$$

$$y' = y$$

Where, Sh_x is a shear parameter in X-direction or magnitude of the force and y is the height of internal layer from the base.

The transformation matrix for X-direction shearing is,

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

Transformation coordinates are computed as,

$$P' = SH_x \cdot P$$

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

Example 3.9.1 : Apply shearing on cube with vertices A(0,0), B(2,0), C(2,2) and D(0,2), where $Sh_x = 3$.

Solution :

For x-direction shearing with y = 0 as reference line

$$x' = x + Sh_x \cdot y$$

$$y' = y$$

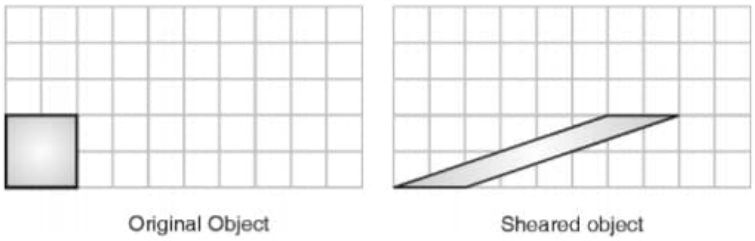


Fig. P. 3.9.1 : Shearing in the X direction

For Point A, $x' = 0 + 3(0) = 0$

For Point B, $x' = 2 + 3(0) = 2$

For Point C, $x' = 2 + 3(2) = 8$

For Point D, $x' = 0 + 3(2) = 6$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	A′ (0, 0)
B (2, 0)	B′ (2, 0)
C (2, 2)	C′ (8, 0)
D(0, 2)	D′(6, 0)

Shearing about a line parallel to X-axis.

Q. Derive transformation matrix for shearing with respect to line parallel to X-axis.

Like other operations, the previous transformation matrix is also position dependent. The transformation matrix SH_x derived above is applicable only when the base of the object is on X-axis, means when the reference line is Y = 0.

We can derive generalized matrix for any reference line $Y = Y_{ref}$ parallel to X-axis using the following steps :

1. Translate line by– Yref so the line gets align with X-axis.
2. Apply pure shear operation in the X direction.
3. Translate line by + Yref to bring it back to the original position.

$\therefore$ The composite transformation matrix, $M = T^{-1}.SH_x.T$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & Sh_x & -y_{ref}.Sh_x \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & Sh_x & -y_{ref}.Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3.9.2 : Apply shearing on a unit square whose base is on line $y_{ref} = -2$ with $Sh_x = 2$.

Solution :

$$P' = M.P$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & -y_{ref}.Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\therefore x' = x + y.Sh_x - y_{ref}Sh_x$$

$$y' = y$$

Y-coordinate of all point will remain as it is, the only x will change.

$$\text{For point A, } x' = 0+2(-2)-(-2)2 = -4$$

$$\text{For point B, } x' = 1+2(-2)-(-2)2 = 3$$

$$\text{For point C, } x' = 1+(-1)2-(-2)2 = 3$$

$$\text{For point D, } x' = 0+(-1)2-(-2)2 = 2$$

Position of sheared cube is shown in Fig. P. 3.9.2(a).

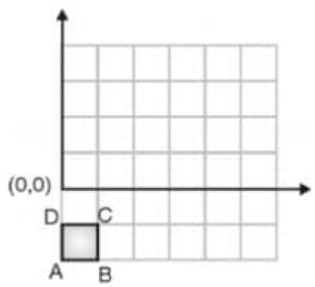


Fig. P. 3.9.2

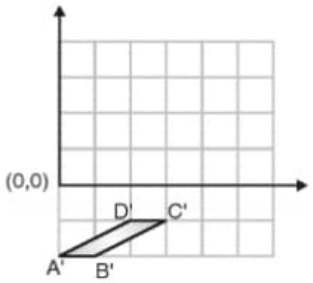


Fig. P. 3.9.2

Original Coordinate	Transformed Coordinate
A(0, – 2)	(0, – 2)
B(1, – 2)	(1, – 2)
C(1, – 1)	(3, – 1)
D(0, – 1)	(2, – 1)

3.9.2 Y direction Shearing

Q. How to perform shearing in Y direction? Derive its transformation matrix.

When shearing is applied in the y-direction, only y coordinates will change, x coordinates remain unchanged.

The change in y-direction is proportional to the magnitude of its x-coordinate and force applied in the y-direction.

$$\therefore \; x' = x$$

$$y' = y + x \cdot Sh_y$$

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

Shearing about a line parallel to Y-axis.

Q. Derive transformation matrix for shearing with respect to line parallel to Y-axis.

We can derive generalized matrix for any reference line $X = X_{ref}$ parallel to Y-axis steps are as follows :

1. Translate line by $-X_{ref}$ so the line will align with Y-axis.
2. Apply pure shear operation in the Y direction.
3. Translate line by $+X_{ref}$ to bring it back to the original place.

$\therefore$ The composite transformation matrix $M = T^{-1}.SH_y.T$

$$M = \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{ref} \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y . x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

Y-direction shear relative to line $X = x_{ref}$ can be produced with transformation matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y . x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3.9.3 : Apply y-direction shearing on unit square with a base on X =0 line and one corner at the origin with $Sh_y = 1/2$.

Solution :

Position of a unit cube is shown in the Fig. P. 3.9.3. Coordinates of vertices would be A(0, 0), B(1, 0), C(1, 1) and D(0, 1). For Y direction shear

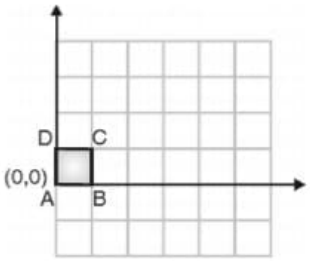


Fig. P. 3.9.3

$$x' = x$$

$$y' = y + x . Sh_y$$

For point A, $y' = 0 + (0)1/2 = 0$

For point B, $y' = 0 + (1)1/2 = 1/2$

For point C, $y' = 1 + (1)1/2 = 3/2$

For point D, $y' = 1 + (0)1/2 = 1$

Position of sheared cube is shown in Fig. P. 3.9.3(a).

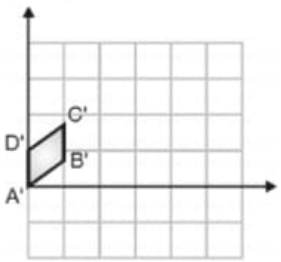


Fig. P. 3.9.3(a)

Original Coordinate	Transformed Coordinate
A(0, 0)	(0, 0)
B(0, 1)	(1, 1/2)
C(1, 1)	(1, 3/2)
D(1, 0)	(0, 1)

Example 3.9.4 : Covert the unit square to shifted parallelogram using x-direction shear transformation operation where parameter $sh_x = ½$ and $Y_{ref} = -1$ and unit square dimensions are (0, 0), (1, 0), (0, 1) and (1, 1).

Solution :

Let us consider the vertices of unit square as A (0, 0), B (1, 0), C (1, 1) and D(0, 1). Shearing parameters are $Sh_x = 1/2$ and $Y_{ref} = -1$.

Resultant co-ordinates are computed as,

$$P' = M.P$$

$$M = \begin{bmatrix} 1 & Sh_x & -Y_{ref}.Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad P' = \begin{bmatrix} 1 & \frac{1}{2} & \frac{-1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	A' (– 1/2, 0)
B (1, 0)	B' (1/2, 0)
C (1, 1)	C' (1, 1)
D (0, 1)	D' (0, 1)

Example 3.9.5 : Apply the shearing transformation to Square with A(0,0), B(1,0), C(1,1) and D(0,1) as given below

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

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

Solution :

(a) Here, reference line is $Y_{ref} = 1$ and shear parameter is 0.5, so $Sh_x = 0.5$.

The transformation matrix for shearing with respect to reference line Y_{ref} is,

$$M_1 = \begin{bmatrix} 1 & Sh_x & -Y_{ref}.Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates would be, $P' = M_1.P$

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

(b) Here, reference line is $X_{ref} = -1$, and shear parameter is 0.5, so $Sh_y = 0.5$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -X_{ref}.Sh_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

$\therefore$ Transformed coordinates are,

$$P' = M_2 P = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

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

Original coordinates	After shearing in the X direction	After shearing in the Y direction
A (0, 0)	A′ (0.5, 0)	A′ (0, 0.5)
B (1, 0)	B′ (1.5, 0)	B′ (1, 1)
C (1, 1)	C′ (2, 1)	C′ (1, 2)
D (0, 1)	D′ (1, 1)	D′ (0, 1.5)

3.10 Solved Examples

Example 3.10.1 : Translate a Square ABCD with the coordinates A(0,0), B(5,0), C(5,5), D(0,5) by 2 units in X-direction and 3 units in Y-direction.

Solution :

Co-ordinates of square are A (0, 0), B (5, 0), C (5, 5) and D (0, 5). It is to be translated by 2 and 3 units in X and Y directions respectively.

So, $t_x = 2$ and $t_y = 3$

Transformed co-ordinates are computed as,

$$P' = M.P.$$

$$= \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot P = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 5 & 5 & 0 \\ 0 & 0 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 2 & 7 & 7 & 2 \\ 3 & 3 & 8 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (0, 0)	A′ (2, 3)
B (5, 0)	B′ (7, 3)
C (5, 5)	C′ (7, 8)
D (0, 5)	D′ (2, 8)

Example 3.10.2 : Find the sequence of transformation to scale the object with respect to point A in XY plane.

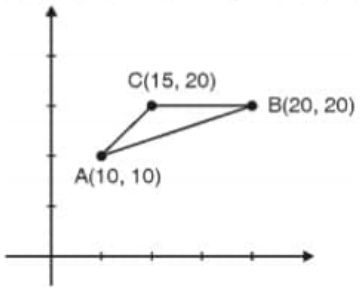


Fig. P. 3.10.2

Solution :

Here, an object is to be scaled with respect to point A. so reference point is A (10, 10). Scaling with respect to the reference point is achieved by the following steps.

1. Translate reference point to the origin.
2. Scale the object
3. Inverse translation of reference point

$\therefore$ Composite transformation $M = T^{-1}. S. T$

Consider S_x and S_y are the scaling parameters in X and Y direction respectively.

$$M = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

This is the required sequence of transformations.

Example 3.10.3 : Consider the triangle with vertices (10, 10), (40, 10), (30, 30). Apply scaling transformation with scale factor 5 in X and Y direction. Draw the triangle before and after transformation.

Solution :

Let's consider the triangle A (10, 10), B (40, 10), C (30, 30).

Scaling factors in X and Y direction is 5.

$\therefore \quad s_x = s_y = 5$

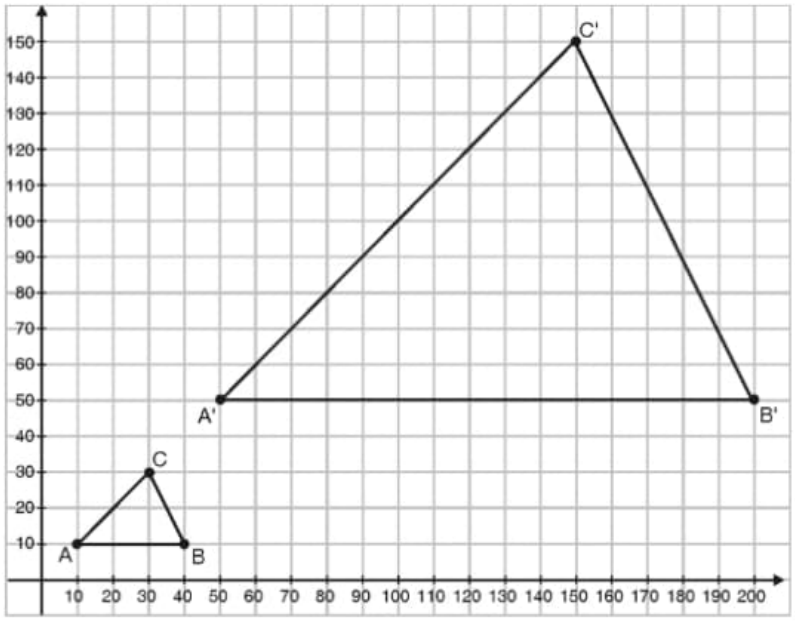


Fig. P. 3.10.3

Scaling operation is carried out as,

$$P' = S \cdot P$$

$$P' = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 40 & 30 \\ 10 & 10 & 30 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 50 & 200 & 150 \\ 50 & 50 & 150 \\ 1 & 1 & 1 \end{bmatrix}$$

Original coordinates	Transformed coordinates
A (10, 10)	A′ (50, 50)
B (40, 10)	B′ (200, 50)
C (30, 30)	C′ (150, 150)

Example 3.10.4 : Rotate a triangle ABC by an angle 30° where the triangle has coordinates A(0, 0) B(10, 2) and C(7,4).

Solution :

Triangle is to be rotated about the origin by the angle $\theta = 30°$. Let's assume P′ is the set of rotated coordinates.

$$\therefore P' = M \cdot P$$

$$= R_{(\theta = 30)} \times P$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\sqrt{3}}{2} & -1/2 & 0 \\ 1/2 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \frac{10\sqrt{3}-2}{2} & \frac{7\sqrt{3}-4}{2} \\ 0 & \frac{10+2\sqrt{3}}{2} & \frac{7+4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Example 3.10.5 : Derive 2× 2 transformation matrix for following operations :

1. Counterclockwise rotation by π/2.
2. Clockwise rotation by – π.
3. Reflection around X =0 line followed by 180° counterclockwise direction.
4. Uniform scaling with $S_x = 2$ followed by reflection about Y = – X line followed by 90° counterclockwise direction.

Solution :

1. Counterclockwise rotation by π/2 :

Here, rotation direction is counterclockwise, so θ is positive.

$$\theta = \pi/2$$

$$\text{Transformation matrix, } M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\pi/2 & -\sin\pi/2 \\ \sin\pi/2 & \cos\pi/2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

2. Clockwise rotation by $-\pi$:

Here, rotation is in clockwise direction and rotation angle $\theta = -\pi$

$$\text{Transformation matrix, } M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

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

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

3. Reflection around X=0 line followed by 180° counterclockwise direction :

Reflection about X = 0 line flips only X coordinate, it does not alter Y coordinate. Hence, the transformation matrix for reflection around X =0,

$$M_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Transformation matrix for rotation by 180° in counterclockwise direction is

$$M_2 = \begin{bmatrix} \cos\pi & -\sin\pi \\ \sin\pi & \cos\pi \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

∴ Resultant transformation matrix,

$$M = M_2 . M_1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

4. Uniform scaling with $S_x = 2$ followed by reflection about Y = – X line followed by 90° counterclockwise direction :

Composite transformation matrix for specified sequence of operation is derived as,

$$M = R_{(\theta=90°)} . Ref_{(Y=-X)} . S$$

$$M = \begin{bmatrix} \cos(90°) & -\sin(90°) \\ \sin(90°) & \cos(90°) \end{bmatrix} \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

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

Example 3.10.6 : Derive transformation matrix :

1. To increase the area of square 4 times.
2. To increase the size by four times.

Solution :

If we double the height and width of the square, its area becomes four times than original. So we need to perform scaling with $S_x = 2$ and $S_y = 2$.

∴ Resultant transformation matrix $M = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

To increase size four times, we have to multiply each side by 4.

$$\therefore S_x = S_y = 4$$

$$\therefore \text{Resultant transformation matrix, } M = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$$

Example 3.10.7 : 1. Find transformation matrix to rotate object about origin by 45° in counterclockwise direction.

2. Find new coordinates of the point (8,4) after rotation.

Solution :

1. Here, rotation direction is counterclockwise, so θ is positive.

$$\therefore \theta = 45^\circ$$

Transformation matrix is,

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

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

2. Find new coordinates of the point (8,4) after rotation.

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore P' = M \cdot P = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 4 \end{bmatrix} = \begin{bmatrix} 2\sqrt{2} \\ 6\sqrt{2} \end{bmatrix}$$

Example 3.10.8 : A triangle is defined by $\begin{bmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \end{bmatrix}$. Find transformed coordinates after the following transformation

1. 90° rotation about the origin.
2. Reflection about line X = Y.

Solution :

1. 90° rotation about origin :

Here, $\theta = 90^\circ$

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore P' = M.P$$

$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

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

Original Coordinate	Transformed Coordinate
A(2, 2)	A′(– 2, 2)
B(4, 2)	B′(– 2, 4)
C(4, 4)	C′(– 4, 4)

2. Reflection about line X = Y

Reflection about line X = Y

$$P'' = M.P' = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -2 & -2 & -4 \\ 2 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \\ -2 & -2 & -4 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(2, 2)	A'(2, – 2)
B(4, 2)	B'(4, – 2)
C(4, 4)	C'(4, – 4)

Example 3.10.9 : Perform 90° rotation of triangle with vertices A(2,2), B(4,2) and C(3,3)

1. About origin

2. About reference point (– 2,2)

Solution :

1. Rotation about origin :

Here, rotation direction is counterclockwise. So θ is positive.

$$\theta = 90^\circ$$

Let us assume P and P' are original and transformed coordinates of the object respectively.

$$\therefore P' = M.P$$

$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \end{bmatrix}$$

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

Original Coordinate	Transformed Coordinate
A(2, 2)	A'(– 2, 2)
B(4, 2)	B'(– 2, 4)
C(3, 3)	C'(– 3, 3)

2. About reference point (– 2, 2) :

The composite transformation matrix for rotation about some reference point is given as,

$$M = T^{-1}.R.T$$

Let us assume P and P' are original and transformed coordinates of the object respectively.

$$\therefore P' = M.P$$

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

$$= \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 6 & 5 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -2 & -3 \\ 6 & 8 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(2, 2)	A'(– 2, 6)
B(4, 2)	B'(– 2, 8)
C(3, 3)	C'(– 3, 7)

Example 3.10.10 : Magnify a triangle with vertices A(0,0), B(1,1) and C(5,2) to twice of its size keeping B fixed.

Solution :

This is fixed point scaling with respect to reference point B(1, 1), so we need to perform following steps to carry out the desired operation.

1. Translate B to the origin.
2. Apply scaling with $S_x = S_y = 2$.
3. Inverse translation of B.

The composite transformation matrix for scaling with respect to some reference point is given as,

$$M = T^{-1}. S. T$$

Let us assume P and P' are original and transformed coordinates of the object respectively.

$$\therefore P' = M. P$$

$$\therefore P' = T^{-1}. S. T. P$$

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

Multiplying first two and last two matrices

$$= \begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 4 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 9 \\ -1 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(0, 0)	A'(– 1, – 1)
B(1, 1)	B'(1, 1)
C(5, 2)	C'(9, 3)

Example 3.10.11 : Comment on statement: "Two dimensional rotation and scaling are cumulative, if $S_x = S_y$ or $\theta = n\pi$".

Solution :

Matrix for 2D rotation, $M_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$\text{Matrix for 2D scaling, } M_2 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can say that operations are cumulative if $M_1 M_2 = M_2 M_1$.

(A) If $S_x = S_y$

Case 1 : Rotation followed by scaling

$$M_1M_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_x & 0 \\ 0 & 0 & 1 \end{bmatrix} (S_x = S_y)$$

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

Case 2 : Scaling followed by a rotation

$$M_2M_1 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Here $M_1M_2 = M_2M_1$

So given statement is true for $S_x = S_y$.

(B) If $\theta = n\pi$

Case 1 : Scaling followed by rotation

$$M_1M_2 = \begin{bmatrix} \cos(n\pi) & -\sin(n\pi) & 0 \\ \sin(n\pi) & \cos(n\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} (\sin(n\pi) = 0, \cos(n\pi) = 1)$$

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

Case 2 : Rotation followed by scaling

$$M_2M_1 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 1 & \end{bmatrix} \begin{bmatrix} \cos(n\pi) & -\sin(n\pi) & 0 \\ \sin(n\pi) & \cos(n\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Here, $M_1M_2 = M_2M_1$, So given statement is true for $\theta = n\pi$

Example 3.10.12 : The reflection along line Y = X is equivalent to reflection along X-axis followed by counterclockwise rotation by θ. Find the value of θ.

Solution :

Transformation matrix for reflection about Y = X line

$$M_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix for reflection about X-axis

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix for counterclockwise rotation by angle θ

$$M_3 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As per given sequence of operation

$$M_1 = M_3 M_2$$

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

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

$$\therefore \cos\theta = 0, \sin\theta = 1$$

$$\therefore \theta = 90^\circ$$

Example 3.10.13 : Show that $\begin{bmatrix} \frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2} \\ \frac{2t}{1+t^2} & \frac{1-t^2}{1+t^2} \end{bmatrix}$ represents pure rotation.

Solution :

For pure rotation, the determinant of the rotation matrix is one.

$$\text{Given matrix is } M = \begin{bmatrix} \frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2} \\ \frac{2t}{1+t^2} & \frac{1-t^2}{1+t^2} \end{bmatrix}$$

The determinant of given matrix M is :

$$|M| = \frac{1-t^2}{1+t^2}\cdot\frac{1-t^2}{1+t^2} - \frac{2t}{1+t^2}\cdot\frac{-2t}{1+t^2}$$

$$= \frac{(1-t^2)^2 + 4t^2}{(1+t^2)^2} = \frac{(1+t^2)^2}{(1+t^2)^2} = 1$$

So given matrix represents pure rotation.

Example 3.10.14 : Show that two-dimensional reflection through X-axis followed by two-dimensional reflection through line Y = – X is equivalent to a pure rotation about the origin by 270°.

Solution:

Transformation matrix for two dimensional reflection through X-axis followed by two dimensional reflection through line Y = – X is,

$$M_1 = Ref_{(Y=-X)} \cdot Ref_{(Y=0)}$$

$$M_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots(1)$$

Transformation matrix for rotation about origin is,

$$M_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here $\theta = 270^\circ$, so

$$M_2 = \begin{bmatrix} \cos(270^\circ) & -\sin(270^\circ) & 0 \\ \sin(270^\circ) & \cos(270^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots(2)$$

Here $M_1 = M_2$, so statement is proved.

Note : In general, two pure reflections about a line passing through origin is equivalent to the pure rotation.

Example 3.10.15 : A mirror is placed vertically such that passes through points (10,0) and (0,10). Find the mirrored coordinate of a triangle with vertices A (2,2) B(4,2) and C(3,3).

Solution :

Arrangement of object and line is shown in Fig. P. 3.10.15.

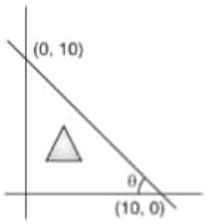


Fig. P. 3.10.15

From Fig. P. 3.10.15,

$$\tan\theta = \frac{dy}{dx} = \frac{10}{10} = 1,$$

$$\text{so, } \theta = \tan^{-1}(1) = 45^\circ$$

To reflect the given triangle about given line, we have to align it with one of the four line, x = 0, y = 0, x = y or x = – y.

Let's align it with X-axis. To align it with X-axis we should perform the following steps :

1. Translate (0,10) to the origin.
2. Perform anticlockwise rotation by $\theta = 45^\circ$
3. Perform reflection about X-axis.
4. Inverse rotation
5. Inverse translation.

So sequence of resultant transformation matrix is,

$$M = T^{-1} \cdot R^{-1} \cdot Ref_{(Y=0)} \cdot R \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45° & \sin 45° & 0 \\ \sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-45°) & \sin(-45°) & 0 \\ \sin(-45°) & \cos(-45°) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 10 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates of triangle are,

$$P' = M \cdot P$$

$$= \begin{bmatrix} 0 & -1 & 10 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 8 & 7 \\ 8 & 6 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

Reflected coordinates of triangle are A'(8, 8), B' (8, 6) and C'(7, 7)

Example 3.10.16 : Reflect triangle ABC with vertices A(2,2) B(4,2) and C(3,3) around line– 3x + 4y – 8 = 0.

Solution :

From line equation– 3x + 4y – 8 = 0

$$4y = 3x + 8$$

$$\therefore y = \frac{3x}{4} + 2 = mx + c$$

$$\therefore m = \frac{3}{4}, c = 2$$

$$\tan\theta = \frac{3}{4} \; ; \quad \therefore \sin\theta = \frac{3}{5} \; ; \quad \cos\theta = \frac{4}{5}$$

Line makes positive angle with X axis (because tan θ> 0) and it intersects Y axis at point (0, 2) (c = 2). Line orientation is shown in Fig. P. 3.10.16.

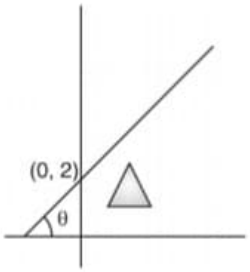


Fig. P. 3.10.16

The transformation matrix for reflection is about the given line is achieved by the following steps :

1. Translate (0,2) to the origin.
2. Rotate line by $\theta = \tan^{-1}\left(\frac{3}{4}\right)$ angle in a clockwise direction.
3. Reflect triangle about X-axis.
4. Inverse rotation by θ.
5. Inverse translation.

$$\therefore M = T^{-1} \cdot R^{-1} \cdot Ref_{(Y=0)} \cdot R \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} 4/5 & -3/5 & 0 \\ 3/5 & 4/5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & 3/5 & -6/5 \\ -3/5 & 4/5 & -8/5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4/5 & 3/5 & 0 \\ 3/5 & -4/5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & 3/5 & -6/5 \\ -3/5 & 4/5 & -8/5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 7/25 & 24/25 & -48/25 \\ 24/25 & -7/25 & 64/25 \\ 0 & 0 & 1 \end{bmatrix}$$

Coordinates of reflected triangle is given by,

$$P' = MP$$

$$= \begin{bmatrix} 7/25 & 24/25 & -48/25 \\ 24/25 & -7/25 & 64/25 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 4 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \frac{62}{25} & \frac{28}{25} & \frac{45}{25} \\ \frac{84}{25} & \frac{146}{25} & \frac{115}{25} \\ 1 & 1 & 1 \end{bmatrix}$$

$$\therefore A' = \left(\frac{62}{25}, \frac{84}{25}\right),\ B' = \left(\frac{28}{25}, \frac{146}{25}\right),\ C' = \left(\frac{45}{25}, \frac{115}{25}\right)$$

Original Coordinates	Transformed Coordinates
A(2, 4)	$A'\left(\frac{62}{25}, \frac{84}{25}\right)$
B(4, 2)	$B'\left(\frac{28}{25}, \frac{146}{25}\right)$
C(3, 3)	$C'\left(\frac{45}{25}, \frac{115}{25}\right)$

3.11 Lab Programs

Program 3.11.1 : Write a program for Two-dimensional transformation. **(Lab 7)**

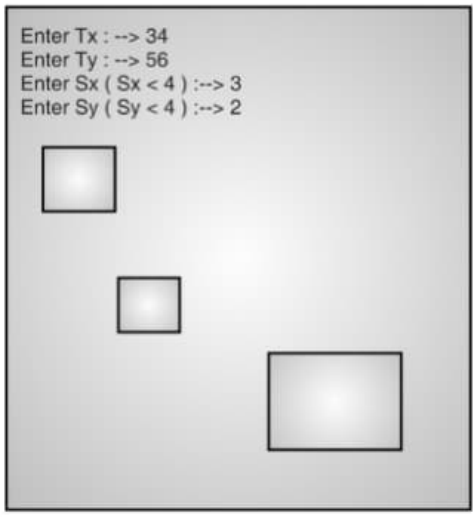
1. Translation
2. Scaling

Solution :

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
int xmin = 20, xmax = 50, ymin = 80, ymax = 110;
void DrawShape();
void Translation(int, int);
void Scaling(float, float);

void main()
{
    clrscr();
    int gdriver = DETECT;
    int gmode;
    int tx, ty;
    float sx, sy;
    initgraph(&gdriver, &gmode, "");
    DrawShape();

    printf("Enter Tx :--> ");
    scanf("%d", &tx);
    printf("Enter Ty :--> ");
    scanf("%d", &ty);
    printf("Enter Sx ( Sx< 4 ) :--> ");
    scanf("%f", &sx);
    printf("Enter Sy ( Sy< 4 ) :--> ");
    scanf("%f", &sy);

    Translation(tx, ty);
    Scaling(sx, sy);

    getch();
    closegraph();
}
void DrawShape()
```

```
{
    line(xmin, ymin, xmax, ymin);
    line(xmin, ymax, xmax, ymax);
    line(xmin, ymin, xmin, ymax);
    line(xmax, ymin, xmax, ymax);
}

void Translation(inttx, int ty)
{
    xmin = tx + xmin;
    xmax = tx + xmax;
    ymin = ty + ymin;
    ymax = ty + ymax;

    setcolor(4);
    DrawShape();
}

void Scaling(float sx, float sy)
{
    xmin = sx * xmin;
    xmax = sx * xmax;
    ymin = sy * ymin;
    ymax = sy * ymax;

    setcolor(10);
    DrawShape();
}
```

Output :



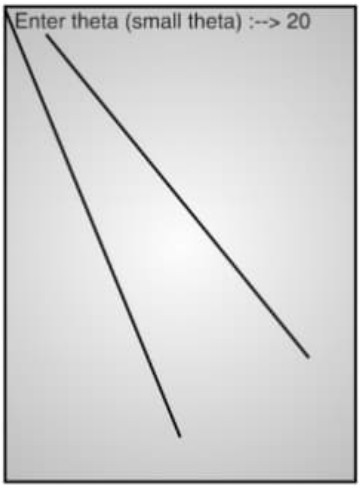
Program 3.11.2 : Write a program for Two-dimensional transformation : Rotation. **(Lab 8)**

Solution :

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

int x1, y1, x2, y2, x3, y3;

void DrawShape(int, int, int, int);
void Rotation(int);

void main()
{
      clrscr();
      int gdriver = DETECT, gmode, theta;
      initgraph(&gdriver, &gmode, "");

      x1 = 50; y1 = 50;
      x2 = 300; y2 = 300;

      DrawShape(x1, y1, x2, y2);

      printf("Enter theta (small theta) :--> ");
      scanf("%d", &theta);

      Rotation(theta);

      getch();
      closegraph();
}

void DrawShape(int x1, int y1, int x2, int y2)
{
      line(x1, y1, x2, y2);
      line(x2, y2, x3, y3);
}

void Rotation(int theta)
{
      float newx1, newy1, newx2, newy2;

      // Convert angle in degree and rotate point
      newx1 = x1*cos(theta*3.14/180) - y1*sin(theta*3.14/180);
```

```
    newy1 = x1*sin(theta*3.14/180) + y1*cos(theta*3.14/180);
    newx2 = x2*cos(theta*3.14/180) - y2*sin(theta*3.14/180);
    newy2 = x2*sin(theta*3.14/180) + y2*cos(theta*3.14/180);

    DrawShape(newx1, newy1, newx2, newy2);
}
```

Output :



Program 3.11.3 : Write a program for Two-dimensional transformation : Reflection. **(Lab 9)**

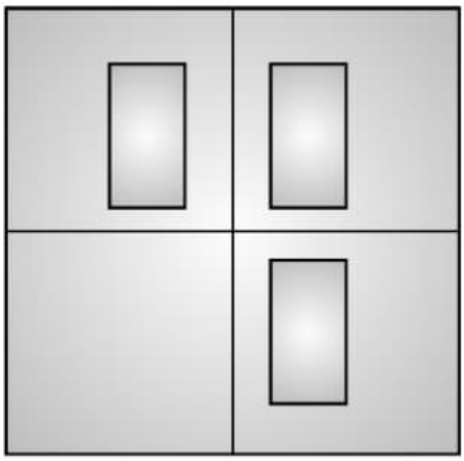
Solution :

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
int xmin, xmax, ymin, ymax;
int midx, midy;
void DrawShape(int, int, int, int);
void Reflection();

void main()
{
    clrscr();
    intgdriver = DETECT, gmode, theta;
    initgraph(&gdriver, &gmode, "");

    midx = floor(getmaxx()/2);
    midy = floor(getmaxy()/2);
```

```
    // Parameters of rectanlge
    xmin = 10;
    ymin = 10;
    xmax = 40;
    ymax = 70;

    line(midx, 0, midx, getmaxy());
    line(0, midy, getmaxx(), midy);

    DrawShape(xmin, ymin, xmax, ymax);
    Reflection();

    getch();
    closegraph();
}

void DrawShape(int xmin, int ymin, int xmax, int ymax)
{
    line(midx + xmin, midy - ymin, midx + xmax, midy - ymin);
    line(midx + xmax, midy - ymin, midx + xmax, midy - ymax);
    line(midx + xmax, midy - ymax, midx + xmin, midy - ymax);
    line(midx + xmin, midy - ymax, midx + xmin, midy - ymin);
}
void Reflection()
{
    // Reflection about X Axis
    int newxmin = xmin;
    int newxmax = xmax;
    int newymin = - ymin;
    int newymax = - ymax;
    setcolor(GREEN);
    DrawShape(newxmin, newymin, newxmax, newymax);

    // Refelction about Y Axis
    newxmin = -xmin;
    newxmax = -xmax;
```

```
    newymin = ymin;
    newymax = ymax;
    setcolor(RED);
    DrawShape(newxmin, newymin, newxmax, newymax);
}
```

Reflection :



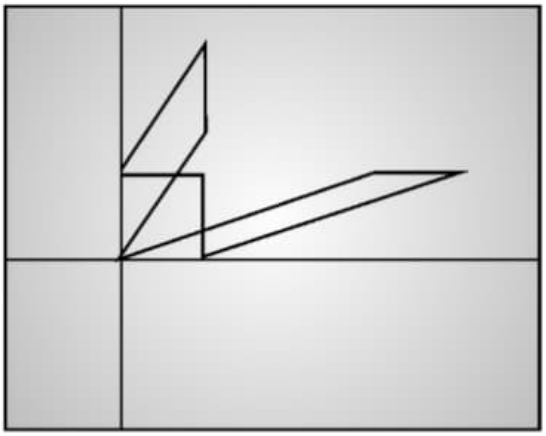
Program 3.11.4 : Write a program for Two-dimensional transformation: Shearing. **(Lab 9)**

Solution :

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
int X[4] = {0, 50, 50, 0};
int Y[4] = {0, 0, 50, 50};
int midx, midy;
void DrawShape(int [], int []);
void Shearing();
void main()
{
    clrscr();
    int gdriver = DETECT, gmode, theta;
    initgraph(&gdriver, &gmode, "");

    midx = floor(getmaxx()/2);
    midy = floor(getmaxy()/2);

    line(midx, 0, midx, getmaxy());
    line(0, midy, getmaxx(), midy);
```

```
    DrawShape(X, Y);
    Shearing();

    getch();
    closegraph();
}

void DrawShape(int X[4], int Y[4])
{
    line(midx + X[0], midy - Y[0], midx + X[1], midy - Y[1]);
    line(midx + X[1], midy - Y[1], midx + X[2], midy - Y[2]);
    line(midx + X[2], midy - Y[2], midx + X[3], midy - Y[3]);
    line(midx + X[3], midy - Y[3], midx + X[0], midy - Y[0]);
}

void Shearing()
{
    // Shearing in X Direction
    float Shx = 3;
    int newX[4], newY[4];
    for(int i=0; i<4; i++)
    {
        newX[i] = int(X[i] + Shx * Y[i]);
        newY[i] = Y[i];
    }

    setcolor(RED);
    DrawShape(newX, newY);

    // Shearing in Y Direction
    float Shy = 1.5;
    for(i=0; i<4; i++)
    {
        newX[i] = X[i];
        newY[i] = int(Y[i] + Shy * X[i]);
    }

    setcolor(YELLOW);
    DrawShape(newX, newY);
}
```

Output :



Review Questions

Topic : Basics of transformation

Q. 1 What is transformation?

Q. 2 Define rigid body transformation. Give examples of rigid body transformation operations.

Q. 3 Explain the geometric and coordinate transformation.

Q. 4 What are basic 2D transformations?

Topic : Translation

Q. 1 Write a short note on 2D translation.

Q. 2 Translate the unit square having lower left corner at (4, 4) and translation vector [4 –2].

Topic : Scaling

Q. 1 What is scaling? Derive 2D transformation matrix for scaling with respect to origin and a reference point.

Q. 2 Derive transformation matrix for scaling with respect to origin.

Q. 3 Derive transformation matrix for scaling with respect to reference point.

Q. 4 Write a short note on uniform and non-uniform scaling.

Q. 5 Scale a triangle with coordinates A(2, 2) B(6, 2) C(4, 4), Where $S_x = 3$ and $S_y = 0.5$

Q. 6 Scale a square with coordinates A(2, 2) B(4, 2) C(4, 4) D(2, 4), with respect to vertex C, Where $S_x = 2$ and $S_y = 3$.

Topic : Rotation

Q. 1 Define: 2D Rotation.

Q. 2 Derive the transformation matrices for rotation with respect to origin and reference point.

Q. 3 Derive transformation matrix for rotation about origin.

Q. 4 Derive transformation matrix for rotation about reference point.

Q. 5 Define: Orthogonal matrix.

Q. 5 Define: Orthogonal matrix.

Q. 6 State the properties of pure rotation matrix

Q. 7 Rotate a point P(10, 10) with respect to origin in clockwise direction by 90°.

Q. 8 Rotate a point P(10, 10) with respect to (5, 5) in clockwise direction by 90°.

Topic : Homogenous coordinate representation

Q. 1 Explain in brief: Homogeneous coordinate.

Q. 2 What is the need of homogenous representation?

Q. 3 List the properties of the homogeneous coordinate representation.

Q. 4 Explain 2D transformations with its homogenous coordinate representation.

Q. 5 Give homogeneous representation for following 2D transformation operations: Translation, scaling, rotation, reflection, shearing.

Topic : Composite transformation

Q. 1 What do you mean by composite transformation?

Q. 2 Prove that two successive translations are additive.

Q. 3 Prove that two successive rotations are additive.

Q. 4 Prove that two successive scaling are multiplicative.

Q. 5 Write a short note on: General pivot point rotation.

Q. 6 Consider a triangle with vertices A(2, 2), B(6, 2) and C(4,4). Find out the transformation matrix which rotates given triangle about point C(4, 4) by an angle 90^{0} clockwise. Also, find the coordinates of rotated triangle.

Q. 7 Derive transformation matrix for general pivot point scaling.

Q. 8 Consider a square with a left-bottom corner at (0, 0) and right-top corner at (6, 6). Find out the transformation matrix which makes its size half such that its center remains the same.

Topic : Reflection

Q. 1 What is reflection? Derive transformation matrices for various cases of reflection.

Q. 2 Write a short note on reflection about Y = 0 line.

Q. 3 Write a short note on reflection about X = 0 line.

Q. 4 Write a short note on reflection about X = Y line.

Q. 5 Write a short note on reflection about X = -Y line.

Q. 6 Write a short note on reflection about origin.

Q. 7 Derive transformation matrix to perform reflection about line y = mx + c.

Q. 8 Prove with example two pure reflection transformation are applied successfully, the result is pure rotation.

Q. 9 Find out composite transformation matrix to reflect a triangle with vertices A (2, 2), B (6, 2) and C (4, 4) about line y = x + 3. Also, find the coordinates of reflected object.

Topic : Shearing

Q. 1 Explain 2D shearing.

Q. 2 How to perform shearing in X direction? Derive its transformation matrix.

Q. 3 Derive transformation matrix for shearing with respect to line parallel to X-axis.

Q. 4 Apply shearing on cube with vertices A(0,0), B(2,0), C(2,2) and D(0,2), where $Sh_x = 2$.

Q. 5 Apply shearing on a unit square whose base is on $liney_{ref} = -3$ with $Sh_x = 3$.

Q. 6 How to perform shearing in Y direction? Derive its transformation matrix.

Q. 7 Derive transformation matrix for shearing with respect to line parallel to Y-axis.

Q. 8 Apply shearing on cube with vertices A(0,0), B(2,0), C(2,2) and D(0,2), where $Sh_y = 2$.

Q. 9 Apply shearing on a unit square whose base is on $linex_{ref} = -3$ with $Sh_y = 3$.

Chapter Ends...

❑❑❑

3D Transformation and Projection

Syllabus

Three Dimensional Transformations : Translation, Scaling, Rotation,
Types of Projections : Perspective and Parallel Projection

4.1 Three Dimensional Transformations

- 3D transformation is an extension of 2D transformation in space. In 3D transformation, transformation operations like translation, rotation, scaling etc. takes place in space rather than on plane.
- Like 2D transformation, we will make use of homogenous representation to describe the 3D transformation matrices.

4.1.1 Translation

Q. How to achieve 3D translation?

Q. Derive 3D transformation matrix for translation operation.

- **Three-dimensional translation** is a process of moving an object from one location to another location along the *straight path* in *space*.
- The translation is achieved by adding the required amount of shift in each direction.
- Amount of translation is specified by translation vector, $T = [t_x, t_y, t_z]$.
- Let us consider the original point P (x, y, z), which becomes P' (x', y', z') after translation. Translated coordinates are computed by following equations :

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$

Equivalent homogeneous matrix representation is :

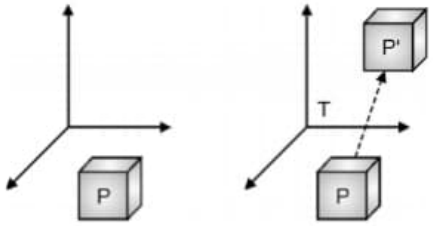
$$P' = T \cdot P$$

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

Inverse Translation matrix :

- We can achieve inverse translation by negating the shift parameters. Inverse translation matrix is given as,

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



(a) Original object P (b) Translated object P'

Fig. 4.1.1 : Translation of object in space

Example 4.1.1 : Translate a triangle with vertices A (2, 2, 2), B (3, 4, 7) and C (8, 9, 12) by translation vector T [2, 4, 5].

Solution :

Here translation vector T = [2, 4, 5]. Let us consider P = [A, B, C] is set of original points and P' = [A', B', C'] is the set of transformed coordinates. Transformed coordinates are computed as,

$$P' = T.P$$

$$= \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 5 & 10 \\ 6 & 8 & 13 \\ 7 & 12 & 17 \\ 1 & 1 & 1 \end{bmatrix}$$

Translated vertices are A' (4, 6, 7), B' (5, 8, 12) and C' (10, 13, 17).

Original Coordinates	Transformed Coordinates
A(1, 2, 8)	A'(4, 6, 7)
B(3, 4, 7)	B'(5, 8, 12)
C(8, 9, 12)	C'(10, 13, 17)

4.1.2 Scaling

W-18

Q.	Write matrices in homogeneous co-ordinate system for 3D scaling transformation.	**(W-18, 6 Marks)**
Q.	How to achieve 3D scaling?	
Q.	Derive 3D transformation matrices for scaling with respect to the origin and with respect to a reference point.	

3D scaling is performed by multiplying all coordinates of the object by some constants. Scaling alters the shape and size of the object; hence it is not rigid body transformation.

4.1.2.1 Scaling with respect to Origin

Q.	Derive 3D transformation matrices for scaling with respect to the origin.

Let, S = [S_x, S_y, S_z] be a vector of the scaling parameters in all three directions, scaling with respect to the origin of point P is computed as,

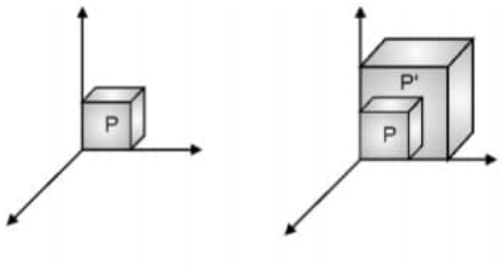
$$P' = S \cdot P$$
$$x' = S_x \cdot x$$
$$y' = S_y \cdot y$$
$$z' = S_z \cdot z$$

Matrix representation of scaling transformation :

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

If all three parameters are same, it is called **uniform scaling**, which preserves the original shape of the object to be scaled. Otherwise, it is called **non-uniform scaling**.

Fig. 4.1.2 shows the scaling operation with respect to the origin.



(a) Object P to be scaled (b) P' is the scaled object

Fig. 4.1.2 : 3D Scaling with respect to the origin

4.1.2.2 Scaling with respect to Reference Point

Q. Derive 3D transformation matrices for scaling with respect to a reference point.

Scaling with respect to some reference point (x_r, y_r, z_r) is carried out by following steps :

- Translate reference point to the origin.
- Apply scaling transformation with respect to the origin.
- Perform inverse translation to move reference point back to the original position.
- The sequence of operations to find transformed coordinates is given as,

$P' = T^{-1} . S . T . P$, more specifically

$P' = T_{(x_r, y_r, z_r)} \cdot S_{(S_x, S_y, S_z)} \cdot T_{(-x_r, -y_r, -z_r)} \cdot P$

Matrix representation of this would be,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & 1 & 0 & -y_r \\ 0 & 0 & 1 & -z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & (1-S_x)x_r \\ 0 & S_y & 0 & (1-S_y)y_r \\ 0 & 0 & S_z & (1-S_z)z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Inverse scaling is achieved by replacing scaling parameters with their reciprocal. Fig. 4.1.3 depicts the scaling operation with respect to the reference point.

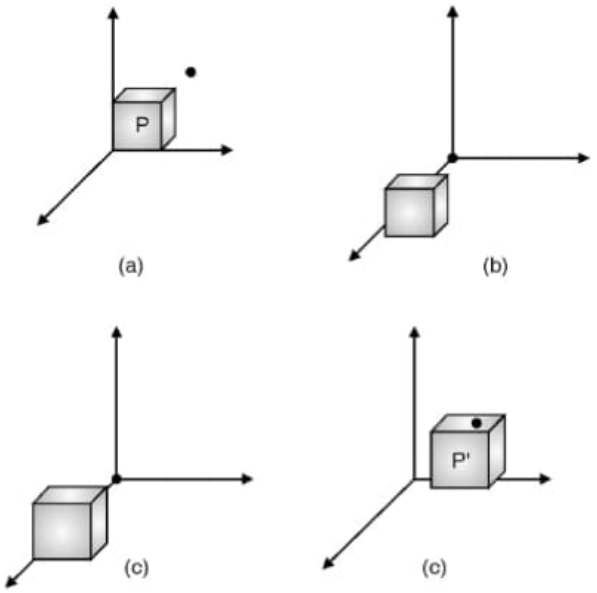


Fig. 4.1.3 : 3D scaling with respect to a reference point

4.1.3 Rotation

Q. How to achieve 3D rotation?

Q. Derive 3D transformation matrix for various cases of 3D rotation.

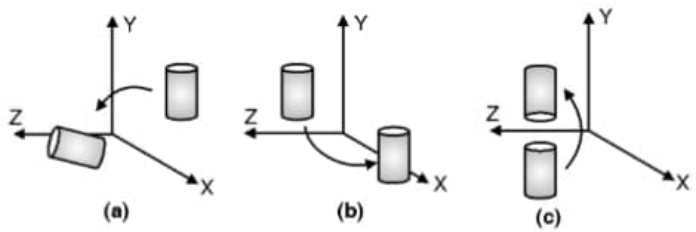
- In 3D, rotation is possible in infinite ways. We can rotate the object around three principal axes, or around any arbitrary line.
- Here, we will discuss all possible cases of rotation in space.

Note : Like 2D rotation, for anticlockwise rotation, angle is considered positive, otherwise it is considered negative.

4.1.3.1 Rotation about a Principal Axis

Q. Derive 3D transformation matrix for rotation about principal axis.

3D rotation is simply an extension of 2D rotation with added dimension. Fig. 4.1.4 depicts the rotation about X, Y and Z axis.



(a) Rotation about X axis (b) Rotation about the Y axis (c) Rotation about Z axis

Fig. 4.1.4 : 3D Rotation about principal axes

Rotation about X-axis :

- Rotation of object about X axis does not alter the X coordinate. It only affects Y and Z coordinates. Rotation about X axis is given by,

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

- Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_x(\theta) \cdot P$$

$$\text{Where, } R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y-axis :

- Similarly, rotation of an object about Y axis does not alter the Y coordinate. It only affects Z and X coordinates. Rotation about Y axis is given by,

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

- Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_y(\theta) \cdot P$$

$$\text{Where, } Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z-axis :

- Similarly, rotation of object about Z axis does not alter the Z coordinate. It only affects X and Y coordinates. Rotation about Z axis is given by,

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

- Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_z(\theta) \cdot P$$

$$\text{Where, } R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse rotation matrix :

Inverse rotation is achieved by negating the rotation angle θ. Transformation matrix for inverse rotation about Z-axis is shown below :

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

Cosine is even function and sine is odd function, hence

$$\cos(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta$$

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

Similarly, we can achieve inverse rotation about X and Y axis by replacing θ by –θ in rotation matrix $R_x(\theta)$ and $R_y(\theta)$

4.1.3.2 Rotation about a Line Parallel to Principal Axis

Q. How to perform 3D rotation about a line parallel to principal axis?

Q. Derive transformation matrix for 3D rotation about a line parallel to the principal axis.

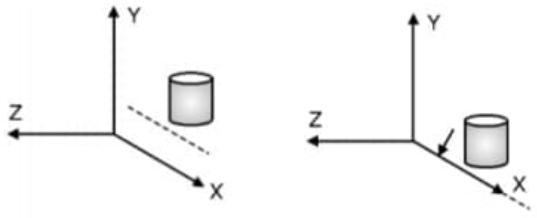
- If the rotation axis is not the principal axis, we can make it coincide with one of the principal axes using composite transformation.
- This is done by performing a necessary series of translation and rotation operations.
- If the object is to be rotated about a line parallel to one of the principal axes, we can get the desired rotation with the following steps :
  - Perform translation such that rotation axis coincides with the principal axis.
  - Perform specified rotation about that principal axis.
  - Perform inverse translation to move the rotation axis to its original location.
- Above sequence of steps can be written as :

$$P' = T^{-1} \cdot R_\lambda(\theta) \cdot T \cdot P$$

Where, $R_\lambda(\theta)$ indicates the rotation about the principal axis with which rotation axis was aligned. The composite transformation matrix is :

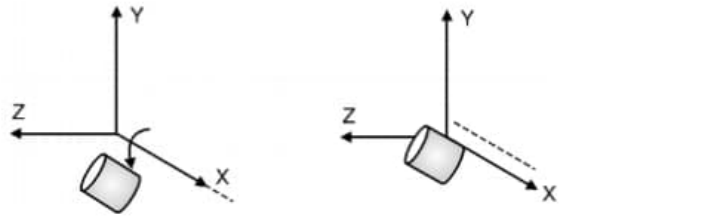
$$M = R(\theta) = T^{-1} \cdot R_\lambda(\theta) \cdot T$$

- This operation is the same as 2D rotation about a pivot point in the plane. Fig. 4.1.5 depicts the rotation about a line parallel to X-axis. The dashed line indicates rotation axis.



(a) Object to be rotated and rotation axis **(b) Translated rotation axis**

Fig. 4.1.5 (Cont...)



(c) Rotated object (d) Inverse translation of rotation axis

Fig. 4.1.5 : Rotation about a line parallel to X-axis

– The same concept can be extended for rotation about a line parallel to Y and Z axis.

4.1.3.3 Rotation about an Arbitrary Line

Q. Derive transformation matrix for rotation about an arbitrary axis is space.

– In real world application, rotation about principal axis is rare case. Application programmer needs to rotate the object around any arbitrary axis in space.

– When the object is to be rotated around a line which is not parallel to the principal axis, it requires more efforts. This case is more complicated than previous cases.

Desired rotation is achieved by composite transformation. Following steps should be performed for that :

Step 1 : Translate the rotation axis such that it passes through the origin.

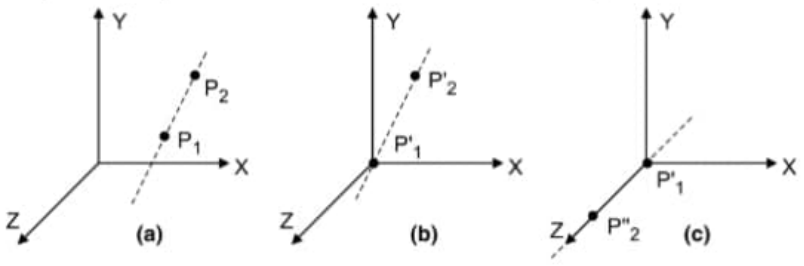
Step 2 : Rotate the translation axis such that it coincides one of the principal axes.

Step 3 : Perform the actual rotation operation.

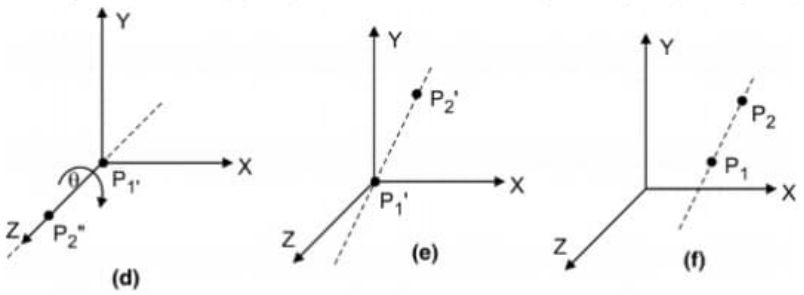
Step 4 : Perform inverse rotation to bring rotation axis to the original orientation.

Step 5 : Perform inverse translation to move back the rotation axis to its original location.

– The sequence of the operation to perform desired rotation is described in Fig. 4.1.6.



(a) Initial Configuration (b) Step 1 : Translate P_1 to the origin (c) Step 2 : Align $P_1'P_2'$ to Z axis



(d) Step 3 : Rotation about Z-axis by given angle θ (e) Step 4 : Rotate the axis to the original orientation (f) Step 5 : Translate the axis to the original location

Fig. 4.1.6 : Rotation about an arbitrary line

- We can align the rotation axis with any of the principal axes. For sake of discussion, we will align it with the Z axis.

  Let $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ be the endpoints of the rotation axis depicted in Fig. 4.1.7.

  Direction of rotation axis P_1P_2 is given by :

$$\overrightarrow{P_1P_2} = P_2 - P_1 = V = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

  Unit vector in direction of rotation axis is defined as :

$$u = \frac{V}{|V|}$$

$$= \frac{x_2 - x_1, y_2 - y_1, z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}$$

$$= (a, b, c)$$

  Where, $a = \frac{x_2 - x_1}{|V|}$, $b = \frac{y_2 - y_1}{|V|}$, $c = \frac{z_2 - z_1}{|V|}$

$$|V| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

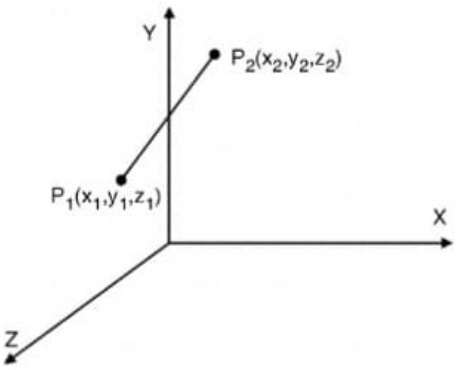


Fig. 4.1.7 : Orientation of rotation axis P_1P_2

- If the rotation axis is not passing through the origin, we should translate any of the endpoints to origin. Omit this step if rotation axis is already passing through the origin.
- For the given case, let us translate the point P_1 to the origin. This is done by applying the following translation matrix to the endpoints of the rotation axis.

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

- This transformation moves the rotation axis as shown in Fig. 4.1.8, P_1' and P_2' are the translated coordinates of original endpoints. Length of rotation axis is already normalized to simplify the mathematical computation. So, normalized coordinates of u would be (a, b, c) as computed before.

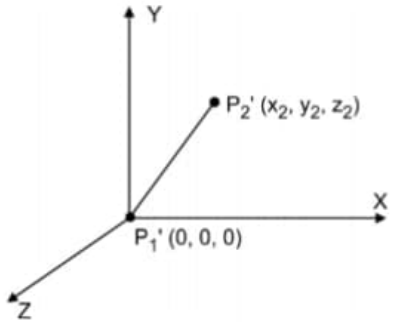


Fig. 4.1.8 : Position of rotation axis after translation by point P_1

- Now align the rotation axis with the Z axis, to do so rotate the vector u around the X axis in anticlockwise direction by the angleα such that unit vector falls in XZ plane.
- Then rotate unit vector in XZ plane around the Y axis in clockwise direction by the angle β to align it with the Z axis.
- To align the vector u with Z-axis, first put the torch on X-axis such that the shadow of vector u falls in YZ plane, call the shadow u_1.
- As this would be parallel projection on YZ plane, x coordinate will become 0 and remaining two coordinate will not alter.
- As shown in Fig. 4.1.9, parallel projection of vector u on YZ plane would be u'(0, b, c). Parallel projection on YZ plane only eliminates the x coordinate; it does not affect remaining coordinates. It is easier to find out angle α with projected vector, compared to u.

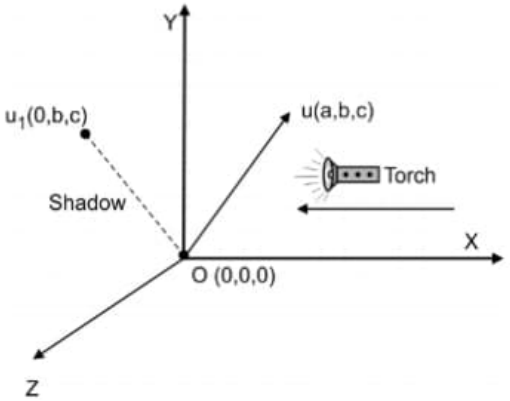


Fig. 4.1.9 : Shadow of vector u on YZ plane

- Vector u_1 makes an angle α with the Z axis. If we rotate u and u_1 by the angleα, u' will align with Z axis and u will fall in XZ plane as shown in Fig. 4.1.10.
- From Fig. 4.1.10, sin and cosine angles between $u_1$and Z axis is computed using trigonometric rule as,

$$\sin\alpha = \frac{\text{Opposite side}}{\text{Hypotenuse}} = \frac{b}{d}$$

$$\cos\alpha = \frac{\text{Adjacent side}}{\text{Hypotenuse}} = \frac{c}{d}$$

$$d = \sqrt{b^2 + c^2}$$

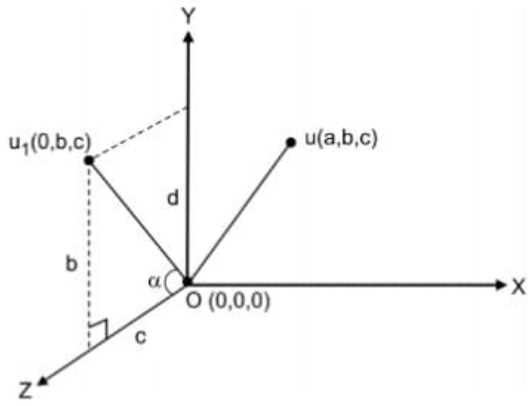


Fig. 4.1.10 : Computation of angle α from projected vector u_1

The rotation matrix around the X axis would be,

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- This transformation matrix moves unit vector u from space to XZ 2D plane as shown in Fig. 4.1.11. Here, u' is the vector after rotation of u around the X axis by angle α.
- As rotation was performed about X-axis, the x component of u won't change. And u' is in XZ plane, so its y component is zero, and z component is d, we already computed. Rotation does not alter the length and hence the length of vector u' will remain 1.

$$l = \sqrt{a^2 + (b^2 + c^2)} = 1$$

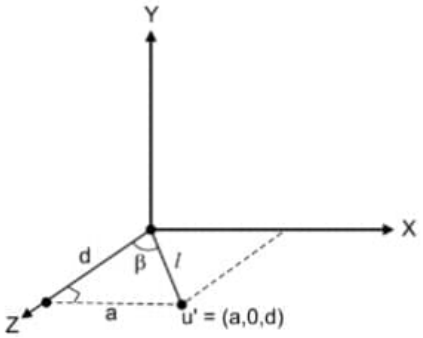


Fig. 4.1.11 : Position of vector u after rotation about X-axis by angle α

- If we rotate u' by the angle β about Y-axis in clockise direction, it will be aligned with the Z axis.
- From Fig. 4.1.11, sin and cosine angles between u' and Z axis is computed using trigonometric rule as,

$$\sin\beta = \frac{\text{Oppositeside}}{\text{Hypotenuse}} = \frac{a}{l} = a$$

$$\cos\beta = \frac{\text{Adjacentside}}{\text{Hypotenuse}} = \frac{d}{l} = d$$

- So transformation matrix for rotation about Y-axis would be,

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Above three steps, translation, rotation about X-axis and rotation about Y-axis would align line P_1P_2 to the Z axis.
- The actual rotation about line P_1P_2 by the angle θ is achieved by rotation about Z-axis using following rotation matrix.

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

– To move the rotation axis to its actual position, perform the first three steps in reverse order. The composite transformation matrix would be,

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & \frac{-b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, $R_y(\beta) \cdot R_x(\alpha) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & \frac{-ab}{d} & \frac{-ac}{d} & 0 \\ 0 & c/d & \frac{-b}{d} & 0 \\ a & b & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_v$

And $R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & \frac{-b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & a & 0 \\ \frac{-ab}{d} & c/d & b & 0 \\ \frac{-ac}{d} & \frac{-b}{d} & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_v^T$

$$\therefore \quad R(\theta) = T^{-1} \cdot A_v^T(\beta) \cdot R_z(\theta) \cdot A_v \cdot T$$

This is the desired transformation sequence.

Note : If rotation axis is passing from origin, then translation and inverse translation steps are not required.

Assumption : Length of rotation axis is normalized, i.e. $l = |v| = 1$.

4.1.4 Solved Examples

Example 4.1.1 : A cube is defined by 8 vertices: A(0, 0, 0), B(2, 0, 0), C(2, 2, 0), D(0, 2, 0), E(0, 0, 2), F(0, 2, 2), G(2,0,2), H(2, 2, 2). Perform following 3D transformations on the cube.

(i) Translation by T = $[t_x, t_y, t_z]$ = [5 34]

(ii) Scaling by S = $[s_x, s_y, s_z]$ = [1 2 0.5]

(iii) Rotation about X-axis by 90 in clockwise direction

Solution :

Given cube is shown in Fig. P. 4.1.1.

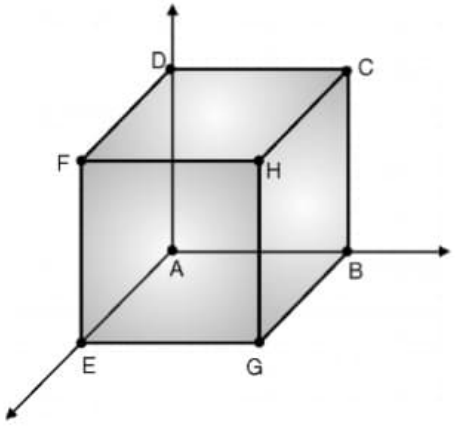


Fig. P. 4.1.1 : 2 × 2 × 2 Cube with A(0, 0, 0) at origin

Let us compute the coordinates for asked operations.

1. Translation by T = [5 3 4]

Coordinates of translated cube are computed as,

$$P' = T \cdot P$$

$$= \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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

$$P = \begin{bmatrix} 5 & 7 & 7 & 5 & 5 & 5 & 7 & 7 \\ 3 & 3 & 5 & 5 & 3 & 5 & 3 & 5 \\ 4 & 4 & 4 & 4 & 6 & 6 & 6 & 6 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

2. Scaling by S = [1 2 0.5]

Coordinates of Scaled cube are computed as,

$$P' = S \cdot P$$

$$= \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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

$$P = \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 4 & 4 & 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3. Rotation about X-axis by 90° in clockwise direction

Here, rotation is in clockwise direction, so $\theta = -90°$

Coordinates of Scaled cube are computed as,

$$P' = R_{x(\theta = -90^0)} \cdot P$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-90°) & -\sin(-90°) & 0 \\ 0 & \sin\theta(-90°) & \cos(-90°) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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

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

Example 4.1.2 : Find a sequence of transformations required to rotate a solid object w.r.t. a line y = mx + C in anti-clockwise direction by angle θ.

Solution :

Step 1 : Line y = mx + c is shown in Fig. P. 4.1.2. Here, c is the y-intercept translating the line by the amount [0 – c 0].

We can make the rotation axis pass from the origin by following translation operation.

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

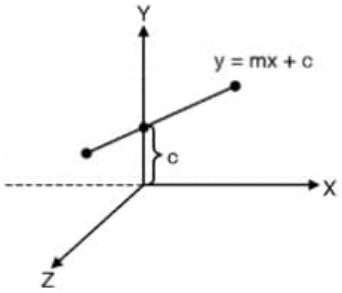


Fig. P. 4.1.2

Step 2 : Rotate line about X-axis by angle α in anticlockwise direction such that it falls in XZ plane.

$$\therefore R_{x(\alpha)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3 : Now rotate line about Y-axis by the angle β in a clockwise direction. Such that it gets aligned with the Z axis.

$$\therefore R_{y(\beta)} = \begin{bmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 4 : Now line y = mx + c is aligned with z-axis,

To perform actual rotation about z-axis by angle θ

$$\therefore R_{z(\theta)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 5 : Inverse rotation about Y axis by angle β

$$\therefore R_{y(\beta)}^{-1} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 6 : Inverse rotation about X-axis by the angle α to put a line in its original orientation.

$$\therefore R_{x(\alpha)}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 7 : Inverse translation of line by vector

[0, c, 0] to place it on its actual position

$$\therefore\ T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & c \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

∴ Desired transformation sequence to rotate object about line y = mx + c by angle θ is,

$$M = T^{-1} \cdot R_{x(\alpha)}^{-1} \cdot R_{y(\beta)}^{-1} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)} \cdot T$$

Example 4.1.3 : Rotate a triangle ABC with vertices A (2, 2, 2), B (3, 4, 7) and C (8, 9, 12) about Y-axis in anticlockwise direction by angle 90°.

Solution :

The rotation angle is 90° and rotation is to be performed in a anticlockwise direction, so θ = + 90°. Rotation is to be performed about the Y-axis, so transformed coordinates are computed as,

$$P' = R_y(\theta) \cdot P$$

$$= \begin{bmatrix} \cos(90^\circ) & 0 & \sin(90^\circ) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(90^\circ) & 0 & \cos(90^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 7 & 12 \\ 2 & 4 & 9 \\ -2 & -3 & -8 \end{bmatrix}$$

Rotated vertices are A′ (2, 2, – 2), B′ (7, 4, – 3) and C′ (12, 9, – 8).

Original Coordinates	Transformed Coordinates
A(2, 2, 2)	A′(2, 2, – 2)
B(3, 4, 7)	B′(7, 4, – 3)
C(8, 9, 12)	C′(12, 9, – 8)

Example 4.1.4 : Find transformation matrix for rotation about a line parallel to the X-axis and passing through the reference point P (x_r, y_r, z_r).

Solution :

Mentioned composite transformation is achieved by performing sequence of following steps :

1. Translation by $T_{(-xr, -yr, -zr)}$
2. Rotation about X-axis by angle θ.
3. Inverse translation.

$$\therefore\ M = T^{-1} \cdot R_x(\theta) \cdot T$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & 1 & 0 & -y_r \\ 0 & 0 & 1 & -z_r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

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

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & z_r\sin\theta + y_r(1-\cos\theta) \\ 0 & \sin\theta & \cos\theta & z_r(1-\cos\theta) - y_r\sin\theta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example 4.1.5 : The pyramid with co-ordinates A (0, 0, 0), B (1, 0, 0), C (0, 1, 0) and D (0, 0, 1) is to be rotated by about line L that has direction vector v = j + k and passing through point (0, 1, 0). Find the coordinates of the transformed pyramid.

Solution :

The orientation of said pyramid is shown in Fig. P. 4.1.5.

To rotate the pyramid about line L, we need to align it with one of the principal axes. Let us align it with Z-axis.

We should perform the following steps to achieve the desired operation.

1. Translate line by a vector (0, – 1, 0) so that it passes through the origin.

$$\therefore \quad T = [0 \; -1 \; 0]$$

2. Rotate this translated line by angle α about X-axis in anticlockwise direction, so that it gets align with Z-axis.

   Line L is in direction of vector v = j + k, so it makes an angle of $\alpha = 45°$ with Y and Z axis and the line is already in YZ plane. So only one rotation is sufficient to align the line with the Z axis.

3. Perform rotation about Z-axis by angle $\theta = 90°$.
4. Inverse rotation by angle α.
5. Inverse translation by vector T.

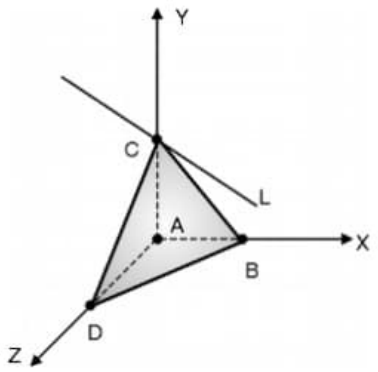


Fig. P. 4.1.5

$$M = T^{-1} \cdot R_{x(-\alpha)} \cdot R_z(\theta) \cdot R_{x(\alpha)} \cdot T$$

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

Multiplying first two and last two matrices,

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

Multiplying last two matrices,

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

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

Transformed coordinates are computed as,

$$P' = M \cdot P$$

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

$$\therefore P' = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \frac{2}{\sqrt{2}} \\ \frac{-1}{2} & \frac{\sqrt{2}-1}{2} & 0 & 0 \\ \frac{-1}{2} & -\left(\frac{\sqrt{2}+1}{2}\right) & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	$A'\left(\frac{1}{\sqrt{2}}, \frac{-1}{2}, \frac{-1}{2}\right)$
B(1, 0, 0)	$B'\left(\frac{1}{\sqrt{2}}, \frac{\sqrt{2}-1}{2}, -\frac{\sqrt{2}+1}{2}\right)$
C(0, 1, 0)	C'(0, 0, 0)
D(0, 0, 1)	$D'\left(\frac{2}{\sqrt{2}}, 0, 0\right)$

Example 4.1.6 : Find out the 3D transformation matrix to rotate a given 3D object by an amount 60° about a line passing from point(1,1,1) and the direction vector V = 2i + 2j + 2k.

Solution :

Here $\theta = 60°$

Line is passing from (1, 1, 1) and its direction vector is V = 2i + 2j + 2k. So line interpolates all points having same x, y and z components hence line also passes through origin.

$$V = 2i + 2j + 2k = i + j + k = ni + nj + nk$$

$$\text{So, } (a, b, c) = (1, 1, 1)$$

As derived in previous section, $d = \sqrt{b^2 + c^2} = \sqrt{1 + 1} = \sqrt{2}$

And Length of the vector, $l = |v| = \sqrt{a^2 + b^2 + c^2} = \sqrt{1 + 1 + 1} = \sqrt{3}$

Transformation matrix to rotate object about arbitrary line is given as,

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)}$$

$$= Av^{-1} \cdot R_{z(\theta)} \cdot Av$$

Length of vector V is not normalized, so

$$R_{x(\beta)} \cdot R_{x(\alpha)} = A_v = \begin{bmatrix} \frac{d}{|v|} & \frac{-ab}{(d \cdot |v|)} & \frac{-ac}{(d \cdot |v|)} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ \frac{a}{|v|} & \frac{b}{|v|} & \frac{c}{|v|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} = A^T_v = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{z(\theta=60°)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{3} & -\sqrt{3} & 0 & 0 \\ \sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = A_v^T \cdot R_{z(\theta=60°)} \cdot A_v$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{3} & -\sqrt{3} & 0 & 0 \\ \sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\cdot \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices,

$$= \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{2} & \frac{-1-\sqrt{3}}{\sqrt{2}} & \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ \sqrt{2} & \frac{\sqrt{3}-1}{\sqrt{2}} & \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} \frac{2\sqrt{3}+1}{3} & -\frac{\sqrt{3}-2}{3} & \frac{4-\sqrt{3}}{3} & 0 \\ \frac{4-\sqrt{3}}{3} & \frac{\sqrt{3}+3}{6} & \frac{7\sqrt{3}-7}{6} & 0 \\ -\frac{\sqrt{3}-2}{3} & \frac{11-5\sqrt{3}}{6} & -\frac{4-2\sqrt{3}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Example 4.1.7 : Rotate a triangle with vertices A(0, 2, 1), B(2, 3, 0) and C(1, 2, 1) by 45° around a line passing through the points (0, 0 0) and (1, 1, 1).

Solution :

As derived in previous section, $d = \sqrt{b^2 + c^2} = \sqrt{(0-1)^2 + (0-1)^2}\sqrt{1+1} = \sqrt{2}$

And Length of the rotation axis, $l = |v| = \sqrt{a^2 + b^2 + c^2} = \sqrt{(0-1)^2 + (0-1)^2 + (0-1)^2} = \sqrt{1+1+1+} = \sqrt{3}$

Rotation axis is already passing through origin, so there is no need of translating it. So the desired transformation sequence can be achieved by,

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)}$$

$$= Av^{-1} \cdot R_{z(\theta)} \cdot Av$$

Where,

$$Av = \begin{bmatrix} \frac{d}{|v|} & \frac{-ab}{(d \cdot |v|)} & \frac{-ac}{(d \cdot |v|)} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ \frac{a}{|v|} & \frac{b}{|v|} & \frac{c}{|v|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{vmatrix} \frac{\sqrt{2}}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$R_{z(\theta = 45^\circ)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) & 0 & 0 \\ \sin(45^\circ) & \cos(45^\circ) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$M = Av^{-1} \cdot R_{z(\theta)} \cdot Av = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 0.815 & 0 & 0.578 & 0 \\ -0.408 & 0.709 & 0.578 & 0 \\ -408 & -0.709 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.709 & -0.709 & 0 & 0 \\ 0.709 & 0.709 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.815 & -0.408 & 0.408 & 0 \\ 0 & 0.709 & 0.709 & 0 \\ 0.578 & 0.578 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices,

$$M = \begin{bmatrix} 0.815 & 0 & 0.578 & 0 \\ -0.408 & 0.709 & 0.578 & 0 \\ -408 & -0.709 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.578 & -0.792 & 0.213 & 0 \\ 0.578 & 0.213 & -0.792 & 0 \\ 0.578 & 0.578 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 0.805 & -0.311 & 0.508 & 0 \\ 0.508 & 0.808 & -0.314 & -0.312 \\ 0.506 & 0.809 & 0 & 0 \\ 0 & 0 & 1 & \end{bmatrix}$$

Transformed coordinates, $P' = M \cdot P$

$$= \begin{bmatrix} 0.805 & -0.311 & 0.508 & 0 \\ 0.508 & 0.808 & -0.0314 & 0 \\ -0.312 & 0.506 & 0.809 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -0.114 & 0.677 & 0.691 \\ 1.302 & 3.440 & 1.810 \\ 1.821 & 0.894 & 1.509 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(0, 2, 1)	A' (-0.114, 1.302, 1.821)
B(2, 3, 0)	B' (0.677, 3.440, 0.894)
C(1, 2, 1)	C' (0.691, 1.810, 1.509)

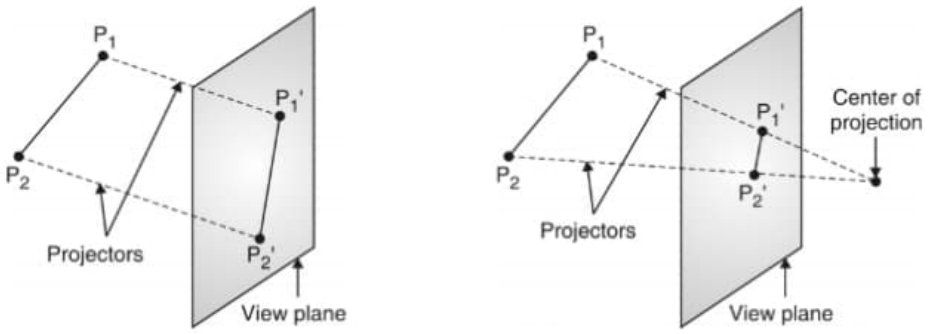
4.2 Types of Projection

4.2.1 Introduction to Projection

Q. What is projection? Why projection is necessary?

Q. Define the terms: Projection, view plane, Direction of projection, center of projection.

- **Projection** is the process of transforming an object representation from n-dimensional space to less than n-dimensional space.
- A 3D object to 2D projection is a vital operation in CAD-CAM and engineering.
- **View plane** is the plane on which the object is projected. It is also known as a **projection plane** or plane of projection.
- Projection is broadly classified into two categories : Parallel projection and perspective projection.



(a) Parallel Projection **(b) Perspective Projection**

Fig. 4.2.1 : Types of projection

- In parallel projection, coordinate positions of the object are transformed to view plane by parallel rays/projectors.
- In perspective projection, rays are fired from a point source, called center of projection, which intersects the object coordinates and projects it on view plane.

- Fig. 4.2.1 demonstrates parallel and perspective projections. P_1P_2 is the original line in space. $P_1'P_2'$ is projected line on 2D view plane. In the case of parallel projection, projectors are parallel to each other, whereas in case of perspective projection rays are inclined and meeting at a point called **center of projection**.

Types of Projections :

Q. Enlist the types of projection.

We can summarize types of projections as follows :

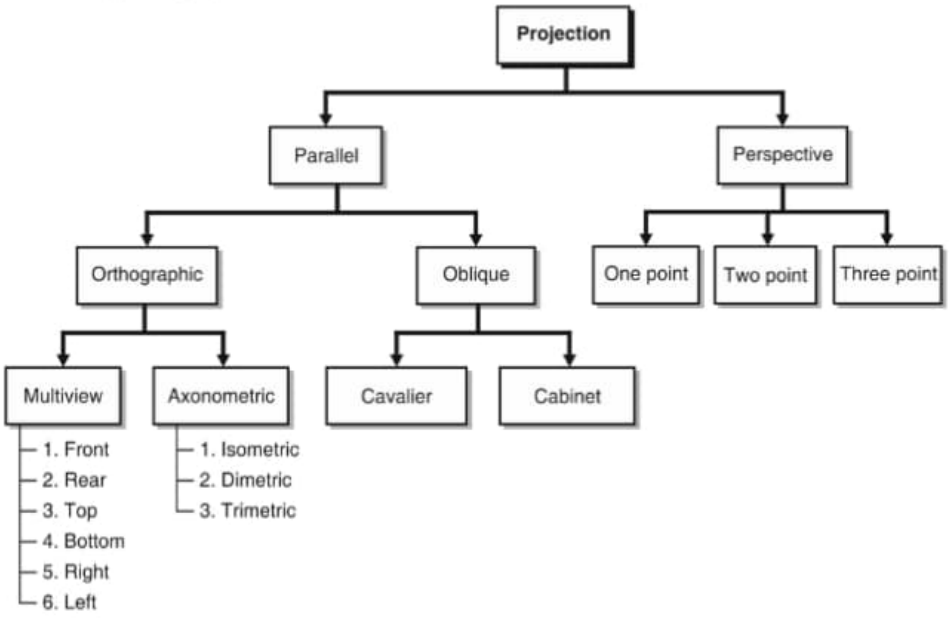


Fig. 4.2.2 : Taxonomy of projection

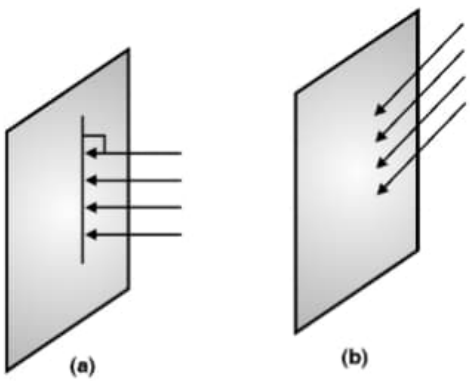
4.2.2 Parallel Projection

W-18

Q. Explain types of Parallel Projection with example. **(W-18, 4 Marks)**

Q. Write a short note on parallel projection.

- Parallel projection is achieved by passing parallel rays from the object vertices and projecting the object on view plane.
- In parallel projection, all projection vectors are parallel to each other.



(a) Orthographic parallel projection **(b) Oblique parallel projection**

Fig. 4.2.3 : Types of parallel projection

- Parallel projection preserves true shape and size of the object on view plane.
- However, it fails to capture depth information and hence cannot produce a realistic view.
- Based on the direction of projection vector and their relation with view plane, parallel projection is further classified as an orthographic parallel projection or oblique parallel projection.
- When all projectors are parallel to each other and perpendicular to the view plane, it is called **orthographic parallel projection**. If projectors are parallel to each other but are not perpendicular to the view plane, it is called **oblique parallel projection**. Both types of projections are illustrated in Fig. 4.2.3.
- Classification of parallel projection is depicted in Fig. 4.2.4.

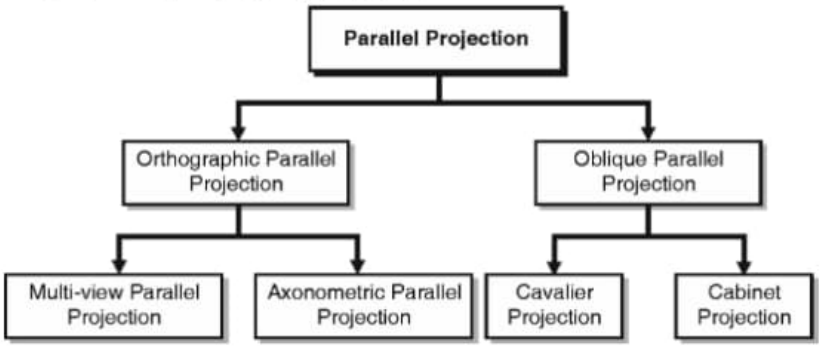


Fig. 4.2.4 : Types of parallel projection

4.2.2.1 Orthographic Parallel Projection

Q. Explain orthographic parallel protection.

1. Multi-view Parallel projection :

- Orthographic projection is used in engineering drawings. It is most often used to generate the front view, side view and top view.
- It is possible to construct the 3D view of an object from these three views.
- In **multi-view parallel projection**, the plane of projection is parallel to the surface of the object. We get only one surface projected on view plane in multi-view projection.
- The front view is called **elevation** and the top view is called **plan**. Generally, the most descriptive face of an object is selected for elevation. Sometimes, cross-sectional view is also used to get more details.

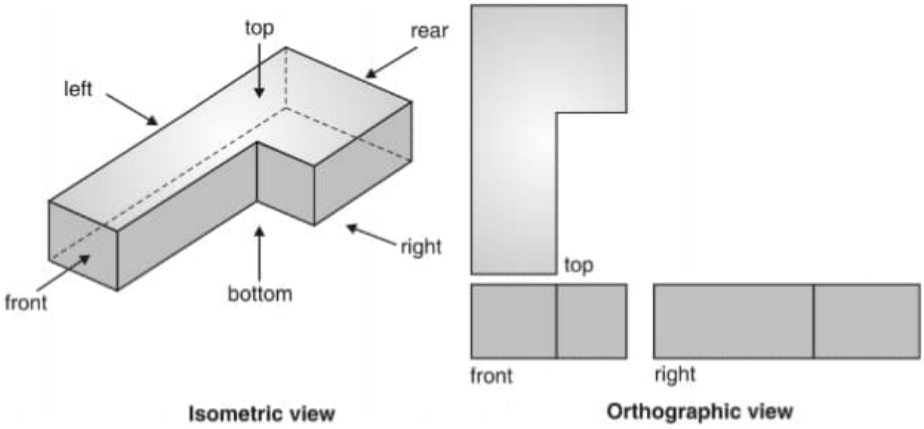


Fig. 4.2.5 : Multi-view parallel projection

- On paper, the top view is placed exactly above elevation and left or right side views are kept on the respective side of the front view.
- Orthographic parallel projection preserves length and angle between lines, so used to measure the actual dimension of an object.
- Fig. 4.2.5 shows the isometric 3D view and its orthographic parallel projection with front, top and right side view.

2. Axonometric Parallel projection :

- As shown in Fig. 4.2.6, by performing rotation of an object, we can display more than one surfaces of an object. Such projection is called **axonometric orthographic projection**.
- Axonometric projection is further classified in isometric, diametric or trimetric by aligning projection plane with three, two or one principal axis respectively.
- Axonometric projection does not preserve the true size of the object. The ratio of projected length to original length is called **foreshortening factor.**
- Isometric projection is achieved by aligning projection vector with the cube diagonal. It preserves the same foreshortening factor for all three axes. For diametric projection, foreshortening factors for any two axes are same. And for trimetric projection, foreshortening factors for all three principal axes are different.
- Thus, diametric is a special case of trimetric and isometric is a special case of diametric.

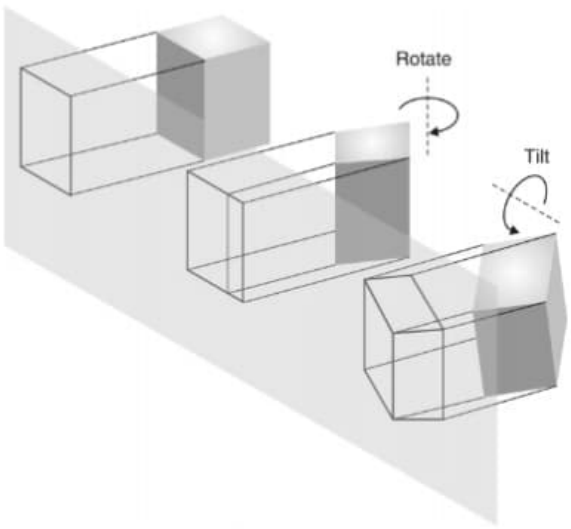


Fig. 4.2.6 : Displaying three surfaces by rotation and tilting

- Fig. 4.2.7 explains all three cases of axonometric projection. If diagonal of a unit cube is aligned such that it makes an equal angle with all three axes, projection foreshortens the line with the same factor in all the directions. This is termed as isometric projection.

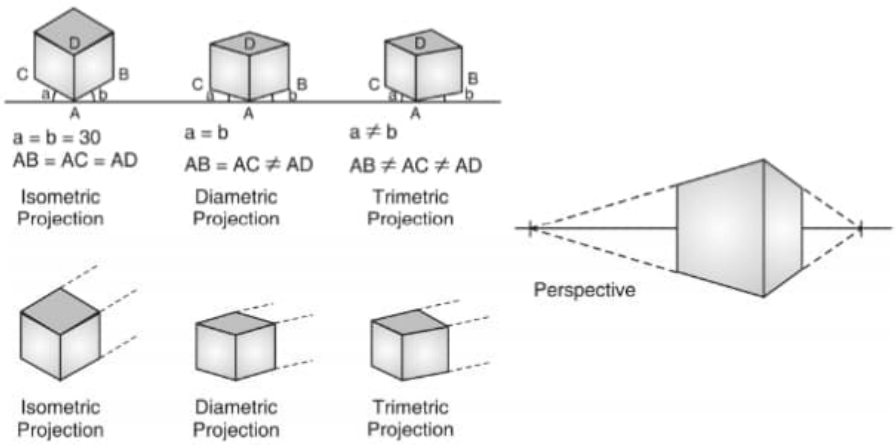


Fig. 4.2.7 : Types of axonometric projection

- In diametric projection, any two sides of cube make equal angle and foreshortening in those two directions would be the same.

- In trimetric projection, all three sides of cube make different angle and their foreshortening factors are also different.
- If the projection plane is placed at Z_{vp} distance on the Z axis and parallel to XY plane, parallel projection gives the coordinates,

$$x_p = x$$
$$y_p = y$$

- Original z coordinate value is preserved for depth cueing and for visible surface determination.

4.2.2.2 Oblique Parallel Projection

Q. Explain in detail Oblique projection.

As we discussed earlier, in **oblique projections**, projectors are parallel to each other and they are not perpendicular to the view plane. View in oblique projection is controlled by two parameters ϕ and α as shown in Fig. 4.2.8.

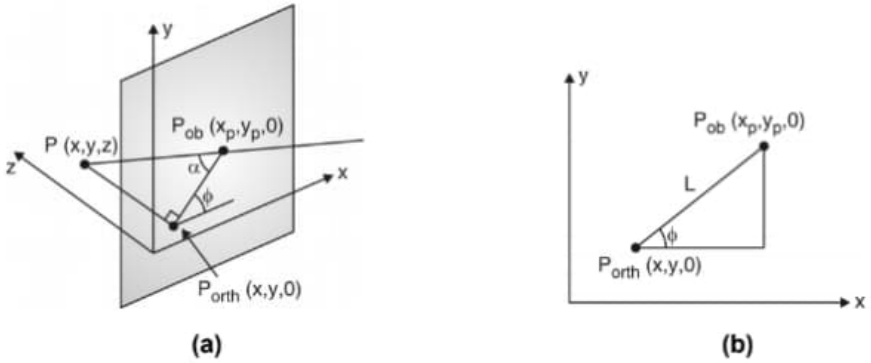


Fig. 4.2.8 : Oblique parallel projection

Let, P(x, y, z) be the point in space. Orthographic projection of point P on view plane XY is P_{orth}(x, y, 0).

Oblique projection of P on the same plane is let's say P_{ob}(xp, yp, 0).

Oblique projection vector passing through points P and P_{ob} makes an angle α with view plane. Let the length of line joining points P_{orth} and P_{ob} is L.

From Fig. 4.2.8(b), values of oblique projection of point P(x, y, z) on XY view plane is given by following equations :

$$x_p = x + L\cos\phi$$
$$y_p = y + L\sin\phi$$

From Fig. 4.2.8 (a),

$$\tan\alpha = \frac{z}{L}$$

$$\therefore L = \frac{z}{\tan\alpha} = zL_1$$

By solving the equations of x_p and y_p for L,

$$x_p = x + zL_1\cos\phi$$
$$y_p = y + zL_1\sin\phi$$

So the transformation matrix for any parallel projection on view plane X_vY_v is written as,

$$M = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

L_1 is the foreshortening factor.

Properties :

- α is the angle between the oblique projector and plane.
- ϕ is the angle between horizontal and projected Z axis.
- $L_1 = 1$ when $\alpha = 90°$, which produces orthographic projection.
- ϕ is a free parameter, but the common choice for ϕ are 30° and 45°.
- Common choice of α are 45° ($\tan\alpha = 1$) and 63.4° ($\tan\alpha = 2$).
- According to the value of α, oblique projection is classified into two categories:
  - Cavalier projection
  - Cabinet projection

1. Cavalier Projection :

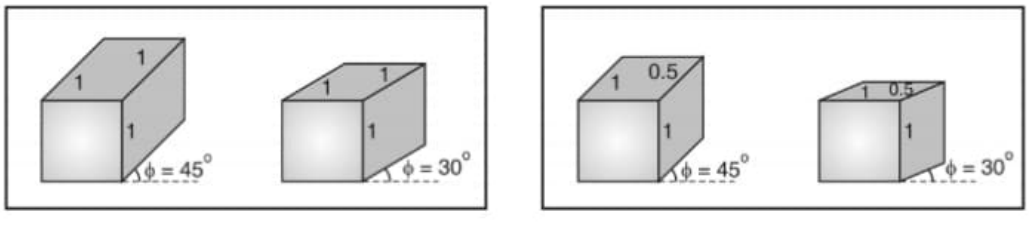
Q. Write a short note on cavalier protection.

- If $\alpha = 45°$(angle between projectors and view plane), the projection is called **cavalier projection**.
- Foreshortening factor for the lines perpendicular to the projection plane would be 1.
- In other words, if the projection angle is 45°, there won't be any change in the projected length of lines perpendicular to the view plane.
- Resulting figure in cavalier projection appears thick. 3D nature of the object can be captured but the shape seems to be distorted for lines not parallel to the principal axis.

2. Cabinet Projection :

Q. Write a short note on cabinet protection.

- If α is set to 63.4°, the type of projection is known as **cabinet projection**.
- Foreshortening factor for the lines perpendicular to the projection plane would be 0.5.
- In other words, if the projection angle is 63.4°, the projected length of lines perpendicular to view plane would be half of the original length.
- Cabinet projection is more realistic than cavalier projection.
- Fig. 4.2.9 shows the effect of angle for cabinet and cavalier projection.



(a) Cavalier projection **(b) Cabinet projection**

Fig. 4.2.9 : Cabinet and cavalier projection

The relation between angle, projection factor and type of projection is summarized in Table 4.2.1.

Table 4.2.1

The angle between projector and view plane(α)	Foreshortening factor	Type of projection
45.0°	1.0	Cavalier
63.4°	0.5	Cabinet
90.0°	0.0	Orthographic

Example 4.2.1 : Derive the transformation matrix for parallel projection of point P (x, y, z) on XY plane in direction $V = a_i + b_j + c_k$.

Solution :

$V = a_i + b_j + c_k$ is the direction of projection. Let us assume that parallel projection of point P (x, y, z) on XY plane in direction of V is $P'(x', y', z')$.

Projection is in XY plane, so $z' = 0$.

Directions of V and PP' are same. So,

$$\overrightarrow{PP'} = k \cdot \vec{V}$$

$$((x' - x), (y' - y), (z' - z)) = k \cdot (a, b, c)$$

$$x' = x + k \cdot a$$

$$y' = y + k \cdot b$$

$$z' = z + k \cdot c$$

but, $z' = 0$

$$\therefore z + k \cdot c = 0$$

$$\therefore k = -\frac{z}{c}$$

$$\therefore x' = x - \frac{a}{c} \cdot z$$

$$\therefore y' = y - \frac{b}{c} \cdot z$$

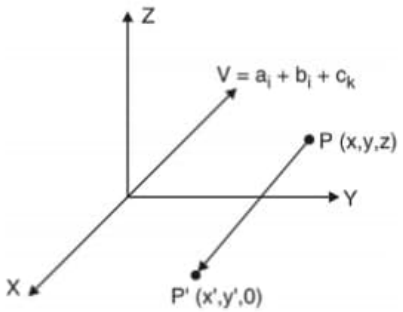


Fig. P. 4.2.1

Transformation matrix for this projection would be,

$$M = \begin{bmatrix} 1 & 0 & -a/c & 0 \\ 0 & 1 & -b/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix representation of transformed coordinates is given as,

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a/c & 0 \\ 0 & 1 & -b/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Example 4.2.2 : Find transformed co-ordinate of unit cube with one vertex at origin for

(i) Cavalier projection with $\phi = 45°$

(ii) Cabinet projection with $\phi = 30°$

Solution :

Let us assume that vertex A of unit cube is positional at the origin.

From Fig. P. 4.2.2,

$$A = (0, 0, 0)$$
$$B = (1, 0, 0)$$
$$C = (1, 1, 0)$$
$$D = (0, 1, 0)$$
$$E = (0, 0, 1)$$
$$F = (1, 0, 1)$$
$$G = (1, 1, 1)$$
$$H = (0, 1, 1)$$

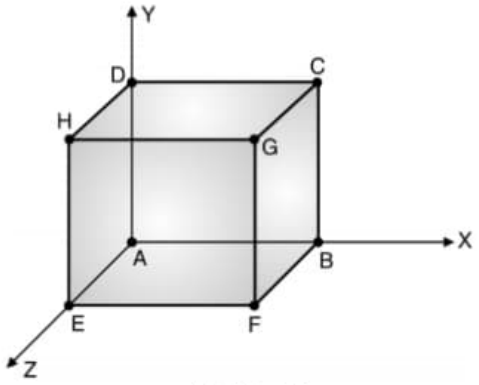


Fig. P. 4.2.2

(i) Cavalier projection :

For cavalier projection, foreshortening factor $L_1 = 1$

Transformation matrix for cavalier and cabinet projection is,

$$M = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{With } \phi = 45° \text{ and } L_1 = 1$$

$$M = \begin{bmatrix} 1 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformed points are computed as,

$$P' = M \cdot P$$

$$M = \begin{bmatrix} 1 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

By solving above matrices,

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	A'(0, 0, 0)
B(1, 0, 0)	B'(1, 0, 0)
C(1, 1, 0)	C'(1, 1, 0)
D(0, 1, 0)	D'(0, 1, 0)
E(0, 0, 1)	$E'\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)$
F(1, 0, 1)	$F'\left(\frac{\sqrt{2}+1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)$
G(1, 1, 1)	$G'\left(\frac{\sqrt{2}+1}{\sqrt{2}}, \frac{\sqrt{2}+1}{\sqrt{2}}, 0\right)$
H(0, 1, 1)	$H'\left(\frac{1}{\sqrt{2}}, \frac{\sqrt{2}+1}{\sqrt{2}}, 0\right)$

(ii) Cabinet projection :

For cabinet projection, foreshortening factor, $f = L_1 = \frac{1}{2}$

Given angle $\phi = 30°$

So,
$$M = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{\sqrt{3}}{4} & 0 \\ 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformed co-ordinates of the cubeare computed as,

$$P' = M \cdot P$$

$$= \begin{bmatrix} 1 & 0 & \frac{\sqrt{3}}{4} & 0 \\ 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

By solving the above matrices,

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	A'(0, 0, 0)
B(1, 0, 0)	B'(1, 0, 0)
C(1, 1, 0)	C'(1, 1, 0)
D(0, 1, 0)	D'(0, 1, 0)
E(0, 0, 1)	$E'\left(\frac{\sqrt{3}}{4}, \frac{1}{4}, 0\right)$
F(1, 0, 1)	$F'\left(\frac{4+\sqrt{3}}{4}, \frac{1}{4}, 0\right)$
G(1, 1, 1)	$G'\left(\frac{4+\sqrt{3}}{4}, \frac{5}{4}, 0\right)$
H(0, 1, 1)	$H'\left(\frac{\sqrt{3}}{4}, \frac{5}{4}, 0\right)$

4.2.3 Perspective Projection

Q. Write a short note on perspective projection.

Q. Derive a perspective projection of point P (x, y, z) on a view plane positioned at z = 0 and center of projection is on negative z-axis at distance d.

- Perspective projection preserves the depth information but it does not preserve true shape and size of the object. Size of the projected object depends on the distance between view plane, the center of projection and the object.
- Projection of a distant object is smaller than that of the nearer object. Fig. 4.2.10 describes this property of perspective projection.

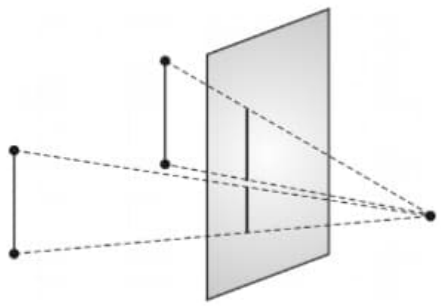


Fig. 4.2.10 : Effect of perspective projection

Let us derive the transformation matrix for perspective projection of point P(x, y, z) on view plane Z = 0 (i.e. XY plane). Assume that the center of projection (COP) is at distance d from origin on Z-axis as shown in Fig. 4.2.11.

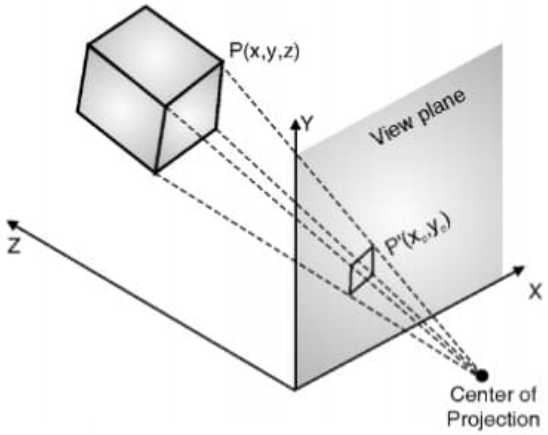


Fig. 4.2.11 : Perspective projection of P(x, y, z) on XY plane

- For this particular case, we are considering that object is on positive Z axis and centre of projection is behind the projection plane i.e. on d distance on negative Z axis. Although, this can be generalized to any valid position of object, view-plane and COP.

  If we look from the X-axis, the Fig. 4.2.11 looks as Fig. 4.2.12.

  From the triangle equality rule,

$$\frac{y_p}{d} = \frac{y}{z+d}$$

$$y_p = \frac{y \cdot d}{z+d}$$

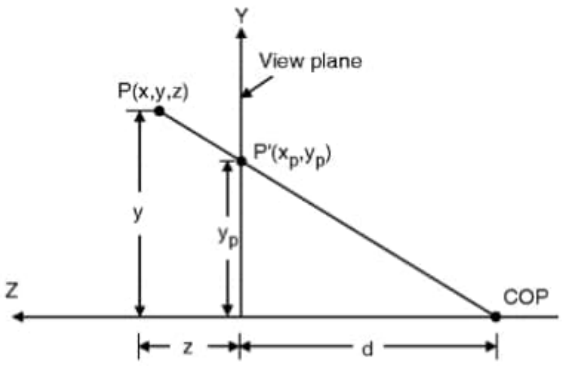


Fig. 4.2.12 : View from X axis

  If we look from the Y-axis, the Fig. 4.2.11 looks as Fig. 4.2.13.

- From the triangle equality rule,

$$\frac{x_p}{d} = \frac{x}{z+d}$$

$$\therefore x_p = \frac{x \cdot d}{z+d}$$

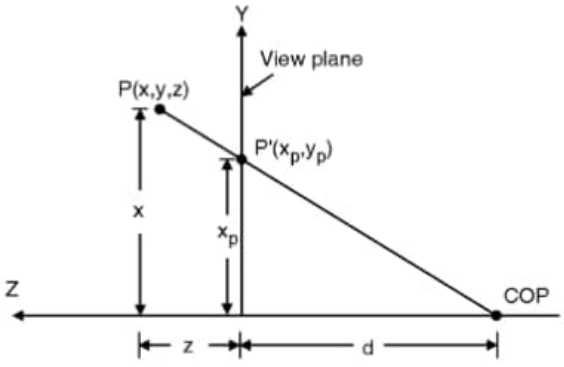


Fig. 4.2.13 : View from Y axis

- As view plane is positioned at z = 0, projected z co-ordinate would be 0.

$$z_p = 0.$$

- From this, we can define a transformation matrix as,

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

Projected Coordinates, $P' = M \cdot P$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- If view plane is not XY plane or it is not parallel to XY plane in that case, we need to perform composite transformation such that the normal of the plane get aligned with Z-axis.
- Other cases of perspective projections are discussed at the end of this section.

Types of Perspective Projection :

- In perspective projection, parallel lines of the object which are not parallel to projection plane are projected into converging point. In perspective projection, the point where projected parallel lines converge to a single point is called the **vanishing point**.
- Projection of railway track forms a vanishing point. It seems they are meeting at a single point at a very far distance. A scene can have any number of vanishing points depending on the orientation of parallel lines in the object.
- If the set of parallel lines are parallel to any of the principal axis, vanishing point for such set is called **principal vanishing point**.
- We can control the numbers of principal vanishing points by rotating the view plane. The intersection of view plane with a number of principal axes defines the number of principal vanishing points.
- If view plane intersects to one, two or three principal axes, we have a single point, two points or three-point perspective projection respectively. Fig. 4.2.14 shows the orientation of cube to achieve one, two and three-point perspective projections.
- Types of perspective projection are given based on a number of the vanishing point. There are mainly three types of perspective projections :

1. One-point perspective projection
2. Two-point perspective projection
3. Three-point perspective projection

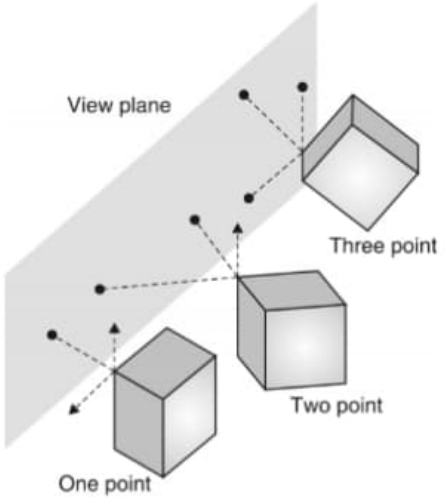


Fig. 4.2.14 : Types of perspective projection

4.2.3.1 One-point Perspective Projection

Q. Write a short note on one-point perspective projection.

- One-point perspective projection is achieved by aligning view plane with two principal axes.
- In other words, one-point perspective projection is achieved by setting up one of the principal axis perpendiculars to the view plane.

- One-point perspective projection contains only one vanishing point at the horizon. Images like straight roads, railway tracks, buildings etc. have such one-point perspective projection.
- Fig. 4.2.15 explains the one, two and three-point perspective projections.

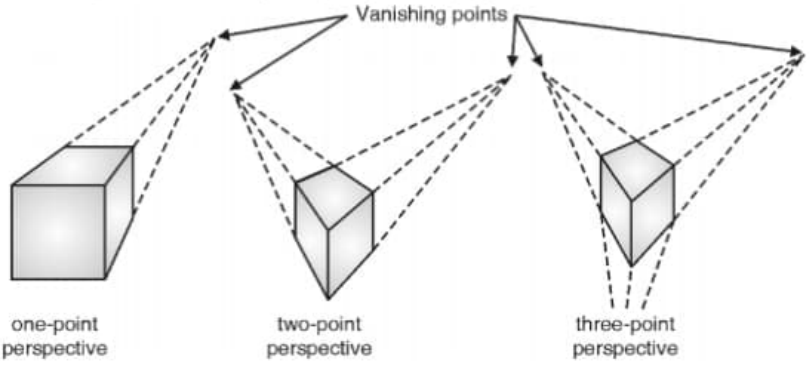


Fig. 4.2.15 : Perspective projection is achieved by projecting result of the perspective transformation to any of the principal plane

X-direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [x \;\; y \;\; z \;\; px + 1]$$

$$x' = \frac{x}{px + 1}$$

$$y' = \frac{y}{px + 1}$$

$$z' = \frac{z}{px + 1}$$

Y direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & q & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [x \;\; y \;\; z \;\; qy + 1]$$

$$x' = \frac{x}{qy + 1}$$

$$y' = \frac{y}{qy + 1}$$

$$z' = \frac{z}{qy + 1}$$

Z direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [x \;\; y \;\; z \;\; rz + 1]$$

$$x' = \frac{x}{rz+1}$$

$$y' = \frac{y}{rz+1}$$

$$z' = \frac{z}{rz+1}$$

4.2.3.2 Two-point Perspective Projection

Q. Write a short note on Two-point perspective projection.

Two-point perspective projection occurs when view plane is parallel to one of the principal axes or if view plane intersects exactly two principal axes.

- Two-point perspective projection is depicted in Fig. 4.2.15.
- When view plan intersects X and Y axis,
- X-direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [x \;\; y \;\; z \;\; (px+qy+1)]$$

$$x' = \frac{x}{px+qy+1}$$

$$y' = \frac{y}{px+qy+1}$$

$$z' = \frac{z}{px+qy+1}$$

In similar way, we can derive the value of (x', y', z') for other two cases.

4.2.3.3 Three-point Perspective Projection

Q. Write a short note on Three-point perspective projection

- Three-point perspective projection happens when view plane is not parallel to any of the principal axes
- Three-point perspective projection is depicted in Fig. 4.2.15.
- For Three-point perspective projection,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [x \;\; y \;\; z \;\; (px+qy+rz+1)]$$

$$x' = \frac{x}{px+qy+rz+1}$$

$$y' = \frac{y}{px+qy+rz+1}$$

$$z' = \frac{z}{px+qy+rz+1}$$

Example 4.2.3 : Derive a transformation matrix for perspective projection when the center of projection is at origin and plane of projection is at distance d on the Z axis.

Solution :

Spatial orientation of the view plane is shown in Fig. P. 4.2.3.

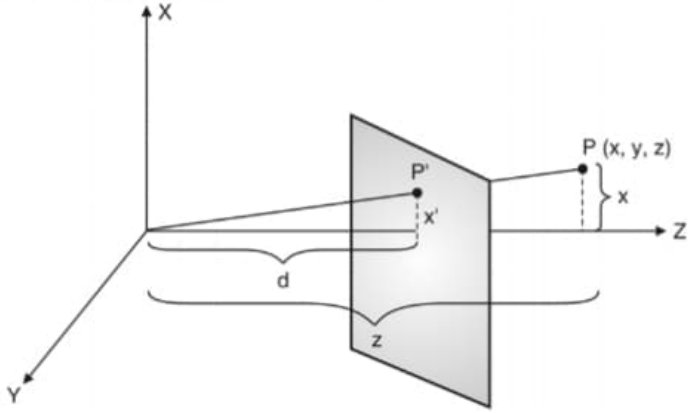


Fig. P. 4.2.3

Suppose projection of point P (x, y, z) on given view plane is P' (x', y', z').

From the rule of triangle equality,

$$\frac{x}{z} = \frac{x'}{d} \qquad \therefore x' = \frac{x}{z} \cdot d$$

Similarly, $$\frac{y}{z} = \frac{y'}{d} \qquad \therefore y' = \frac{y}{z} \cdot d$$

And z' would be d because view plane is on Z-axis at distance d.

$$\therefore z' = d$$

Transformation matrix would be,

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

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Example 4.2.4 : Using origin as the center of projection, derive transformation matrix for perspective projection on a view plane passing through reference point $R_0 (x_0, y_0, z_0)$ and having normal vector $\overline{N} = n_1 i + n_2 j + n_3 k$.

Solution :

Spatial orientation of the view plane is shown in Fig. P. 4.2.4.

Let us assume that projection of point P (x, y, z) on the given view plane is P′ (x′, y′, z′).

From Fig. P. 4.2.4, OP′ and OP both vectors have the same direction but different magnitude.

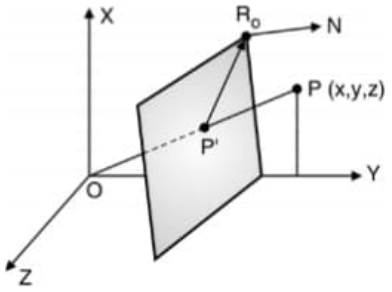


Fig. P. 4.2.4

$$\therefore \quad \overrightarrow{OP'} = \alpha \overrightarrow{OP}$$

$$(P' - O) = \alpha (P - O)$$

$$\therefore \quad (x', y', z') = \alpha (x, y, z)$$

$$x' = \alpha x \; ; \; y' = \alpha y \; ; \; z' = \alpha z$$

We need to find out α to find projected co-ordinates.

R_0 and P′ are on the view plane and $\overline{N}$ is normal to the plane. The dot product of two perpendicular vectors is always zero. So,

$$\overrightarrow{(R_0 P')} \cdot \overline{N} = 0$$

$$(P' - R_0) \cdot \overline{N} = 0$$

$$(x', y', z') - (x_0, y_0, z_0) \cdot (n_1, n_2, n_3) = 0$$

$$(x' - x_0)\, n_1 + (y' - y_0)\, n_2 + (z' - z_0)\, n_3 = 0$$

$$n_1 x' + n_2 y' + n_3 z' = n_1 x_0 + n_2 y_0 + n_3 z_0$$

Put, $x' = \alpha x$, $y' = \alpha y$ and $z' = \alpha z$ in above equation.

$$n_1 \alpha x + n_2 \alpha y + n_3 \alpha z = n_1 x_0 + n_2 y_0 + n_3 z_0$$

$$\alpha (n_1 x + n_2 y + n_3 z) = n_1 x_0 + n_2 y_0 + n_3 z_0$$

$$\alpha = \frac{n_1 x_0 + n_2 y_0 + n_3 z_0}{n_1 x + n_2 y + n_3 z}$$

Let, $d_0 = n_1 \cdot x_0 + n_2 \cdot y_0 + n_3 \cdot z_0$

$$\alpha = \frac{d_0}{n_1 x + n_2 y + n_3 z}$$

$$x' = \alpha x = \frac{d_0}{n_1 x + n_2 y + n_3 z} \cdot x$$

$$y' = \alpha y = \frac{d_0}{n_1 x + n_2 y + n_3 z} \cdot y$$

$$z' = \alpha z = \frac{d_0}{n_1 x + n_2 y + n_3 z} \cdot z$$

Transformation matrix for P′ = M · P would be,

$$M = \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & 0 \\ n_1 & n_2 & n_3 & 0 \end{bmatrix}$$

Example 4.2.5 : Derive a general perspective transformation matrix for projection on a a plane passing from reference point R_0 (x_0, y_0, z_0) having a normal vector $N = n_1 i + n_2 j + n_3 k$. Consider centre of projection at C (a, b, c).

Solution :

Spatial orientation of the view plane is shown in Fig. P. 4.2.5

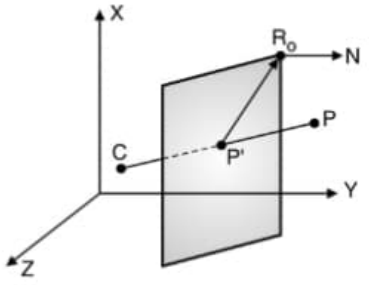


Fig. P. 4.2.5

Coordinate of center of projection is C(a, b, c). Let the point P be (x, y, z) and its projection on view plane is P'(x', y', z').

From Fig. P. 4.2.5,

$$\overrightarrow{CP'} = \alpha \overrightarrow{CP}$$

$$(P' - C) = \alpha (P - C)$$

$$(x', y', z') - (a, b, c) = \alpha \{(x, y, z) - (a, b, c)\}$$

$$(x' - a, y' - b, z' - c) = \alpha (x - a, y - b, z - c)$$

$$x' = a + \alpha (x - a)$$

$$y' = b + \alpha (y - b)$$

$$z' = c + \alpha (z - c)$$

Dot product of two perpendicular vectors is zero.

$$\therefore \overrightarrow{(R_0 P')} \cdot N = 0$$

$$(P' - R_0) \cdot N = 0$$

$$\{(x', y', z') - (x_0, y_0, z_0)\} \cdot (n_1, n_2, n_3) = 0$$

$$(x' - x_0) n_1 + (y' - y_0) n_2 + (z' - z_0) n_3 = 0$$

$$x' n_1 + y' n_2 + z' n_3 = n_1 x_0 + n_2 y_0 + n_3 z_0$$

Replacing value of x', y' and z'

$$(a + \alpha (x - a)) n_1 + (b + \alpha (x - b)) n_2 + (c + \alpha (x - c)) = n_1 x_0 + n_2 y_0 + n_3 z_0$$

Rearranging the terms,

$$\alpha (n_1 (x - a) + n_2 (y - b) + n_3 (z - c)) = (n_1 x_0 + n_2 y_0 + n_3 z_0) - (a \cdot n_1 + b \cdot n_2 + c \cdot n_3)$$

Let us take $d_0 = n_1 x_0 + n_2 y_0 + n_3 z_0$ and

$$d_1 = n_1 a + n_2 b + n_3 c$$

$$\alpha = \frac{d_0 - d_1}{(x - a) \cdot n_1 + (y - b) \cdot n_2 + (z - c) \cdot n_3}$$

And from previous comparison,

$$x' = (x - a) \cdot \alpha + a$$

$$y' = (y - b) \cdot \alpha + b$$

$$z' = (z - c) \cdot \alpha + c$$

From this, transformation matrix is written as,

$$M = \begin{bmatrix} d_0 + an_1 & an_2 & an_3 & (d_0 - a) + a((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ bn_1 & d_0 + bn_2 & bn_3 & (d_0 - b) + b((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ cn_1 & cn_2 & d_0 + cn_3 & (d_0 - c) + c((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ n_1 & n_2 & n_3 & (n_1 - a) + (n_2 - b) + (n_3 - c) \end{bmatrix}$$

Second Method :

1. Translate C to the origin.
2. This is identical to the previous case. Use that transformation matrix.
3. Inverse translation of C.

$$M = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & c \\ n_1 & n_2 & n_3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2.4 Parallel Vs. Perspective Projection

Q. Differentiate: Parallel projection vs. Perspective projection.

Parallel Projection	Perspective Projection
Projectors are parallel to each other.	Projectors are not parallel.
Need to specify the direction of projection.	Need to specify the center of projection.
Center of projection is at infinite distance.	Center of projection is at finite distance.
Does not produce realistic view.	Produces realistic view.
Depth information is lost.	Depth information is preserved.
Preserve relative proportion of object.	Does not Preserve relative proportion of object.
Subtypes : Orthographic projection, oblique projection.	Subtypes: Single point, two points, three-point projection

4.3 Lab Programs

Program 4.3.1 : Write a program for Three-dimensional transformation.
1. Translation 2. Scaling 3. Rotation. **(Lab 10 & Lab 11)**

Solution :

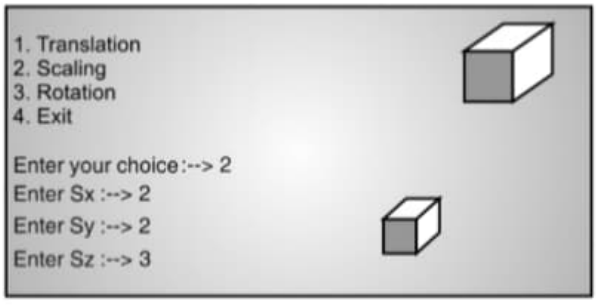
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void Translation();
void Scaling();
void Rotation();
int maxx, maxy, midx, midy;
void main()
{
    int choice;
    int gdriver = DETECT,gm;
```

```
    initgraph(&gdriver,&gm,"");

    maxx = getmaxx();
    maxy = getmaxy();
    midx = maxx/2;
    midy = maxy/2;

    printf("\n1.Translation");
    printf("\n2.Scaling");
    printf("\n3.Rotation");
    printf("\n4.exit");
    printf("\n\nEnter your choice :--> ");
    scanf("%d",&choice);

    do
    {
        switch(choice)
        {
            case 1 :    Translation();
                        getch();
                    break;
            case 2 :    Scaling();
                        getch();
                    break;
            case 3 :    Rotation();
                        getch();
                    break;
            case 4 :  break;
        }
            printf("\nEnter your choice :--> ");
            scanf("%d",&choice);
    } while(choice < 4);
}

    void Translation()
    {
        int tx,ty;
        bar3d(midx+50,midy-100,midx+60,midy-90,10,1);

        printf("Enter tx :-->");
        scanf("%d", &tx);
```

```
        printf("Enter ty :-->");
        scanf("%d", &ty);

        setcolor(GREEN);
        bar3d(midx +tx + 50,midy - (ty + 100), midx +tx + 60, midy - (ty + 90), 10, 1);
    }

    void Scaling()
    {
        float Sx,Sy,Sz;

        bar3d(midx + 50, midy - 100, midx + 60, midy - 90, 5, 1);

        printf("\nEnterSx :--> ");
        scanf("%f", &Sx);

        printf("\nEnterSy :--> ");
        scanf("%f", &Sy);

        printf("\nEnterSz :--> ");
        scanf("%f", &Sz);

        setcolor(RED);
        bar3d(midx + int(Sx * 50), midy - int(Sy * 100), midx + int(Sx * 60), midy - int(Sy * 90),
        int(5*Sz), 1);
}

    void Rotation()
    {
        int theta,x1,x2,y1,y2;

        bar3d(midx + 50, midy - 100, midx + 60, midy - 90, 5, 1);

        printf("\nEnter rotating angle :--> ");
        scanf("%d",&theta);

        x1 = 50*cos(theta*3.14/180) - 100*sin(theta*3.14/180);
        y1 = 50*sin(theta*3.14/180) + 100*cos(theta*3.14/180);
        x2 = 60*cos(theta*3.14/180) - 90*sin(theta*3.14/180);
        y2 = 60*sin(theta*3.14/180) + 90*cos(theta*3.14/180);
```

```
        setcolor(YELLOW);
        printf("\nAfter rotation about X axis: ");
        bar3d(midx + 50, midy - x1, midx + 60, midy - x2, 5, 1);

        printf("\nAfter rotation about Y axis: ");
        bar3d(midx + x1, midy - 100, midx + x2, midy - 90, 5, 1);
    }
```

Output :



Review Questions

Topic : 3D Translation

Q. 1 Derive 3D transformation matrix for translation operation.

Q. 2 Translate a triangle with vertices A (2, 2, 2), B (3, 4, 7) and C (8, 9, 12) by translation vector T [-2, -4, 5].

Topic : 3D Scaling

Q. 1 Derive 3D transformation matrices for scaling with respect to the origin and with respect to a reference point.

Q. 2 Derive 3D transformation matrices for scaling with respect to the origin.

Q. 3 Derive 3D transformation matrices for scaling with respect to a reference point.

Topic : 3D Rotation

Q. 1 Derive 3D transformation matrix for various cases of 3D rotation.

Q. 2 Derive 3D transformation matrix for rotation about principal axis.

Q. 3 How to perform 3D rotation about a line parallel to principal axis?

Q. 4 Derive generalized transformation matrix for rotation.

Q. 5 Rotate a triangle ABC with vertices A (2, 2, 2), B (3, 4, 7) and C (8, 9, 12) about Y-axis in a anti-clockwise direction by angle 45°.

Topic : Parallel Projection

Q. 1 Define the terms : Projection, Projection plane, Projector, Direction of Projection, and Center of Projection.

Q. 2 List and explain types of parallel projection.

Q. 3 Write a short note on parallel projection.

Q. 4 Write a short note on orthographic parallel projection.

Q. 5 Write a short note on axonometric parallel projection.

Q. 6 Define the term: Foreshortening factor.

Topic : Oblique Projection

Q. 1 What is oblique projection?

Q. 2 Discuss oblique projection with its types.

Q. 3 Define and discuss with its properties: Cavalier projection.

Q. 4 Define and discuss with its properties: Cabinet projection.

Q. 5 What is the relation between angle α and type of projection.

Q. 6 Find transformed co-ordinate of unit cube with one vertex at origin for,

(i) Cavalier projection with $\theta = 45°$.

(ii) Cabinet projection with $\theta = 30°$.

Topic : Perspective Projection

Q. 1 What do you mean by perspective projection?

Q. 2 Explain in detail: Perspective projection.

Q. 3 Define: Vanishing point, Principal vanishing point.

Q. 4 Write a short note on one-point perspective projection.

Q. 5 Explain view volume and view frustum with suitable diagram.

Q. 6 Compare and contrast: Parallel projection vs. Perspective projection.

Chapter Ends...

❑❑❑

Windowing and Clipping

Syllabus

Windowing and Clipping Concepts : Window-to-Viewport Transformation,
Line Clipping : Cohen Sutherland Clipping Algorithm, Cyrus beck, Liang Barsky, Midpoint Subdivision, Polygon Clipping Sutherland-Hodgeman, Text Clipping

5.1 Windowing and Clipping Concepts

5.1.1 Introduction

Q. Explain the concept of windowing and clipping.

- Typically, graphics packages allow the user to specify which part of the scene is to be displayed.
- The process of selecting the part of real-world scene to display it on some device is called **windowing.**
- **Clipping window** is the rectangular window against which visibility of scene object is tested.
- **Clipping** is the process of deciding and removing the portion of the object which is outside the clipping window.
- Part of the primitive outside the clipping window is not drawn. Clipping region may be of any arbitrary shape, but to simplify the clipping procedure, rectangle region is used.
- User can select single or clipping multiple windows simultaneously. Clipping reduces the computation by not processing the unnecessary part of the object.

Q. Explain various approaches for clipping the primitives.

There are two approaches for clipping the primitives.

1. **Scan conversion approach :** In this approach visibility of each pixel of an object is tested, and the pixel is drawn if it is visible. In other words, we perform the clipping during scan conversion of the object. This approach is simple but does extensive calculations.
2. **Analytical approach :** In this approach, before scan converting the object, visible part of the primitives is calculated analytically. This approach does not perform the check for individual pixel. It reduces the computation but it is complex.

5.1.2 Viewing Pipeline

Q. Write a short note on viewing pipeline.
Q. Define the terms: Window, Viewport, Window-to-viewport transformation.
Q. Discuss the viewing pipeline in computer graphics with a neat diagram.

- The visible region of the scene inside the clipping window is displayed on some device. More formally, the world coordinate area which is selected for display is called a **window**.
- An area on display device to which window is mapped is called **viewport**.

- Window defines *what* is to be displayed from the scene and viewport defines *where* it is to be displayed on the screen.
- In most of the graphics package window and viewport is a rectangle, with edges parallel to the principal axis.
- The process of mapping the part of world coordinate scene to device coordinate is referred to as a **viewing transformation** or **window to viewport transformation** or **windowing transformation**.
- Fig. 5.1.1 illustrates the concept and relationship of window and viewport.

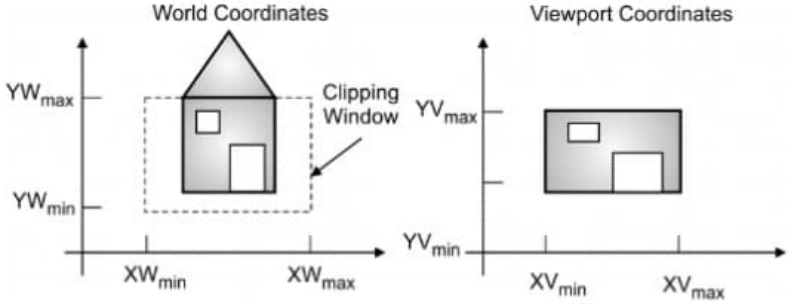


Fig. 5.1.1 : Window to viewport transformation

- We can define the scene by creating objects with their actual dimensions, called **modelling coordinates**, or **local coordinates** or **master coordinates** i.e. (x_{mc}, y_{mc}).
- Once the object is designed, it is placed at an appropriate position in the scene using **world coordinates** i.e. (x_{wc}, y_{wc}).
- The scene may be displayed on different devices, so it must be mapped to the proper dimension to render it properly. World coordinates are then represented in **normalized coordinate** (x_{nc}, y_{nc}), usually between 0 and 1. This makes the system independent of the various display devices.
- These normalized coordinate can be easily scaled to fit on any size of display by multiplying it with a proper scaling factor that is known as **device coordinate** (x_{dc}, y_{dc}). Fig. 5.1.2 shows sequence of coordinate transformations.

$(x_{mc}, y_{mc}) \rightarrow (x_{wc}, y_{wc}) \rightarrow (x_{nc}, y_{nc}) \rightarrow (x_{dc}, y_{dc})$

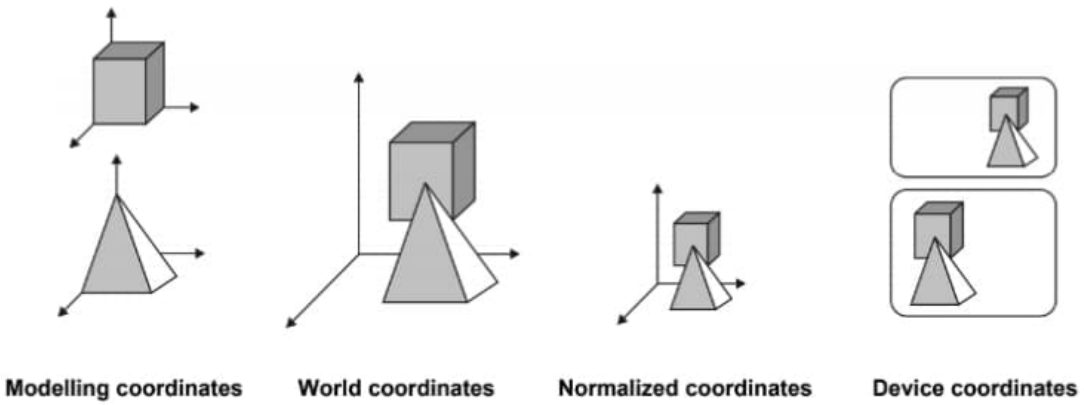


Fig. 5.1.2 : Coordinate transformation

The viewing pipeline is depicted in Fig. 5.1.3.

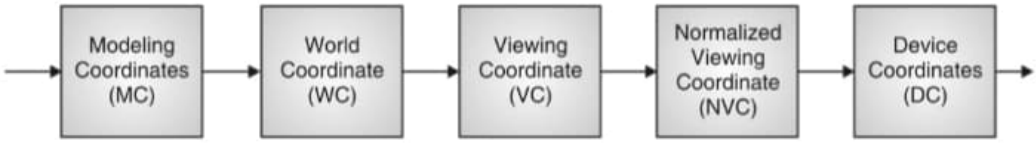


Fig. 5.1.3 : Steps for the window to viewport transformation

Q. Discuss the effect of different size and position of window and viewport in viewing.

- As shown in the Fig. 5.1.4, we can view the scene on the device at different locations by changing the position of viewport.

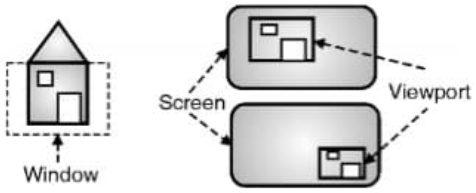


Fig. 5.1.4 : Mapping same window to different viewports

- As shown in Fig. 5.1.4, the same window is mapped to viewports with different size and location on the screen
- To achieve zoom in effect, we shall map window with different size on fixed size viewport. If the window size is made smaller, we get a zoom in effect and if the window size is made larger, we get a zoom out effect.
- **Panning** is achieved by moving a fixed size window over various positions in the scene.

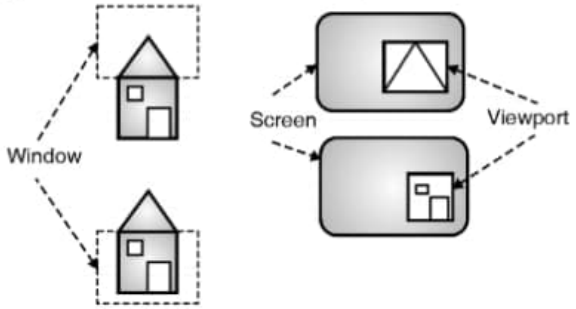


Fig. 5.1.5 : Effect of panning

5.1.3 Window-to-viewport Transformation

Q. Derive transformation matrix for window-to-viewport transformation.

- Physical description of the object goes through multiple sequences as shown in the Fig. 5.1.5.
- Window to the viewport to transformation is necessary because the size of window and viewport may not be the same all the time. So actual scene selected by window needs to be rescaled to fit it in the viewport.
- Let (XW_{min}, YW_{min}) and (XW_{max}, YW_{max}) represent the lower left and upper-top corner points of clipping window, respectively.
- And let (XV_{min}, YV_{min}) and (XV_{max}, YV_{max}) represent the lower left and upper-top corner points of the viewport, respectively.
- As shown in Fig. 5.1.6, point (xw, yw) in window is to be mapped to point (xv, yv) in viewport.

- To maintain the same relative placement in the viewport as in a window, we normalize both.

$$\frac{xv - XV_{min}}{XV_{max} - XV_{min}} = \frac{xw - XW_{min}}{XW_{max} - XW_{min}}$$

$$xv - XV_{min} = (XV_{max} - XV_{min}) \cdot \frac{xw - XW_{min}}{XW_{max} - XW_{min}}$$

$$v = XV_{min} + (xw - XW_{min}) \cdot S_x$$

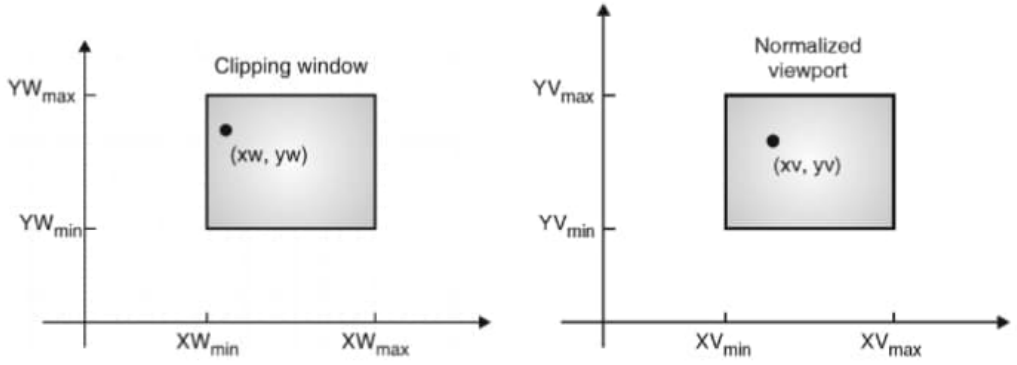


Fig. 5.1.6 : Relation between window and viewport

Similarly,

$$\frac{yv - YV_{min}}{YV_{max} - YV_{min}} = \frac{yw - YW_{min}}{YW_{max} - YW_{min}}$$

$$yv - YV_{min} = (YV_{max} - YV_{min}) \cdot \frac{yw - YW_{min}}{YW_{max} - YW_{min}}$$

$$yv = YV_{min} + (yw - YW_{min}) \cdot S_y$$

Where the scaling factors are,

$$S_x = \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}}$$

$$S_y = \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}}$$

Matrix Representation :

- If the lower left corner of the window is not at the origin, we should first translate it to the origin before we map the points in the window to the viewport.
- Window to viewport transformation is achieved by the following three steps :
- Translate the lower-left point of window to the origin.

Translation matrix is defined as, $T = \begin{bmatrix} 1 & 0 & -XW_{min} \\ 0 & 1 & -YW_{min} \\ 0 & 0 & 1 \end{bmatrix}$

- Apply scaling to map the scene to the viewport.

Scaling matrix is defined as, $S = \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & 0 \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- Inverse translation in the viewport.

Inverse translation matrix is defined as, $T^{-1} = \begin{bmatrix} 1 & 0 & XV_{min} \\ 0 & 1 & YV_{min} \\ 0 & 0 & 1 \end{bmatrix}$

The composite transformation matrix for the window to viewport transformation is given as,

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

$$M = \begin{bmatrix} 1 & 0 & XV_{min} \\ 0 & 1 & YV_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & 0 \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -XW_{min} \\ 0 & 1 & -YW_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & -XW_{min} \cdot \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} + XV_{min} \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & -YW_{min} \cdot \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} + YV_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

5.2 Point Clipping

Q. What is the point clipping? How to achieve it?

- Before we discuss line clipping, let's look at the simpler problem of clipping individual point.
- Point clipping is very simple. It's a trivial case of acceptance or rejection. The point can be either fully inside or fully outside the clipping region. The point on the clipping window boundary is considered inside.
- Let (X_{min}, Y_{min}) and (X_{max}, y_{max}) represents the lower-left and top-right corners of the clipping window, respectively.
- Point (x, y) is inside the region only if the following four inequalities are true,

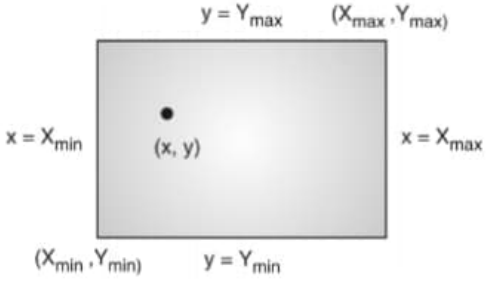


Fig. 5.2.1 : Point clipping

$$X_{min} \leq x \leq X_{max}$$

$$Y_{min} \leq y \leq Y_{max}$$

- For point being inside the clipping window, all four conditions must hold. If any of the four inequalities does not hold, the point is outside the clipping rectangle.

5.3 Line Clipping

5.3.1 Introduction

Q. What is line clipping? Discuss various cases of clipping with suitable diagram.

- The line is widely used primitive in engineering drawing. Most of the engineering shapes contain lines.
- When we display the scene on the monitor screen, we should not render the portion of the scene which is outside the clipping window. It is essential to determine which portion of the line is inside the clipping region.
- Various cases of line membership are discussed here. Consider the clipping rectangle PQRS in Fig. 5.3.1. Scene contains four lines. If both endpoints of the line are within a rectangle, the line is completely visible. Otherwise, line may be partially visible or completely outside the window.
- Line EF is completely visible and will be displayed as it is.
- One endpoint of line AB is outside the clipping region. The intersection point of line AB with the top edge is A′. Hence segment AA′ will be displayed and A′B will be clipped.
- Line CD is completely outside the clipping window so it won't be displayed at all.
- Line HI intersects left and top border of clipping regions. Intersections with respective boundaries are H′ and I′. So only segment H′I′ will be displayed. Segments HH′ and I′I would be clipped.
- Fig. 5.3.2 shows the output of the scene after the clipping procedure.

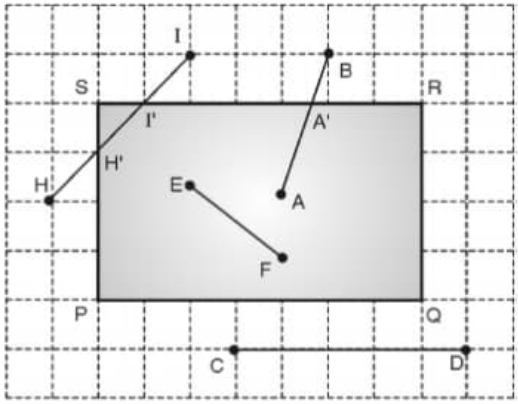


Fig. 5.3.1

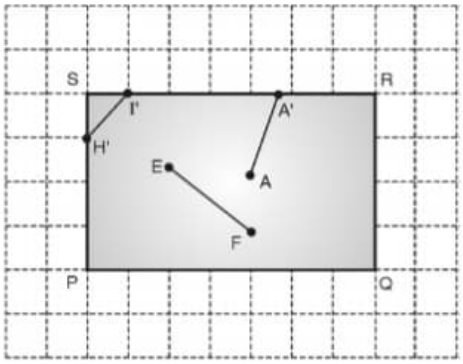


Fig. 5.3.2

5.3.2 Line Clipping Algorithm

In the following section, we will study four line clipping algorithms.

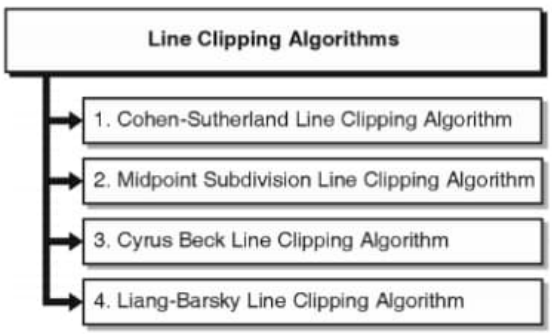


Fig. 5.3.3 : Clipping algorithms

5.3.2.1 Cohen Sutherland Line Clipping Algorithm

Q. Discuss the Cohen-Sutherland Line clipping algorithm.

Q. Write and explain with an example Cohen-Sutherland line clipping algorithm.

Cohen Sutherland is a 2D line clipping algorithm. The main advantage of the algorithm is that it vastly reduces the number of line intersections that must be calculated in scan conversion approach.

It operates in two phases :

1. Region code generation
2. Clipping

1. Region Code Generation :

Q. How region codes are computed in the Cohen-Sutherland Algorithm?

- It divides the plane into nine regions and determines the visible portion of the line using outcodes (region code) of the endpoints. Outcode is a four-bit number. Values of these bits are set 1 if conditions shown in Fig. 5.3.4 are true for a particular bit.

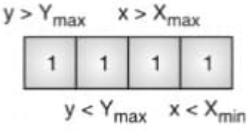


Fig. 5.3.4 : Setting the bits of outcode

For any endpoint (x, y) of a line, the outcode is set according to the following conditions :

- Set first bit (most significant bit) if a point lies above window i.e. $y > Y_{max}$
- Set second bit if a point lies below window i.e. $y < Y_{min}$
- Set third bit if a point lies to the right of window i.e. $x > X_{max}$
- Set fourth bit (least significant bit) if a point lies to left of window i.e. $x < X_{min}$

Fig. 5.3.5 : Outcodes for all nine regions

- If the point is inside the clipping window, none of the above conditions would be true. So, outcode of endpoint inside the clipping region would be 0000. Outcode of all nine regions are shown in Fig. 5.3.5.

2. Clipping Procedure :

- The algorithm quickly detects two trivial cases.
- If both endpoints of a line lie inside the window, the entire line is visible. It is trivially accepted and needs no clipping.
- On the other hand, if both endpoints of a line lie completely on one side of the window, the line lies completely outside the window. It is completely rejected.
- After calculating outcodes of both endpoints of the line, logical OR-AND conditions are evaluated to check trivial acceptance, rejection or partial visibility. We will discuss three possible cases.

Case I :

- The line is completely inside clipping window if logical OR operation of outcodes yields to 0000. For example, line AB in Fig. 5.3.6 is trivially accepted.
- In short, if outcode of both endpoints is 0000, the line is fully visible.

Case II :

- If logical OR of the outcodes is not 0000, there are two possibilities, the line may be partially visible or it may be completely outside.
- The line is completely outside the clipping window if logical AND operation of outcomes does not yield to 0000. Above both tests are performed in the same order, i.e. Case II is applicable only if Case I is not true.
- For example, one endpoint of the line CD in Fig. 5.3.6 has outcode 1010 while the other endpoint has a code 0010. Logical OR is 1010, so it is not fully visible. The logical AND would be 0010 which indicates the line segment lies completely outside of the window.

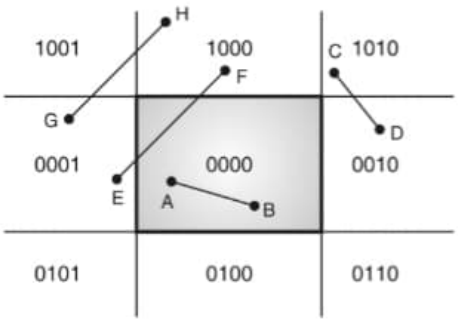


Fig. 5.3.6 : Different clipping cases

Case III :

- Outcodes of endpoints of the line, EF are 1000 and 0001. Logical OR of endpoints is 1001, so the line is not fully visible. The logical AND would be 0000, and the line could not be trivially rejected.
- In this case, we need to process line further. If logical AND operation of both endpoints is 0000, the line may or may not pass from clipping window. For example, like line EF, logical AND of outcodes of endpoints of line GH is also 0000, but that line is completely outside.
- If the line is not trivially accepted or rejected, then compare endpoints with window boundary to determine how much line segment can be discarded. The intersection point of line with clipping region edge can be computed as follow.

  Consider the line with endpoints(x_1, y_1) and (x_2, y_2). The slope of a line is given as,

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

- Using explicit representation of the line, $y = mx + c$, we can compute Y-intercept of the line as, $c = y - mx$, where (x, y) is any point on line. Both endpoints of the line are known, so we can put anyone in this equation to compute c, let's put (x_1, y_1), so $c = y_1 - mx_1$.
- The Y Coordinate of an intersection with a vertical window boundary can be calculated as $y = mx + c$, where x is either X_{min} or X_{max}, depends on for which vertical line we are calculating Y.

$$Y = m \cdot X_{min} + c \qquad \text{(for intersection with left edge)}$$

Or

$$Y = m \cdot X_{max} + c \qquad \text{(for intersection with left-right)}$$

In a similar way, X coordinate of an intersection with a horizontal window boundary is computed as,

$$X = (y - c) / m$$

Where y is either Y_{min} or Y_{max}, depends on for which horizontal line we are calculating X.

$$X = \frac{(Y_{min} - c)}{m} \quad \text{(for intersection with bottom edge)}$$

Or $$X = \frac{(Y_{max} - c)}{m} \quad \text{(for intersection with a top edge)}$$

- This process is repeated until we find the entire line segment within the window.

Q. State the advantages and disadvantages of the Cohen-Sutherland Algorithm.

Advantages :

- It is easy to understand.
- Simple to implement.
- Best suitable for the lines fully inside or outside.
- It can easily be extended for 3D line clipping.

Limitations :

- Repeated clipping is expensive.
- Only applicable to rectangular clipping window. It cannot handle any other shape.
- It can be improved using more regions (e.g. Nichol Lee Nichol approach).

Algorithm for Cohen-Sutherland line clipping approach :

Algorithm for Cohen-Sutherland line clipping approach is depicted here.

W-18

Q. Write down Cohen-Sutherland Line clipping algorithm. **(W-18, 4 Marks)**

Step 1 : Read (X_{min}, Y_{min}) and (X_{max}, Y_{max}) – Lower-left and top-right corner of clipping window

Step 2 : Read $A(x_1, y_1)$ and $B(x_2, y_2)$ end points of line

Step 3 : Compute outcode of A and B

if $y_1 > Y_{max}$ **then** Outcode_A(1) = 1 **else** 0	**if** $y_2 > Y_{max}$ **then** Outcode_B(1) = 1 **else** 0
if $y_1 < Y_{min}$ **then** Outcode_A(2) = 1 **else** 0	**if** $y_2 < Y_{min}$ **then** Outcode_B(2) = 1 **else** 0
if $x_1 > X_{max}$ **then** Outcode_A(3) = 1 **else** 0	**if** $x_2 > X_{max}$ **then** Outcode_B(3) = 1 **else** 0
if $x_1 < X_{min}$ **then** Outcode_A(4) = 1 **else** 0	**if** $x_2 < X_{min}$ **then** Outcode_B(4) = 1 **else** 0

Step 4 : **if** Outcode_A **OR** Outcode_B == 0000 **then**

Display entire line and goto step 5

else if Outcode_A **AND** Outcode_B ≠ 0000 **then**

Reject the entire line

else

Compute the intersection point with window boundaries

Repeat Step 4.

Step 5 : Stop

Example 5.3.1 : Use Cohen-Sutherland 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). **(W-18, 4 Marks)**

Solution :

Spatial position of lines and rectangle is shown in Fig. P. 5.3.1.

Line P_1P_2 :

Given that, end points of line are P_1(40, 15) and P_2(75, 45)

Outcode (P_1) = 0001 = A_1

Outcode (P_2) = 1000 = B_1

A_1 OR B_1 = 0001 OR 1000 = 1001

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

A_1 AND B_1 = 0001 AND 1000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

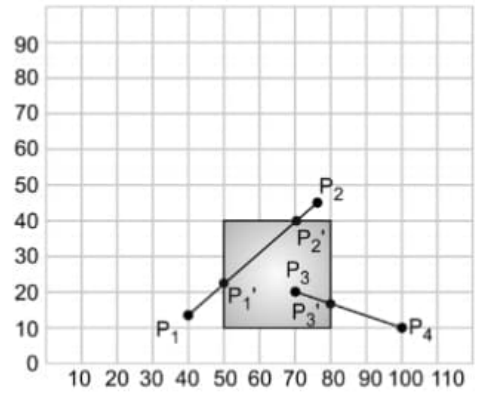


Fig. P. 5.3.1

Outcode of endpoint A is 0001, that means line intersects left edge while moving from P_1 to P_2. Let us call the intersection of point P_1'. P_1' is on line $x = x_{min} = 50$. So its x coordinate is 50, we have to compute only y coordinate of P_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{45 - 15}{75 - 40} = \frac{30}{35} = 0.857$$

$$\text{From } y = mx + c,\ c = y - mx$$

By putting one of the points of line AB in this equation, we can find intercept of line with Y axis. Let's put P_1 (40, 15) in above equation,

$$\therefore \quad c = 15 - (0.857) \times (40) = 15 - 34.28 = -19.28$$

Put $x = x_{min} = 50$ in explicit line equation to compute corresponding Y coordinate

$$y = mx + c = 0.857 \times (50) - 19.28 = 42.85 - 19.28 = 23.57$$

Thus, $P_1' = (50, 23.57)$

Outcode of endpoint P_2 is 1000, that means line intersects top edge. Let us call the intersection of point P_2'. P_2' is on line y = 40. So its y coordinate is 40, we have to compute only x co-ordinate of B'.

From $y = mx + c,$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x coordinate of P_2'.

$$\therefore \quad x = \frac{40 + 19.28}{0.857} = \frac{59.28}{0.857} = 69.17$$

Thus, $P_2' = (69.17, 40)$

So, line segment with end points P_1'(50, 23.57) and P_2'(69.17, 40) is the visible segment.

Line P_3P_4 :

Given that, endpoints of the line are $P_3(70, 20)$ and $P_4(100, 10)$

Outcode $(P_3) = 0000 = C_1$

Outcode $(P_4) = 0010 = D_1$

C_1 OR $D_1 = 0000$ OR $0010 = 0010$

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

C_1 AND $D_1 = 0000$ AND $0010 = 0000$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{10 - 20}{100 - 70} = \frac{-10}{30} = -0.33$$

From, $y = mx + c,\ c = y - mx$

By putting one of the points of line P_3P_4 in this equation, we can find intercept of line with Y axis. Let's put $P_3(70, 20)$ in above equation,

$$\therefore \quad c = 20 - (-0.33) \times (70) = 20 + 23.1 = 43.1$$

Outcode of end point P_4 is 0010, that means line intersects right edge. Let us call the intersection of point P_3'.

From, $y = mx + c,$

By putting the value of x of a right edge in this equation, we can find the intersection y coordinate of P_3'.

$$\therefore \quad y = (-0.33 \times 80) + 43.1 = -26.4 + 43.1 = 16.7$$

So, line segment with end points $P_3(70, 20)$ and $P_3'(80, 16.7)$ is the visible segment.

Original line Segments	Visible line segments
$P_1(40, 15)$ and $P_2(75, 45)$	$P_1'(50, 23.57)$ and P_2' (69.17, 40)
$P_3(70, 20)$ and $P_4(100, 10)$	P_3' (70, 20) and P_3' (80, 16.7)

Example 5.3.2 : Let R be the rectangular window whose lower left-hand corner at L(–3, 1) and the upper right corner is at R (2, 6). Find the region codes for the endpoints and use the Cohen Sutherland algorithm to clip the line segments. Coordinates for line segments are,

For line AB, A(–4, 2) and B(–1, 7)

For line CD, C(–1, 5) and D(3, 8)

For line EF, E(–2, 3) and F(1, 2)

Solution :

The spatial position of lines and rectangle is shown in the Fig. P. 5.3.2.

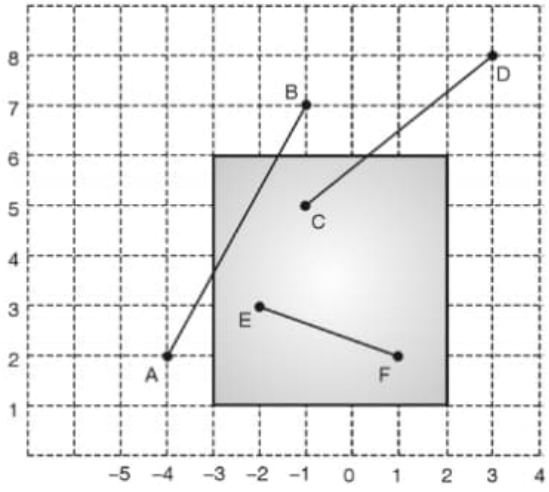


Fig. P. 5.3.2

Line AB :

Given that, endpoints of the line are A (– 4, 2) and B (– 1, 7)

Outcode (A) = 0001 = A_1

Outcode (B) = 1000 = B_1

A_1 OR B_1 = 0001 OR 1000 = 1001

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

A_1 AND B_1 = 0001 AND 1000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint A is 0001, that means line intersects left edge while moving from A to B. Let us call the intersection of point A with the left edge is A′. A′ is on line x = – 3. So its x coordinate is – 3, we have to compute only y co-ordinate of A′.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-2}{-1+4} = \frac{5}{3} = 1.67$$

$$\text{From } y = mx + c, \quad c = y - mx$$

By putting one of the points of line AB in this equation, we can find intercept of line with y axis. Let's put A (– 4, 2) in above equation,

$$\therefore \quad c = 2 - (1.67) \times (-4) = 2 + 6.68 = 8.68$$

$$\text{From , } \quad y = mx + c = 1.67 \times (-3) + 8.68 = -5.01 + 8.68 = 3.67$$

Outcode of end point B is 1000, that means line intersects top edge while moving from A to B. Let us call the intersection of point B with top edge is B′. B′ is on line y = 6. So its y co-ordinate is 6, we have to compute only x co-ordinate of B′.

From, $y = mx + c,$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x coordinate of B′.

$$\therefore \quad x = \frac{6 - 8.68}{1.67} = \frac{-2.68}{1.67} = -1.60$$

Outcode (A′) OR outcode (B′) = 0000 OR 0000

$$= 0000$$

So line segment with endpoints A′(– 3, 3.67) and B′(– 1.6, 6) is the visible segment.

Line CD :

Given that, end points of line are C (–1, 5) and D (3, 8)

Outcode (C) = 0000 = C_1

Outcode (D) = 1010 = D_1

C_1 OR D_1 = 0000 OR 1010 = 1010

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

C_1 AND D_1 = 0000 AND 1010 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint C is 0000, that means endpoint C is inside the clipping region.

Slope of line = m= $\frac{y_2 - y_1}{x_2 - x_1} = \frac{8 - 5}{3 + 1} = \frac{3}{4} = 0.75$

From $y = mx + c, \quad c = y - mx$

By putting one of the points of line CD in this equation, we can find intercept of line with the y-axis. Let's put C (– 1, 5) in the above equation,

$$\therefore \quad c = 5 - (0.75) \times (-1) = 5 + 0.75 = 5.75$$

Outcode of endpoint D is 1010, that means line intersects top and right edges while moving from A to B. Let us call the intersection of point D with the top edge is D′. D′ is on line y = 6. So its y coordinate is 6, we have to compute only x coordinate of D′.

From $y = mx + c,$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x coordinate of D′.

$$\therefore \quad x = \frac{6 - 5.75}{0.75} = \frac{0.25}{0.75} = 0.33$$

Outcode (C) OR outcode (D′) = 0000 OR 0000

$$= 0000$$

So line segment with endpoints C(– 1, 5) and D′(0.33, 6) is the visible segment.

Line EF :

Given that, endpoints of the line are E (– 2, 3) and F (1, 2)

Outcode (E) = 0000 = E_1

Outcode (F) = 0000 = F_1

E_1 OR F_1 = 0000 OR 0000 = 0000

Logical OR of outcodes of both endpoint is 0000, so the line is completely visible. It does not require any clipping

Original line Segments	Visible line segments
A(– 4, 2) and B(– 1, 7)	A'(– 3, 3.67) and B'(– 1.6, 6)
C(– 1, 5) and D(3, 8)	C(– 1, 5) and D'(0.33, 6)
E(– 2, 3) and F(1, 2)	E(– 2, 3) and F(1, 2)

Example 5.3.3 : Let ABCD be the rectangular window with A(20, 20), B(90, 20), C(90, 70) and D(20, 70). Find region codes for endpoints and use the Cohen Sutherland algorithm to clip the lines:
(i) P_1P_2 with P_1(10, 30), P_2(80, 90)
(ii) Q_1Q_2 with Q_1(10, 10), Q_2(70, 60)

Solution :

The spatial position of lines and rectangle is shown in Fig. P. 5.3.3.

Line P_1P_2:

Given that, end points of line are P_1 (10, 30) and P_2 (80, 90)

Outcode (P_1) = 0001 = X

Outcode (P_2) = 1000 = Y

X OR Y = 0001 OR 1000 = 1001

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

X AND Y = 0001 AND 1000 = 0000

Logical AND of outcodes of both endpoint is 0000. So line is not completely outside. Let us decide the visible portion of the line.

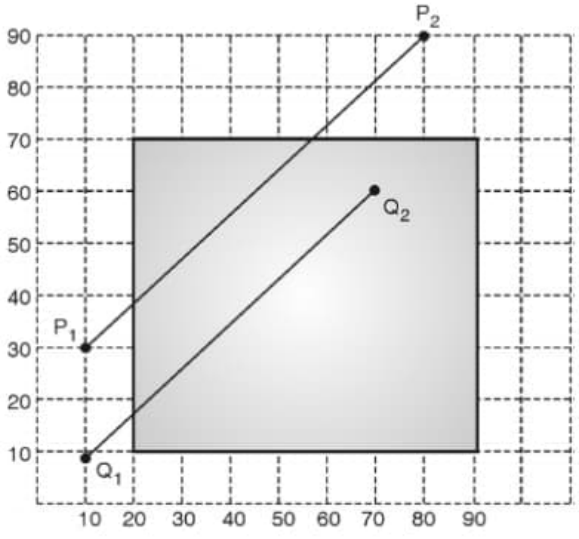


Fig. P.5.3.3

Outcode of end point, P_1 is 0001, that means line intersects left edge while moving from P_1 to P_2. Let us call the intersection of point P_1 with the left edge is P_1'. P_1' is on line x = 20. So its x coordinate is 20, we have to compute the only y co-ordinate of P_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{90 - 30}{80 - 10} = \frac{60}{70} = 0.86$$

$$\text{From } y = mx + c, \quad c = y - mx$$

By putting one of the points of line P_1P_2 in this equation, we can find intercept of line with y axis. Let's put P_1 (10, 30) in above equation,

$$\therefore \quad c = 30 - (0.86) \times (10) = 30 - 8.6 = 21.4$$

$$\text{From } y = mx + c = 0.86 \times (20) + 21.4 = 17.2 + 21.4 = 38.6$$

Outcode of end point P_2 is 1000, that means line intersects top edge while moving from P_1 to P_2. Let us call the intersection of point P_2 with top edge is P_2'. P_2' is on line y = 70. So its y co-ordinate is 70, we have to compute only x co-ordinate of P_1'.

From,

$$y = mx + c,$$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x coordinate of P_2'.

$$\therefore x = \frac{70 - 21.4}{0.86} = \frac{48.6}{0.86} = 56.51$$

$$\text{Outcode } (A') \text{ OR outcode } (B') = 0000 \text{ OR } 0000$$

$$= 0000$$

So line segment with endpoints P_1' (20, 38.6) and P_2' (56.51, 70) is the visible segment.

Line Q_1Q_2:

Given that, endpoints of the line are Q_1 (10, 10) and Q_2 (70, 60)

Outcode (Q_1) = 0101 = X

Outcode (Q_2) = 0000 = Y

X OR Y = 0101 OR 0000 = 0101

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

X AND Y = 0101 AND 0000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint Q_1 is 0101, that means line intersects bottom and left edges while moving from Q_1 to Q_2. Let us call the intersection of point Q_1 with the bottom edge is Q_1'. Q_1' is on line y = 20. So its y coordinate is 20, we have to compute only x co-ordinate of Q_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{60 - 10}{70 - 10} = \frac{50}{60} = 0.83$$

$$\text{From } y = mx + c, \quad c = y - mx$$

By putting one of the points of line P_1P_2 in this equation, we can find intercept of line with y axis. Let's put Q_1 (10, 10) in above equation,

$$\therefore \quad c = 10 - (0.83) \times (10) = 10 - 8.3 = 1.7$$

$$\text{From } y = mx + c,$$

$$x = \frac{y - c}{m}$$

By putting value of y of top edge in this equation, we can find the intersection x coordinate of Q_1'.

$$\therefore x = \frac{20 - 1.7}{0.83} = \frac{18.3}{0.83} = 22.04$$

$$\text{Outcode } (Q_1') \text{ OR outcode } (Q_2) = 0000 \text{ OR } 0000$$

$$= 0000$$

So line segment with endpoints Q_1' (22.04, 20) and P_2' (70, 60) is the visible segment.

Original line Segments	Visible line segments
P_1(10, 30) and P_2(80, 90)	P_1' (20, 38.6) and P_2' (56.51, 70)
Q_1(10, 10) and Q_2(70, 60)	Q_1' (22.04, 20) and Q_2(70, 60)

Example 5.3.4 : 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 :

Given that, endpoints of the line are P (4, 1) and Q (6, 4)

The line is to be clipped against rectangle A (3, 2), B (7, 2), C (7, 6) and D (3, 6).

The spatial position of the line and rectangle is shown in Fig. P. 5.3.4.

Outcode (P) = 0100 = P_1

Outcode (Q) = 0000 = Q_1

P_1 OR Q_1 = 0100 OR 0000 = 0100

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

P_1 AND Q_1 = 0100 AND 0000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

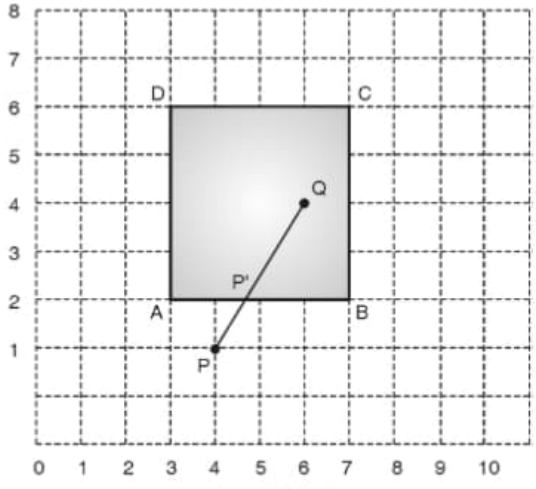


Fig. P. 5.3.4

Outcode of endpoint P is 0100, that means line intersects bottom edge while moving from P to Q. Let us call the intersection point of P with the bottom edge is P′. P′ is on line y = 7. So its y coordinate is 7, we have to compute only x coordinate of P′.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4 - 1}{6 - 4} = \frac{3}{2} = 1.5$$

$$\text{From } y = mx + c, \quad c = y - mx$$

By putting one of the points of line PQ in this equation, we can find intercept of line with y axis. Let's put P (4, 1) in above equation,

$$\therefore c = 1 - (1.5) \times (4) = 1 - 6 = -5$$

$$\text{From } y = mx + c$$

$$x = \frac{y - c}{m}$$

By putting the value of y of a bottom edge in this equation, we can find the intersection x coordinate of P'.

$$\therefore x = \frac{2 - (-5)}{1.5} = \frac{7}{1.5} = 4.67$$

∴ Coordinate of P' would be (4.67, 2).

$$\text{outcode (P')} \text{ OR outcode (Q)} = 0000 \text{ OR } 0000$$

$$= 0000$$

So line segment P'Q is the visible segment.

Original line Segments	Visible line segments
P(4, 1) and Q(6, 4)	P'(4.67, 2) and Q(6, 4)

5.3.2.2 Midpoint Subdivision Line Clipping Algorithm

Q. How midpoint line clipping algorithm works?

Q. Explain Mid-point subdivision algorithm.

Midpoint subdivision is the special case of the Cohen-Sutherland algorithm. Like Cohen-Sutherland, midpoint algorithm also operates in two phases.

Phase 1 :

- This phase is identical to the first stage of the Cohen-Sutherland algorithm.
- It computes the outcome of line endpoints and determines the intersecting side of the clipping region with the line.

Phase 2 :

- Treatment of partially visible line differs from the Cohen-Sutherland algorithm.
- The partially visible line is subdivided into two halves (i.e. divide line from midpoint) and processed both halves recursively.
- If subpart of the line becomes fully visible or fully invisible, stop processing otherwise recursively keep dividing the line from the midpoint. Consider the Fig. 5.3.7.

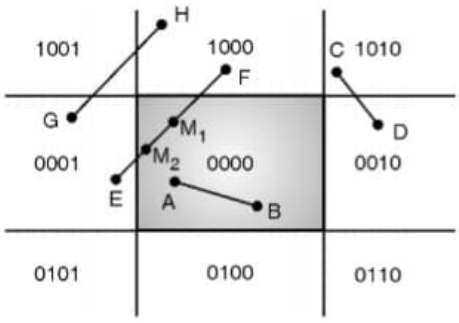


Fig. 5.3.7

Case - I : Fully visible line

Outcode of endpoints of line AB are 0000 and 0000, respectively.

$$\text{Outcode(A)} = 0000$$

$$\text{Logical OR} \quad \text{Outcode(B)} = 0000$$

$$= 0000$$

So, the line is fully visible.

Case - II : Fully outside line

	Outcode(C)	=	1010
Logical OR	Outcode(D)	=	0010
	1010	≠	0000

So, line is not fully visible

	Outcode(C)	=	1010
Logical AND	Outcode(D)	=	0010
	0010	≠	0000

So, the line is not completely outside.

Case - III : Partially visible line

Outcode of endpoints of line, EF are 0001 and 1000, respectively.

	Outcode(E)	=	0001
Logical OR	Outcode(F)	=	0010
	1000	≠	0000

So, line is not fully visible.

Now perform logical end operation,

	Outcode(E)	=	0001
Logical AND	Outcode(F)	=	1000
		=	0000

So, line is partially visible.

This line may intersect with the clipping window boundary. Let the endpoints of the line are $E(x_1, y_1)$ and $F(x_2, y_2)$. To compute the visible span, we shall first divide the line two halves and process each half individually.

The midpoint of the line EF would be $M_1 = (x_{m1}, y_{m1}) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$

Compute the outcome of point M_1 and see if line segment EM_1 is within the polygon, if so display the line segment, otherwise subdivide it. From the figure, it seems that the line segment is not fully visible, so let us divide it further.

The midpoint of the line EM_1 would be $M_1 = (x_{m1}, y_{m1}) = \left(\frac{x_1 + x_{m1}}{2}, \frac{y_1 + ym_2}{2}\right)$

This process is repeated until the segment is fully visible. The same procedure is repeated to other halves too.

Steps for midpoint line clipping algorithm :

Q. Mention the steps for midpoint line clipping algorithm.

Steps for midpoint line clipping approach are stated below :

Step 1 : Read (X_{min}, Y_{min}) and (X_{max}, Y_{max}) – Lower-left and top-right corner of clipping window

Step 2 : Read $A(x_1, y_1)$ and $B(x_2, y_2)$ end points of line

Step 3 : Compute outcode of A and B

if $y_1 > Y_{max}$ **then** Outcode_A(1) = 1 **else** 0 **if** $y_2 > Y_{max}$ **then** Outcode_B(1) = 1 **else** 0

if $y_1 < Y_{min}$ **then** Outcode_A(2) = 1 **else** 0 **if** $y_2 < Y_{min}$ **then** Outcode_B(2) = 1 **else** 0

if $x_1 > X_{max}$ **then** Outcode_A(3) = 1 **else** 0 **if** $x_2 > X_{max}$ **then** Outcode_B(3) = 1 **else** 0

if $x_1 < X_{min}$ **then** Outcode_A(4) = 1 **else** 0 **if** $x_2 < X_{min}$ **then** Outcode_B(4) = 1 **else** 0

Step 4 : **if** Outcode_A **OR** Outcode_B == 0000 **then**

Display entire line and goto step 5

else if Outcode_A **AND** Outcode_B ≠ 0000 **then**

Reject the entire line

else

Divide the line in two halves from mid-point

Recursively call step 3 for both the halves

Step 5 : Stop

5.3.2.3 Cyrus-Beck Line Clipping Algorithm

Q. Write a short note on Cyrus-Beck Line Clipping algorithm.
Q. Explain the Cyrus-Beck algorithm.

- The Cyrus-Beck algorithm is a generalized line-clipping algorithm. It is more efficient than the Cohen-Sutherland algorithm which uses repetitive clipping. Cohen Sutherland algorithm is applicable only for rectangular clipping area.
- Cyrus-Beck is a general algorithm and can be used with a convex polygon clipping window.
- It uses a parametric equation of a line to perform the clipping, so it is also known as **parametric line clipping** algorithm.
- It can easily be extended to clip 3D line against polyhedron.

Parametric equation of a line in the plane is given as,

$$P(t) = P_1 + (P_2 - P_1) \cdot t, \quad 0 \le t \le 1,$$

$P_1(x_1, y_1)$ and $P_2(x_2, y_2)$ are the endpoints of the line. For t = 0, above equation interpolates end point P_1 and with t = 1, it interpolates P_2. Any value of t between 0 and 1 generates an appropriate intermediate point on line. t < 0 define a point before P_1 on extended line, and t > 1 define a point after P_2.

Individual coordinates of a pointon line are computed as follow :

$$x = x_1 + (x_2 - x_1) \cdot t$$

$$y = y_1 + (y_2 - y_1) \cdot t$$

- Cyrus-Beck algorithm uses the normal vector for determining whether a point is inside, on, or outside a window.
- Consider a convex planar polygon R as shown in Fig. 5.3.8. If F is any point on the boundary of a convex polygon, and N_i is inward normal ($N_{out} = -N_{in}$) for one of its boundaries, then for any value of parameter 't' (i.e. for any point on line P_1P_2),
  - Ni.[P(t) – F]>0 implies that vector(P(t) – F) is pointing towards the interior of R.

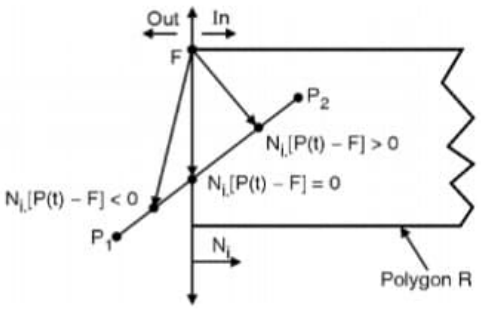


Fig. 5.3.8

- N_i .[P(t) – F] = 0 implies that vector (P(t) – F) is perpendicular to the normal.
- N_i .[P(t) – F]< 0 implies that vector (P(t) – F) is pointing away from the interior of R.

– If F is any point on the boundary of convex clipping window R, and N_i is the inward normal for the given boundary,

$$N_i.\ [P(t) - F_i],\ i \quad = \quad 1, 2, 3, \ldots..$$

– There are three possibilities:

- If the above dot product is positive, the point is inside the clipping region.
- If the above dot product is zero, the point is on the border of the clipping region.
- If the above dot product is negative, the point is outside the clipping region.

– We can formulate the equation for the intersection point of parametric line and region boundary as,

$$N_i.\ [P(t) - F_i] = 0$$

$$N_i.[P_1 + (P_2 - P_1)\ t - F_i] = 0 \qquad \text{(Parametric equation of line)}$$

$$N_i.\ P_1 + N_i.\ (P_2 - P_1)\ t - N_i.\ F_i = 0$$

$$N_i.\ [P_1 - Fi] + N_i.[P_2 - P_1]t = 0$$

– Let $D = P_2 - P_1$, that is the direction of parametric line, and

– $W_i = P_1 - F_i$,is a vector proportional to a distance from the end point of the line to the boundary point. So, $N_i.W_i + N_i.D.t = 0$.

$$N_i.D.t = -N_i.W_i$$

$$T = \frac{N_i,\ W_i}{D\ ,N_i}$$

D = 0, implies $P_2 = P_1$, means lines is a point

– $D.N_i = 0$ implies D and Ni are perpendicular. That means region boundary and line P_1P_2 are parallel to each other.

If $P_2 \neq P_1$, and if $W_i.N_i < 0$,

then point is outside = 0, then point is on the boundary > 0, then point is inside the region

– We can classify the intersection point either as potentially entering P_E ($D.N_i > 0$) or potentially leaving P_L ($D.N_i < 0$).

– Let t_E and t_L represent the value of parameter t while line enters in and leaves out the clipping region, respectively, and P_E and P_L are the corresponding intersection point with boundary. For valid line, t_E must be less than t_L, because as we move along the line, t increases.

– The line may intersect multiple boundaries while enter or leaving the clipping region (consider the line EF).

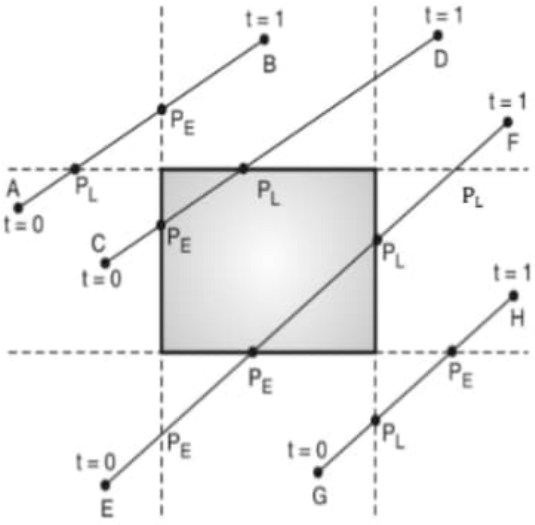


Fig. 5.3.9

- We should select P_E with the largest t value i.e. t_E = max (0, all t_E), and P_L with smallest t value i.e. t_L = min (1, all t_L).
  - If $t_L < t_E$, the line is completely outside the clipping region and hence fully rejected. Otherwise, t_E and $t_L$define the endpoints of a visible segment of line P_1P_2.
  - If $P_1'P_2'$ is the visible line segment, then the coordinates of P_1' and P_2' can be achieved as follow,

$$P_1 = P_1 + (P_2 - P_1)\, t_E$$
$$P_2 = P_1 + (P_2 - P_1)t_L$$

More precisely,

$$P_1.x = P_1.x + (P_2.x - P_1.x)\, t_E$$
$$P_1.y = P_1.y + (P_2.y - P_1.y)\, t_E$$
$$P_2.x = P_1.x + (P_2.x - P_1.x)\, t_L$$
$$P_2.y = P_1.y + (P_2.y - P_1.y)\, t_L$$

Q. State merits and demerits of Cyrus-Beck line clipping algorithm.

Advantages :

- Computation of t-intersections is cheap.
- Computation of (x, y) clip points is only done once.

Limitations :

- The algorithm doesn't consider trivial accepts/rejects.
- Best when many lines must be clipped.

Cyrus-beck line clipping algorithm :

W-18

Q. Write down Cyrus-beck line clipping algorithm. **(W-18, 6 Marks)**

Algorithm :

```
Algorithm CRYUS_BECK (P1,P2, Ni, Fi, k)
tE← 0
tL← 0
D ← P2 – P1
for i ← 1 to k do                    // k is number of polygon edges
   wi ← P1 – Fi
   if D.Ni ≠ 0then                   // Line is not a point
      t ← – (wi.Ni) / (D.Ni)
      if D.Ni>0then                  // Potential Entering
        if t >1then
           Reject the line
        else
```

```
            tE← max {t, tE}
          end
        else        // Potential Leaving
          if t < 0then
            Reject the line
          else
            tL← min {t, tL}
          end
        end
      else          //i.e. if D.Ni = 0
        if wi.Ni < 0then
          Line is trivially invisible or its invisible point so reject it
        end
      end
end

if tE> tLthen
   return NULL
else
   return P(tE) and P(tL)as a clip intersection points
end
```

5.3.2.4 Liang Barsky Line Clipping Algorithm

Q. Explain the Liang Barsky Line Clipping Algorithm.

- The liang-barsky algorithm uses the parametric equation of a line. It is also known as parametric line clipping approach.
- Cohen-Sutherland is good at trivial acceptance and rejection cases. Liang Barsky algorithm is significantly more efficient than Cohen-Sutherland when actually clipping is required.
- Let's consider the line with end points $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$. Parametric representation of line is given as,

$$x = x_0 + (x_1 - x_0) \cdot u = x_0 + u \cdot \Delta x$$

$$y = y_0 + (y_1 - y_0) \cdot u = y_0 + u \cdot \Delta y$$

Where, u is a parameter that controls the line points, $0 \leq u \leq 1$

Point is inside the clipping region only if it holds following inequalities :

$$X_{min} \leq x_0 + u \cdot \Delta x \leq X_{max} \quad \ldots(5.3.1)$$

$$Y_{min} \leq y_0 + u \cdot \Delta x \leq Y_{max} \quad \ldots(5.3.2)$$

By re writing above inequalities,

From Equation (5.3.1),

$$X_{min} - x_0 \leq u \cdot \Delta x \leq X_{max} - x_0$$

First inequality in above equation is, $X_{min} - x_0 \leq u \cdot \Delta x$,

$$-u.\Delta x \leq x_0 - X_{min} \quad \ldots(5.3.3)$$

Second inequality is, $u.\Delta x \leq X_{max} - x_0$ …(5.3.4)

Similarly, from Equation (5.3.2),

$$Y_{min} - y_0 \leq u.\Delta y \leq Y_{max} - y_0$$

First inequality in above equation is, $Y_{min} - y_0 \leq u.\ dy$,

$$-u \cdot \Delta y \leq Y_0 - Y_{min} \quad \ldots(5.3.5)$$

Second inequality is,

$$u \cdot \Delta y \leq Y_{max} - y_0 \quad \ldots(5.3.6)$$

Equation (5.3.3) to (5.3.6) can be written as,

$$P_k.u \leq q_k,\ k = 1, 2, 3, 4$$

Where,

$p_1 = -\Delta x$	$q_1 = x_0 - X_{min}$	Left
$p_2 = \Delta x$	$q_2 = X_{max} - x_0$	Right
$p_3 = -\Delta y$	$q_3 = y_0 - Y_{min}$	Bottom
$p_4 = \Delta y$	$q_4 = Y_{max} - y_0$	Top

To compute the final visible segment, we do the following calculations:

- If $p_k = 0$, line is parallel to the boundary of the clipping region. With $p_k = 0$, if $q_k < 0$, the line is completely outside the clipping region and line can be completely clipped. When $p_k < 0$ the line proceeds outside to inside the clip window for a k^{th} boundary. Similarly, when $p_k > 0$, the line proceeds inside to outside for the k^{th} boundary. Value of k = 1, 2, 3, 4 corresponds to left, right, bottom and top edge.
- The line can enter or leave the region maximum two times. So for entering intersections, we should select the highest value of parameter u, and for leaving intersections, we should select the minimum value of the parameter. For non-zero value of p_k, we calculate $r_k = q_k / p_k$ for k = 1, 2, 3, 4
- For each value of k, such that $p_k < 0$, find $r_k$and, $u_1 = \max(0, r_k)$.

  For each value of k, such that $p_k > 0$, find $r_k$and, $u_2 = \min(1, r_k)$.
- Where, u_1 is the parameter value of the line where it enters in the clipping region, and u_2 is the parameter value of the line where it leaves the clipping region. If the value of u_1 is greater than u_2, the line is completely outside. Otherwise visible segment $P_1'P_2'$ of the line is computed as,

$$P_1' = P_1 + (P_2 - P_1) \cdot u_1$$

$$P_2' = P_1 + (P_2 - P_1) \cdot u_2$$

Advantages :

- More efficient.
- Only requires one division to update u_1 and u_2.
- It is more efficient than Cohen Sutherland since intersection calculation is reduced.
- It requires only one division and window intersection is computed only once
- Works for rectangular clipping region in 2D and 3D.
- Cohen Sutherland algorithm may perform repeated computation even though the line is completely outside the window.

Example 5.3.5 : Using Liang Barsky algorithm, find the clipping coordinates of the line segment with end coordinates, A (– 10, 50) and B(30, 80) against the window ($X_{min} = -30$, $Y_{min} = 10$) ($X_{max} = 20$, $Y_{max} = 60$).

Solution :

Here, $Xw_{min} = -30$, $Yw_{min} = 10$, $Xw_{max} = 20$, $Yw_{max} = 60$.

End points of line are A = $(x_1, y_1) = (-10, 50)$ and B = $(x_2, y_2) = (30, 80)$.

Let us compute p_i, q_i and r_i for i = 1, 2, 3, 4

$$p_1 = -(\Delta x) = -(x_2 - x_1)$$
$$= -(30 - (-10)) = -40$$
$$p_2 = \Delta x = 40$$
$$p_3 = -\Delta y = -(y_2 - y_1)$$
$$= -(80 - 50) = -30$$
$$p_4 = \Delta y = 30$$

None of the p_i is zero, so line is neither horizontal, nor vertical. Let us compute q_i.

$$q_1 = x_1 - xw_{min} = -10 + 30 = 20$$
$$q_2 = xw_{max} - x_1 = 20 + 10 = 30$$
$$q_3 = y_1 - yw_{min} = 50 - 10 = 40$$
$$q_4 = yw_{max} - y_1 = 60 - 50 = 10$$
$$r_1 = \frac{q_1}{p_1} = \frac{20}{-40} = -0.5$$
$$r_2 = \frac{q_2}{p_2} = \frac{30}{40} = 0.75$$
$$r_3 = \frac{q_3}{p_3} = \frac{40}{-30} = -1.33$$
$$r_4 = \frac{q_4}{p_4} = \frac{10}{30} = 0.33$$
$$u_1 = \max(0, \text{all } r_i \text{ having } p_i < 0)$$
$$= \max(0, r_1, r_3)$$
$$= \max(0, -0.5, -1.33) = 0$$

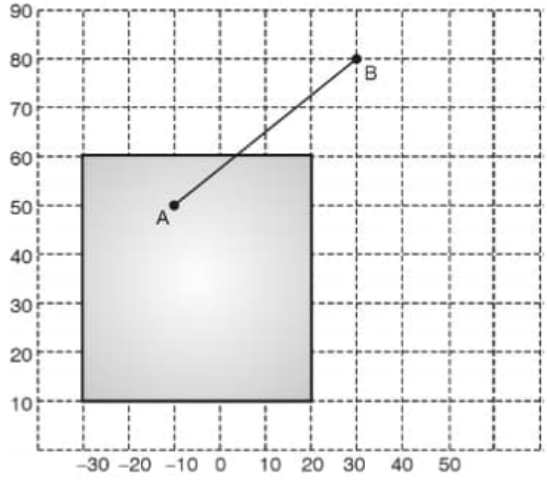


Fig. P. 5.3.5

$$u_2 = \min(1, \text{all } r_i \text{ having } P_i > 0)$$

$$= \min(1, r_2, r_4)$$

$$= \min(1, 0.75, 0.33) = 0.33$$

$$x_1' = x_1 + \Delta x \cdot u_1 = -10 + (40) \times (0) = -10$$

$$y_1' = y_1 + \Delta y \cdot u_1 = 50 + (30) \times (0) = 50$$

$$x_2' = x_1 + \Delta x \cdot u_2$$

$$= -10 + (40) \times (0.33)$$

$$= -10 + 13.2 = 3.2$$

$$y_2' = y_1 + \Delta y \cdot u_2$$

$$= 50 + (30) \times (0.33)$$

$$= 50 + 9.9 = 59.9$$

∴ Visible line segment is (– 10, 50) to (3.2, 59.9)

Original line Segments	Visible line segments
A(– 10, 50) and B(30, 80)	A'(– 10, 50) and B'(3.2, 59.9)

Example 5.3.6 : Explain an algorithm which uses parametric equation for the purpose of line clipping, using the same algorithm, find the coordinates of the line segment A(10, 10), B(70, 40), after it is clipped against a window with two diagonal vertices at (20, 20) and (40, 50).

Solution :

Here, $Xw_{min} = 20$, $Yw_{min} = 20$, $Xw_{max} = 40$, $Yw_{max} = 50$.

End points of line are A = $(x_1, y_1) = (10, 10)$ and B = $(x_2, y_2) = (70, 40)$.

Let us compute p_i, q_i and r_i for i = 1, 2, 3, 4

$$p_1 = -(\Delta x) = -(x_2 - x_1)$$

$$= -(70 - 10) = -60$$

$$p_2 = \Delta x = 60$$

$$p_3 = -\Delta y = -(y_2 - y_1)$$

$$= -(40 - 10) = -30$$

$$p_4 = \Delta y = 30$$

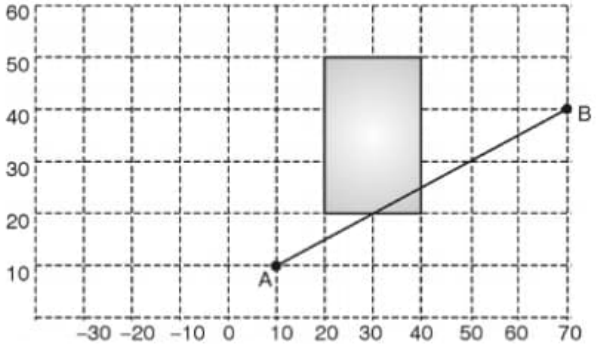


Fig. P. 5.3.6

None of the p_i is zero, so line is neither horizontal, nor vertical. Let us compute q_i.

$$q_1 = x_1 - xw_{min}$$

$$= 10 - 20 = -10$$

$$q_2 = xw_{max} - x_1$$

$$= 40 - 10 = 30$$

$$q_3 = y_1 - yw_{min}$$
$$= 10 - 20 = -10$$
$$q_4 = yw_{max} - y_1$$
$$= 50 - 10 = 40$$
$$r_1 = \frac{q_1}{p_1} = \frac{-10}{-60} = 0.167$$
$$r_2 = \frac{q_2}{p_2} = \frac{30}{60} = 0.5$$
$$r_3 = \frac{q_3}{p_3} = \frac{-10}{-30} = 0.333$$
$$r_4 = \frac{q_4}{p_4} = \frac{40}{30} = 1.333$$
$$u_1 = \max(0, \text{all } r_i \text{ having } p_i < 0)$$
$$= \max(0, r_1, r_3)$$
$$= \max(0, 0.167, 0.333) = 0.333$$
$$u_2 = \min(1, \text{all } r_i \text{ having } P_i > 0)$$
$$= \min(1, r_2, r_4)$$
$$= \min(1, 0.5, 1.333) = 0.5$$
$$x_1' = x_1 + \Delta x \cdot u_1 = 10 + (60) \times (0.333)$$
$$= 10 + 19.98 = 28.98$$
$$y_1' = y_1 + \Delta y \cdot u_1 = 10 + (30) \times (0.333)$$
$$= 10 + 10 = 20$$
$$x_2' = x_1 + \Delta x \cdot u_2$$
$$= 10 + (60) \times (0.5)$$
$$= 10 + 30 = 40$$
$$y_2' = y_1 + \Delta y \cdot u_2$$
$$= 10 + (30) \times (0.5)$$
$$= 10 + 15 = 25$$

∴ Visible line segment is (28.98, 20) to (40, 25)

Original line Segments	Visible line segments
A(10, 10) and B(70, 40)	A(28.98, 20) and B'(40, 25)

Example 5.3.7 : Clip the line using Liang Barsky algorithm against the window with (Xw_{min}, Yw_{min}) = (0,0) and (Xw_{max}, Yw_{max})=(100,50). Line end points are A(10,10) and B(110, 40).

Solution :

Here, $Xw_{min} = 0$, $Yw_{min} = 0$, $Xw_{max} = 100$, $Yw_{max} = 50$.

End points of line are A = (x_1, y_1) = (10, 10) and B = (x_2, y_2) = (110, 40).

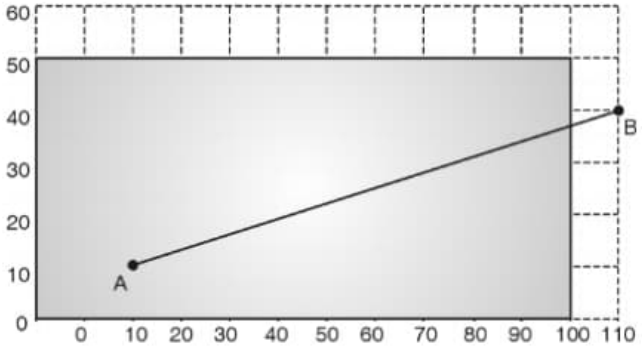


Fig. P. 5.3.7

Let us compute p_i, q_i and r_i for i = 1, 2, 3, 4

$$p_1 = -(\Delta x) = -(x_2 - x_1)$$
$$= -(110 - 10) = -100$$
$$p_2 = \Delta x = 100$$
$$p_3 = -\Delta y = -(y_2 - y_1)$$
$$= -(40 - 10) = -30$$
$$p_4 = \Delta y = 30$$

None of the p_i is zero, so line is neither horizontal, nor vertical. Let us compute q_i.

$$q_1 = x_1 - xw_{min} = 10 - 0 = 10$$
$$q_2 = xw_{max} - x_1 = 100 - 10 = 90$$
$$q_3 = y_1 - yw_{min} = 10 - 0 = 10$$
$$q_4 = yw_{max} - y_1 = 50 - 10 = 40$$
$$r_1 = \frac{q_1}{p_1} = -\frac{10}{100} = -0.1$$
$$r_2 = \frac{q_2}{p_2} = \frac{90}{100} = 0.9$$
$$r_3 = \frac{q_3}{p_3} = \frac{10}{-30} = -0.33$$
$$r_4 = \frac{q_4}{p_4} = \frac{40}{30} = 1.33$$
$$u_1 = \max(0, \text{all } r_i \text{ having } p_i < 0) = \max(0, r_1, r_3) = \max(0, -0.1, -0.33) = 0$$
$$u_2 = \min(1, \text{all } r_i \text{ having } P_i > 0)$$
$$= \min(1, r_2, r_4)$$
$$= \min(1, 0.9, 1.33) = 0.9$$
$$u_1 = 0;\ \text{So, } (x_1', y_1') = (x_1, y_1)$$
$$x_2' = x_1 + \Delta x \cdot u_2 = 10 + (100) \times (0.9)$$
$$= 10 + 90 = 100$$
$$y_2' = y_1 + \Delta y \cdot u_2 = 10 + (30) \times (0.9)$$
$$= 10 + 27 = 37$$

∴ Visible line segment is (10, 10) to (100, 37)

Original line Segments	Visible line segments
A(10, 10) and B(110, 40)	A′(10, 10) and A′(100, 37)

Example 5.3.8 : Determine the visible span of line AB against given clipping window using Liang Barsky algorithm.

Solution :

Let us consider line end points $(x_1, y_1) = (11, 6)$ and $(x_2, y_2) = (11, 10)$. From clipping window position, $Xw_{min} = 1$, $Xw_{max} = 9$, $Yw_{min} = 2$ and $Yw_{max} = 8$ Let us compute p_i, q_i and r_i,

$$P_1 = -\Delta x = -(x_2 - x_1)$$

$$= -(11 - 11) = 0$$

$$P_2 = \Delta x = (x_2 - x_1)$$

$$= 11 - 11 = 0$$

$$q_1 = x_1 - xw_{min}$$

$$= 11 - 1 = 10$$

$$q_2 = xw_{max} - x_1$$

$$= 9 - 11 = -2$$

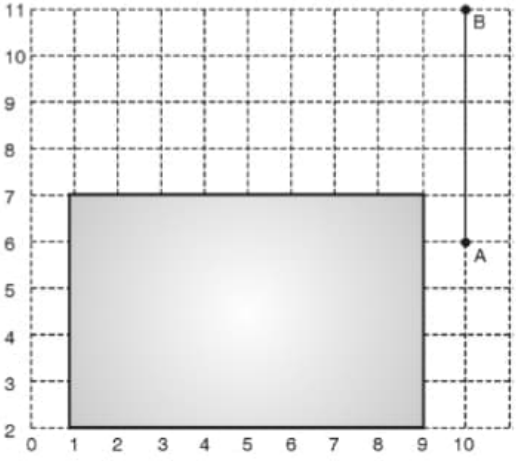


Fig. P. 5.3.8

Here, $p_1 = 0$ so line is vertical, and $q_1 < 0$ implies that line is right side of the right boundary of clipping window. So line is trivially rejected.

Example 5.3.9 : Determine the visible span of line AB against given clipping window using Liang Barsky algorithm.

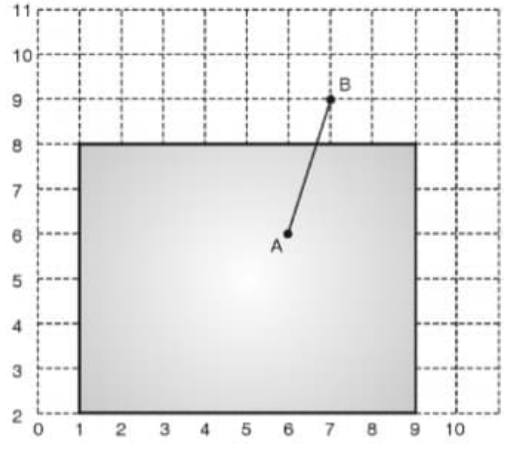


Fig. P. 5.3.9

Solution :

Let us consider line end points $(x_1, y_1) = (6,6)$ and$(x_2, y_2) = (8, 9)$ and From clipping window, $xw_{min} = 1$, $xw_{max} = 9$, $yw_{min} = 2$ and $yw_{max} = 8$. Let us compute p_i, q_i and r_i.

$$p_1 = -\Delta x = -(x_2 - x_1) = -(8-6) = -2$$

$$p_2 = \Delta x = 2$$

$$p_3 = -\Delta y = -(y_2 - y_1) = -(9-6) = -3$$

$$p_4 = \Delta y = 3$$

None of the p_i is zero, so line is neither horizontal nor vertical. Let us check if it passes through window.

$$q_1 = x_1 - xw_{min} = 6 - 1 = 5$$

$$q_2 = xw_{max} - x_1 = 9 - 6 = 3$$

$$q_3 = y_1 - yw_{min} = 6 - 2 = 4$$

$$q_4 = yw_{max} - y_1 = 8 - 6 = 2$$

We compute r_i as follow to find intersection of line with window boundary.

$$r_i = \frac{q_i}{p_i},\ i = 1, 2, 3, 4$$

$$r_1 = \frac{q_1}{p_1} = -\frac{5}{2} = -2.5$$

$$r_2 = \frac{q_2}{p_2} = \frac{3}{2} = 1.5$$

$$r_3 = \frac{q_3}{p_3} = -\frac{4}{3} = -1.33$$

$$r_4 = \frac{q_4}{p_4} = \frac{2}{3} = 0.66$$

u_1 = max (0, all r_i having $p_i < 0$) = max (0, r_1, r_3) = max (0, – 2.5, – 1.33) = 0

u_2 = min (1, all r_i having $p_i > 0$) = min (1, r_2, r_4) = min (1, 1.5, 0.66) = 0.66

Let us consider (x_1', y_1') and (x_2', y_2') are the end points of clipped line. From parametric equation of line,

$$x_1' = x_1 + \Delta x.u_1 = 6 + 2 \times 0 = 6$$

$$y_1' = y_1 + \Delta y.u_1 = 6 + 2 \times 0 = 6$$

$$x_2' = x_1 + \Delta x.u_2 = 6 + 2 \times 0.66 = 7.33$$

$$y_2' = y_1 + \Delta y.u_2 = 6 + 3 \times 0.66 = 8$$

Visible line segment is (6, 6) to (7.33, 8)

Original line Segments	Visible line segments
A(6, 6) and B(8, 9)	A'(6, 6) and B'(7.33, 8)

5.4 Polygon Clipping

5.4.1 Introduction to Polygon Clipping

- Clipping of polygon requires more attention.
- The output of line clipping is either point or line.
- The output of polygon clipping may be point, line or polygon.
- Following issues arise in polygon clipping. All in all, polygon clipping is more complex compared to line clipping.
- In polygon clipping, each edge needs to be tested against each edge of the clipping region. This process may add new edges, may delete existing edges, may retain or divide the edges.

Issues in polygon clipping :

Q. State the issues in polygon clipping.

1. Clipping a line segment yields at the most one-line segment.
2. Clipping of convex polygon can yield at most one polygon (Fig. 5.4.1(a)).
3. Clipping of concave polygon can yield multiple polygons (Fig. 5.4.1(b)).
4. For the same input polygon, a number of output edges may be different depending on its orientation and spatial location (Fig. 5.4.1(c)).
5. May need to add new edges.
6. The edge may be discarded or divided.

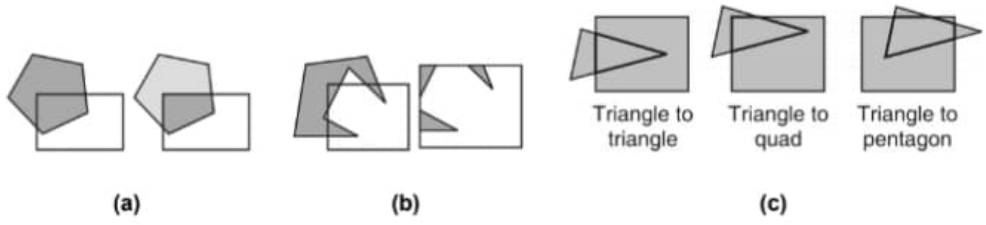


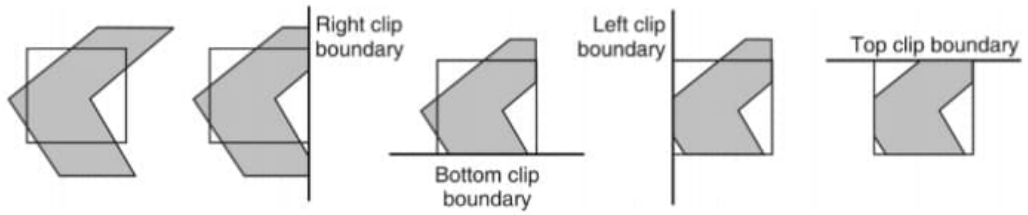
Fig. 5.4.1 : Cases of polygon clipping

5.4.2 Sutherland-Hodgeman Algorithm

Q. Explain Sutherland-Hodgeman Polygon Clipping Algorithm.

- Sutherland-Hodgman polygon-clipping algorithm uses a divide-and-conquer strategy. Divide and conquer method divides the large problem into small subproblems, solve them recursively and combine the results to build the solution of parent problem.
- Sutherland-Hodgeman algorithm works in four stages. In each stage, each edge of subject polygon is tested against the edge of the clipping rectangle.
- To accomplish this task,we process all vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we could first clip the polygon against any of the edge (i.e left, right, top or bottom) to produce a new sequence of vertices.

– This new set of vertices then successively passed to remaining boundary clippers. At each step, a new sequence of output vertices is generated and passed to the next clipper. Fig. 5.4.2 shows the simulation of the algorithm.



(a) Original polygon **(b) Output of right clipper** **(c) Output of bottom clipper** **(d) Output of left clipper** **(e) Output of right clipper**

Fig. 5.4.2 : Stages of polygon clipping

When we traverse along the polygon boundary, one of the following four cases occurs:

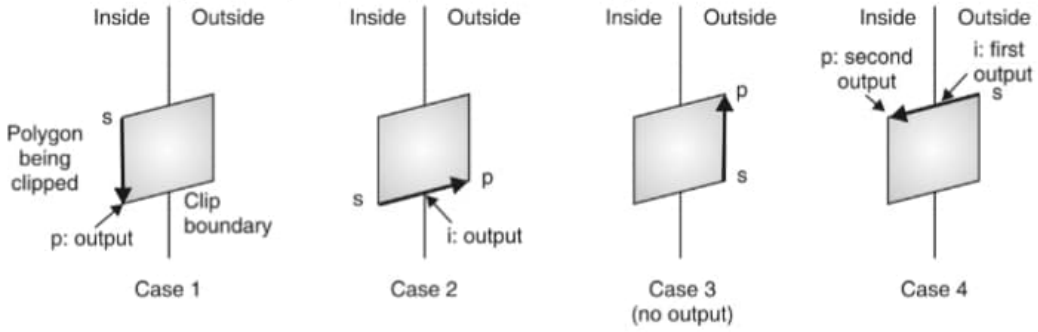


Fig. 5.4.3 : Output for various clippers

Case 1 :

While traversing along the boundary of the subject polygon, if the movement is from inside to inside i.e when the source(s) and destination(p) vertices of the edge are inside the clipping region, we output destination vertex (p) only.

Case 2 :

When the movement is from inside to outside i.e when source vertex is inside the clipping region and destination vertex is outside the clipping region, the polygon edge intersects the clipping boundary. In that case, we output intersection point *i*only.

Case 3 :

When the movement is from outside to outside i.e when source vertex is outside the clipping region and destination vertex is also outside the clipping region, nothing is selected for output.

Case 4 :

When the movement is from outside to inside i.e when source vertex is outside the clipping region and destination vertex is inside the clipping region, the polygon edge intersects the clipping boundary. In that case, we output both, the intersection point and the destination vertex.

Q. What are the issues involved in Sutherland-Hodgeman polygon clipping algorithm? How to handle them.

Output list of the first clipper is passed to next clipper and processed in the same way. Convex polygons are correctly clipped by this algorithm, but concave polygons sometimes may generate extraneous lines as shown in Fig. 5.4.4. Edge ab was not part of original polygon, but it has been now part of output polygon.

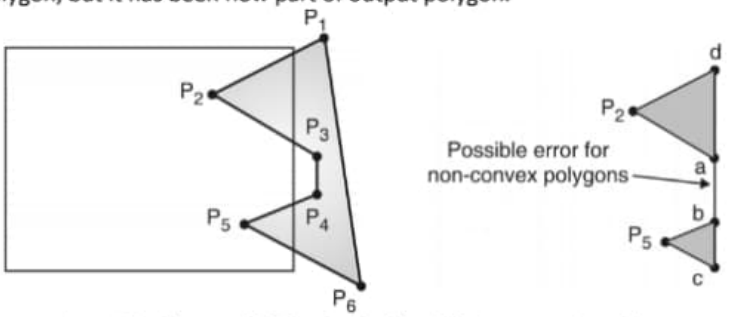


Fig. 5.4.4 : Issue with the Sutherland-Hodgeman algorithm

This problem can be handled in many ways. One way is to convert the concave polygon to multiple convex polygon as shown in Fig. 5.4.5. The process of converting a polygon to a set of trianglesis known as **tessellation**. Then process each convex polygon individually.

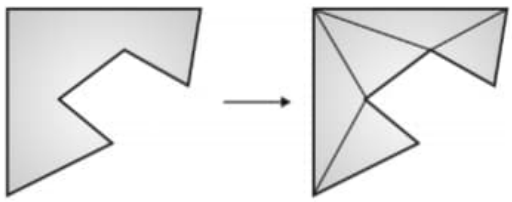


Fig. 5.4.5 : Tessellation process

Another way of solving this problem is to tag the different new vertices as follow.

Call α the new vertex generated on the transition from IN to OUT. Call β the new vertex generated on the transition from OUT to IN. Starting from OUT, connect α and β as soon α as is generated. This process successfully eliminates the extraneous edge.

This algorithm is well suited for hardware implementation and can easily be parallelized on multiple processors, each processor works as a clipper. However, this method works only for convex clipping region.

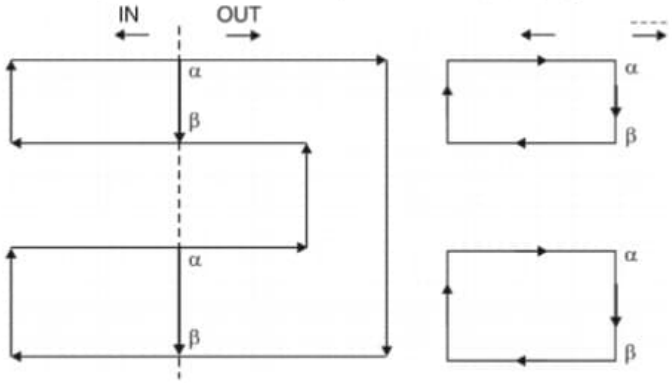


Fig. 5.4.6 : Solving issue of Sutherland-Hodgeman algorithm

5.5 Text Clipping

W-18

Q. Explain different types of Text clipping in brief. **(W-18, 4 Marks)**

- Text clipping is not much frequent operation as like line or polygon clipping. However, some graphics packages provide the functionality to clip the text in the scene.
- Choice of text clipping technique depends on application and type of character generation method. We will discuss three methods of text clipping.

5.5.1 All or None String Clipping

Q. Write a note on All or None String Clipping approach.

- The simplest way to perform the text clipping is all or none string clipping. In this technique, either entire text is displayed or it is fully clipped.
- The test of deciding full membership of text in the clipping window is very simple. We compare the bounding box of a string with boundaries of the clipping window.
- If minimal rectangle covering text is completely within clipping window, then accept the string otherwise clip it fully.
- This method is very fast.
- Fig. 5.5.1 shows the scenario before and after text clipping.

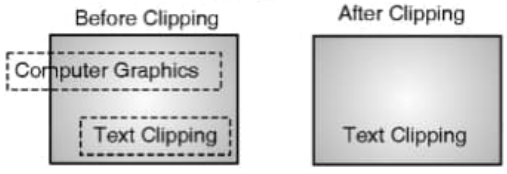
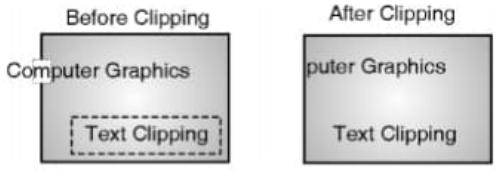


Fig. 5.5.1 : All or none string clipping

5.5.2 All or none Character Clipping

Q. Write a note on all or None Character Clipping approach.

- All or none string clipping is a very rigid strategy. It clips the entire text string even if onlysingle character is outside the window boundary.
- Another alternative to this is all or none character clipping method. In this method, only those characters of the string are clipped which are not completely inside the window.
- In this method, the extent of individual character is compared against clipping window boundaries.
- The text before clipping and after clipping is depicted in Fig. 5.5.2.



(a) Before clipping **(b) After clipping**

Fig. 5.5.2 : All or none character clipping

5.5.3 Individual Character Clipping

Q. Write a note on Individual Character Clipping approach.

- Another approach of text clipping is to clip the part of the character which is outside the clipping window.
- This case is identical to handling lines. We need to decide the part of the character which is visible. Outline characters formed with line can be clipped using this approach.
- Character generated using bitmap technique is clipped in a different way. In such a case, individual bit location is compared and clipped if necessary.
- The method is depicted in Fig. 5.5.3.

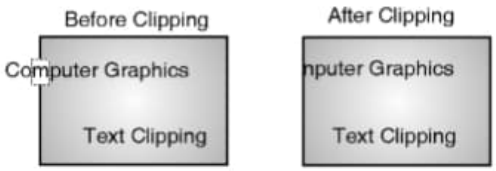


Fig. 5.5.3 : Individual character clipping

5.6 Lab Programs

Program 5.6.1 : Write a program to clip line using the following algorithm: Cohen-Sutherland Algorithm. **(Lab 12)**

Solution :

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#define MAX 20

enum { TOP = 0x1, BOTTOM = 0x2, RIGHT = 0x4, LEFT = 0x8 };
enum { FALSE, TRUE };
typedef unsigned int outcode;

outcode FindOutCode(int, int, int, int, int, int);
void CohenSutherland (double, double, double, double, double, double, double, double);

void main()
{
   int x1, y1, x2, y2;
   int XWmin, YWmin, XWmax, YWmax;
   int gdriver = DETECT, gm;
```

```
    // Scan line end points
    printf("Enter x1 :-->  ");
    scanf ("%d", &x1);
    printf("Enter y1 :-->  ");
    scanf ("%d", &y1);
    printf("Enter x2 :-->  ");
    scanf ("%d", &x2);
    printf("Enter y2 :-->  ");
    scanf ("%d", &y2);

    // Scan window coordinates
    printf("\nEnter XW_min :--> ");
    scanf("%d", &XWmin);
    printf("Enter YW_min :--> ");
    scanf("%d", &YWmin);
    printf("\nEnter XW_max :--> ");
    scanf("%d", &XWmax);
    printf("Enter YW_max :--> ");
    scanf("%d", &YWmax);

    initgraph (&gdriver, &gm, "");

    rectangle (XWmin, YWmin, XWmax, YWmax);
    line (x1, y1, x2, y2);

    getch();
    cleardevice();

    rectangle (XWmin, YWmin, XWmax, YWmax);
    CohenSutherland(x1, y1, x2, y2, XWmin, YWmin, XWmax, YWmax);

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

```
outcode FindOutCode(int x, int y, int XWmin, int YWmin, int XWmax, int YWmax)
{
   outcode oc = 0;

   if (y > YWmax)
   oc |= TOP;
   else if (y < YWmin)
   oc |= BOTTOM;

   if (x > XWmax)
   oc |= RIGHT;
   else if (x < XWmin)
   oc |= LEFT;

   return oc;
}

void CohenSutherland (double x1, double y1, double x2, double y2,
     double XWmin, double YWmin, double XWmax, double YWmax)
{
   int accept;
   int done;
   outcode outcode1, outcode2;

   accept = FALSE;
   done = FALSE;

   outcode1 = FindOutCode (x1, y1, XWmin, YWmin, XWmax, YWmax);
   outcode2 = FindOutCode (x2, y2, XWmin, YWmin, XWmax, YWmax);
   do
   {
   if (outcode1 == 0 && outcode2 == 0)
   {
      accept = TRUE;
      done = TRUE;
```

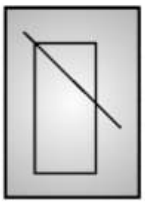
```
}
else if (outcode1 & outcode2)
{
   done = TRUE;
}
else
{
   double x, y;
   int outcode_ex = outcode1 ? outcode1 : outcode2;
   if (outcode_ex & TOP)
   {
  x = x1 + (x2 - x1) * (YWmax - y1) / (y2 - y1);
  y = YWmax;
   }

   else if (outcode_ex & BOTTOM)
   {
  x = x1 + (x2 - x1) * (YWmin - y1) / (y2 - y1);
  y = YWmin;
   }
   else if (outcode_ex & RIGHT)
   {
  y = y1 + (y2 - y1) * (XWmax - x1) / (x2 - x1);
  x = XWmax;
   }
   else
   {
  y = y1 + (y2 - y1) * (XWmin - x1) / (x2 - x1);
  x = XWmin;
   }
   if (outcode_ex == outcode1)
   {
  x1 = x;
  y1 = y;
  outcode1 = FindOutCode (x1, y1, XWmin, YWmin, XWmax, YWmax);
```

```
      }
      else
      {
     x2 = x;
     y2 = y;
     outcode2 = FindOutCode (x2, y2, XWmin, YWmin, XWmax, YWmax);
      }
   }
   } while (done == FALSE);
   if (accept == TRUE)
   line (x1, y1, x2, y2);
}
```

Output :

```
Enter x1 :-->  30
Enter y1 :-->  30
Enter x2 :-->  120
Enter y2 :-->  120

Enter XW_min :--> 40
Enter YW_min :--> 40

Enter XW_max :--> 100
Enter YW_max :--> 170
```

Read coordinates



After clipping

Before clipping

Program 5.6.2 : Write a program to clip line using Midpoint subdivision algorithm. **(Lab 13)**

Solution :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<graphics.h>

typedef struct coordinate
{
   int x,y;
```

```
   char code[4];
}PT;

int XWmin, YWmin, XWmax, YWmax;   // Coordinates of clipping window
int visibility (PT p1,PT p2);
PT FindOutCode(PT p);
PT resetendpt (PT p1,PT p2);
void MidpointSubdivison(PT, PT);

void main()
{
   int gdriver=DETECT, gm,v;
   PT p1,p2,ptemp;
   initgraph(&gdriver,&gm,"");
   cleardevice();

   // Scan Line endpoints
   printf("Enter x1 :--> ");
   scanf("%d", &p1.x);
   printf("Enter y1 :--> ");
   scanf("%d", &p1.y);
   printf("Enter x2 :--> ");
   scanf("%d", &p2.x);
   printf("Enter y2 :--> ");
   scanf("%d", &p2.y);

   // Scan clipping window coordinates
   printf("\nEnter XW_min :--> ");
   scanf("%d", &XWmin);
   printf("Enter YW_min :--> ");
   scanf("%d", &YWmin);
   printf("Enter XW_max :--> ");
   scanf("%d", &XWmax);
   printf("Enter YW_max :--> ");
   scanf("%d", &YWmax);
```

```
    cleardevice();
    rectangle(XWmin, YWmin, XWmax, YWmax);
    getch();
    line(p1.x, p1.y, p2.x, p2.y);
    getch();

    cleardevice();
    rectangle (XWmin, YWmin, XWmax, YWmax);

    MidpointSubdivison(p1, p2);
    getch();
    closegraph();
}

void MidpointSubdivison(PT p1,PT p2)
{
    PT mid;
    int v;
    p1=FindOutCode(p1);
    p2=FindOutCode(p2);
    v=visibility(p1,p2);
    switch(v)
    {
      case 0: // Fully visible
         line(p1.x, p1.y, p2.x, p2.y); break;
      case 1: // Fully outside
         break;
      case 2: // Partially visible
         mid.x = p1.x + (p2.x - p1.x) / 2;
         mid.y = p1.y + (p2.y - p1.y) / 2;
         MidpointSubdivison(p1,mid);
         mid.x = mid.x + 1;
         mid.y = mid.y + 1;
         MidpointSubdivison(mid, p2);
         break;
```

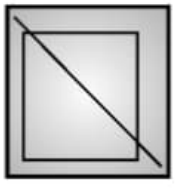
```
    }
}

PT FindOutCode(PT p)
{
    PT ptemp;
    if(p.y <= YWmin)
      ptemp.code[0] = '1';
    else
      ptemp.code[0] = '0';
    if(p.y >= YWmax)
      ptemp.code[1] = '1';
    else
      ptemp.code[1] = '0';
    if(p.x >= XWmax)
      ptemp.code[2] = '1';
    else
      ptemp.code[2] = '0';
    if(p.x <= XWmin)
      ptemp.code[3] = '1';
    else
      ptemp.code[3] = '0';

    ptemp.x = p.x;
    ptemp.y = p.y;

    return(ptemp);
}

int visibility (PT p1,PT p2)
{
    int i, flag = 0;
    for(i=0; i<4; i++)
    {
      if((p1.code[i] != '0') || (p2.code[i] != '0'))
```

```
    flag = 1;
  }
  if(flag == 0)
    return(0);

  for(i=0; i<4; i++)
  {
    if((p1.code[i] == p2.code[i]) && (p1.code[i] == '1'))
    flag = 0;
  }
  if(flag == 0)
    return(1);

  return(2);
}
```

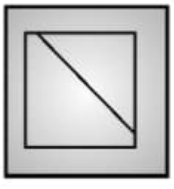
Output :

```
Enter x1 :--> 20
Enter y1 :--> 20
Enter x2 :--> 200
Enter y2 :--> 200

Enter XW_min :--> 30
Enter YW_min :--> 40
Enter XW_max :--> 170
Enter YW_max :--> 190
```

Read coordinates



After clipping



Before clipping

Program 5.6.3 : Write a program to clip polygon using Sutherland-Hodgeman Algorithm. **(Lab 14)**

Solution :

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


void dda(int, int, int, int, int, int, int, int);
```

```
void main()
{
   int gdriver = DETECT, gm;
   int x1, x2, y1, y2, xRec, yRec, n;
   int l, b, yMin, yMax, xMin, xMax;
   float m;
   int Lines[10][4],i,j, flag = 0,in = 0;

   initgraph(&gdriver,&gm,"");
   int XWmin, YWmin, XWmax, YWmax;

   printf("\nEnter XWmin :--> ");
   scanf("%d",&XWmin);
   printf("Enter YWmin :--> ");
   scanf("%d",&YWmin);
   printf("Enter XWmax :--> ");
   scanf("%d",&XWmax);
   printf("Enter YWmax :--> ");
   scanf("%d",&YWmax);

   cleardevice();

   rectangle(XWmin, YWmin, XWmax, YWmax);
   getch();
   cleardevice();
```

```
printf("\nEnter Number of Polygon Sides (Min. 3) :--> ");
scanf("%d",&n);

for(i=0; i<n; i++)
{
  printf("Enter x%d :--> ", i+1);
  scanf("%d", &Lines[i][0]);
  printf("Enter y%d :--> ", i+1);
  scanf("%d", &Lines[i][1]);

  Lines[i][2] = Lines[0][0];
  Lines[i][3] = Lines[0][1];
  if(i!=0)
  {
     Lines[i-1][2] = Lines[i][0];
     Lines[i-1][3] = Lines[i][1];
  }
}

cleardevice();
rectangle(XWmin, YWmin, XWmax, YWmax);

for(i=0; i<n; i++)
{
  line(Lines[i][0], Lines[i][1], Lines[i][2], Lines[i][3]);
}
getch();
```

```
cleardevice();
rectangle(XWmin, YWmin, XWmax, YWmax);

for(i=0; i<n; i++)
{
  flag = 0;
  x1 = Lines[i][0];
  x2 = Lines[i][2];
  y1 = Lines[i][1];
  y2 = Lines[i][3];

  // Fully Visible
  if(x1 >= XWmin && x1 <= XWmax && y1 >= YWmin && y1 <= YWmax)
     flag++;
  if(x2 >= XWmin && x2 <= XWmax && y2 >= YWmin && y2 <= YWmax)
     flag++;

  switch(flag)
  {
     case 0:
       break;

     case 1:
       if(x2 >= XWmin && x2 <= XWmax && y2 >= YWmin && y2 <= YWmax)
       {
          dda(x1, y1, x2, y2, XWmin, YWmin, XWmax, YWmax);
       }
```

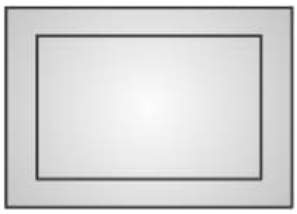
```
            else
            {
               dda(x2, y2, x1, y1, XWmin, YWmin, XWmax, YWmax);
            }
            break;
          case 2:
            line(x1, y1, x2, y2);
            break;
       }
    }
    getch();
    closegraph();
}

void dda(int x1,int y1,int x2,int y2,int XWmin,int YWmin,int XWmax,int YWmax)
{
    float dx, dy;
    float steps, x = x1, y = y1;
    int i = 0;
    dx = x2 - x1;
    dy = y2 - y1;

    if(abs(dx) >= abs(dy))
      steps = abs(dx);
    else
      steps = abs(dy);
```

```
    dx = dx / steps;
    dy = dy / steps;

    while(i++ <= steps)
    {
        if(x >= XWmin && x <= XWmax && y >= YWmin && y <= YWmax)
        {
      line(x,y,x2,y2);
          return;
        }
        x = x + dx;
        y = y + dy;
    }
}
```

Output :

```
Enter XWmin :--> 40
Enter YWmin :--> 40
Enter XWmax :--> 400
Enter YWmax :--> 300
```

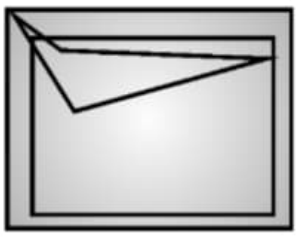
(a) Read window coordinate



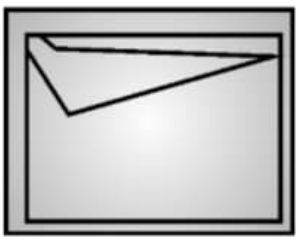
(b) Display window

```
Enter Number of Polygon Sides (Min. 3) :--> 4
Enter x1 :--> 10
Enter y1 :--> 10
Enter x2 :--> 100
Enter y2 :--> 150
Enter x3 :--> 400
Enter y3 :--> 70
Enter x4 :--> 80
Enter y4 :--> 60
```

(c) Read coordinates of polygon vertices



(d) Polygon before clipping



(e) Polygon after clipping

Review Questions

Topic : Window to Viewport Transformation

Q. 1 Explain the working principle of scan conversion approach and analytical approach.

Q. 2 Define the terms : window, viewport.

Q. 3 Define the process : Windowing, Clipping.

Q. 4 Describe viewing transformation.

Q. 5 Describe normalization transformation.

Q. 6 Derive transformation matrix for 2D viewing transformation.

Q. 7 Give the transformation matrix for window-to-viewport transformation.

Q. 8 Discuss the effect of different size and position of window and viewport in viewing.

Q. 9 What are the categories of any line segment for its visibility? How they are identified in the Cohen Sutherland line clipping algorithm?

Q. 10 Find the normalization transformation matrix that maps a window whose lower. left corner is at (2, 2) and upper right corner is at (10, 10) onto :

(i) Viewport which is entire normalized device screen.

(ii) Viewport which has a lower left corner at (0.5, 0.5) and upper right corner is at (1, 1).

Topic : Line Clipping Algorithms

Q. 1 Write Cohen-Sutherland line clipping algorithm.

Q. 2 Describe region codes.

Q. 3 Explain mid-point subdivision line clipping algorithm.

Q. 4 Describe the Cyrus-Beck line clipping algorithm.

Q. 5 Write Liang-Barsky line clipping algorithm.

Q. 6 List out merits and demerits of Cohen-Sutherland Algorithm.

Q. 7 State merits and demerits of the Cyrus-Beck line clipping algorithm.

Q. 8 What are the advantages of Liang Barsky algorithm?

Q. 9 Given a window whose lower left corner is at (–3, 1) and upper right corner is at (2, 6). Find the region codes for the endpoints of following line segments.

(i) A(–4, 7), B(–2, 10)

(ii) A(–4, 2), B(–1, 7)

(iii) A(–2, 3), B(1, 2)

(iv) A(1, –2), B(3, 3)

Q. 10 Illustrate to clipping of a line segment joining two endpoints A(-1, 5) and B(3, 8) using midpoint subdivision algorithm. Given that clipping, window having a lower left corner at (-3, 1) and upper right corner at (2, 6).

Q. 11 Write a C program to clip the line using the Cohen-Sutherland line clipping algorithm.

Q. 12 Write a C program to clip the line using the Mid-point subdivision line clipping algorithm.

Topic : Polygon Clipping Algorithms

Q. 1 Discuss the issues involved in polygon clipping.

Q. 2 What are the issues involved in the Sutherland-Hodgeman polygon clipping algorithm? How to handle them.

Q. 3 Describe the Sutherland-Hodgeman polygon clipping algorithm.

Q. 4 Write the Sutherland-Hodgeman polygon clipping algorithm.

Q. 5 Illustrate the Sutherland-Hodgeman polygon clipping algorithm with an example.

Q. 6 Write a C program to clip the polygon using the Sutherland-Hodgeman polygon clipping algorithm.

Topic : Text Clipping

Q. 1 State and explain types of text clipping.

Q. 2 Write a note on All or Note String clipping method.

Q. 3 Write a note on All or Note Character clipping method.

Q. 4 Write a note on Individual Character clipping method.

Chapter Ends...

❑❑❑

CHAPTER 6

Unit V

Introduction to Curves

Syllabus

Curve Generation : Arc Generation Using DDA Algorithm, Interpolation,
Types of Curves : Hilbert's Curve, Koch Curve, B-Spline, Bezier Curves

6.1 Introduction to Curve Generation

- Real world objects are not always made up of basic primitives like line, circle, ellipse, square etc.
- Usually, natural objects are very irregular in shape and size. It may not be possible to describe such shapes with standard mathematical formulas.
- Through the curve generation procedure, it is possible to model any random shape. However, the process of curve generation requires detail mathematical analysis of object shape. The process is very complex.
- In the following section, we will see a few basic and few advance curve generation methods.

6.1.1 Arc Generation using DDA Algorithm

Q. Explain the process of arch generation using DDA algorithm.

Q. How to generate arch DDA approach?

- In the second chapter, we have already discussed the DDA approach for line drawing. To draw the arc using DDA, we must know the differential equation for the curve.
- DDA is an incremental approach; it computes the coordinates of the next pixel by adding some amount to the current pixel.

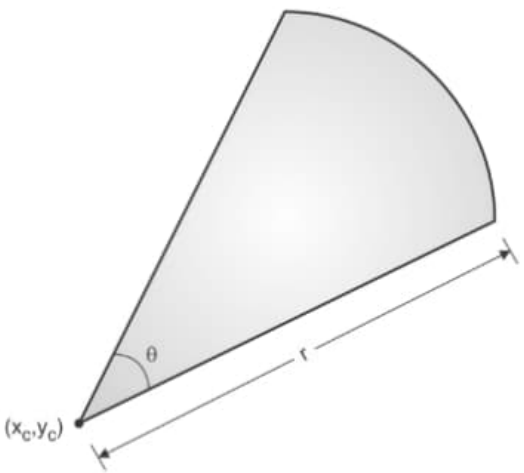


Fig. 6.1.1 : Arc generation

From the Fig. 6.1.1, the polar representation of any point (x, y) on arc boundary would be,

$$x = x_c + r \cos \theta \quad ...(6.1.1)$$

$$y = y_c + r \sin \theta \quad ...(6.1.2)$$

Here, (xc, yc) is the center of arc and r is the radius of the arc.

- By changing the θ, we get different points on the arc. Increment in θ determines how far the next point will be from the current point.
- Larger θ increment implies bigger steps and small θ increment implies smaller steps.
- We need to find out the change in (x, y) coordinate of the next point over the change in θ. This change can be measured by taking the derivation of Equation (6.1.1) and Equation (6.1.2) with respect to θ.

$$\frac{dx}{d\theta} = 0 + (-r \sin \theta)$$

$$\frac{dy}{d\theta} = 0 + (r \cos \theta)$$

$$\therefore dx = -r \sin \theta . d\theta \quad ... (6.1.3)$$

$$dy = r \cos \theta . d\theta \quad ... (6.1.4)$$

From Equation (6.1.1) and Equation (6.1.2),

$$r \cos \theta = x - x_c$$

$$r \sin \theta = y - y_c$$

By replacing these values in Equation (6.1.3) and Equation (6.1.4)

$$dx = -(y - yc). d\theta$$

$$dy = (x - xc). d\theta$$

It says when we increment θ by dθ, x and y will change by dx and dy, respectively. Thus, we can find the new point by adding dx and dy in the current point. So if a current point on the arc is (x_1, y_1), next point (x_2, y_2) can be computed incrementally as follows,

$$x_2 = x_1 + dx = x_1 - (y - y_c). d\theta$$

$$y_2 = y_1 = dy = y_1 + (x - x_c). d\theta$$

Q. Enlist advantages and disadvantages of DDA arc generation.

Advantages :

- Easy to understand.
- Easy to implement.
- This can be implemented in hardware so it would be fast.

Disadvantages :

- On scaling, curves behave differently, may not remain arc.
- Difficult to perform clipping.
- Need complex data structures of efficient implementation.

6.1.2 Interpolation

W-18

Q. Explain curve generation using Interpolation technique. **(W-18, 4 Marks)**

The curve is specified by a set of control points. Position of control points controls the shape of the curve. Fitting a single curve of the higher degree to all the control points is difficult. Normally, the big curve is approximated by joining piecewise polynomials of lower degree curves, which approximates few control points only.

If the curve passes through the control points, it is called **interpolation**. Interpolation curves are used in animation and specifying camera motion. It is also used in digitizing the coordinates.

If the curve does not pass through the control points and approximate the shape, it is called **approximation** or **extrapolation**. Such curves are used to estimate the shape of the object surface. Fig. 6.1.2 shows the geometry of both methods.

Manipulating spline curves using control point representation is very easy. The designer can adjust the shape of the curve by repositioning the control point. It is highly preferable in interactive graphics designing. Applying transformation on each point of the curve is equivalent to applying transformation first to control points and redrawing the curve.

(a) Interpolation (b) Approximation

Fig. 6.1.2 : Interpolation v/s Approximation

Complex, big curves for which the mathematical equations are not known, can be generated by piecewise approximation. The curve is sampled into a set of control points. The known small curve is generated for a subset of control points. All such small curves are then joined to get the final big complex curve.

Known curves for a small subset of control points are often derived using mathematical expressions. The curve can be made to pass from control points by adjusting the parameters of the mathematical expression. Explicitly, curves are represented as,

$$x = fx(u)$$

$$y = fy(u)$$

$$z = fz(u)$$

6.1.2.1 Lagrange Interpolation

Q. Write a short note on Lagrange interpolation method.

- Let us assume that the curve passes through the n control points (x_i, y_i, z_i), i = 1, 2, ..., n. Curve is represented by parametric form. Lagrange polynomials are used for polynomial interpolation.
- Coordinates of each point on the curve is given by the sum of each point weighted by the blending functions.

Blending functions :

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

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

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

- **Function B_i (u)** is called **blending function**. Blending function determines the effect of i^{th} control point for given 'u' on the rest of the curve points.
- We can force the curve to pass form given point i, by setting $B_i(u) = 1$. This value of u has complete control over the curve.

Following blending functions will be used to interpolate the four control points :

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

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

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

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

For $u = -1$, $B_1(u) = -1$, $B_2(u) = B_3(u) = B_4(u) = 0$

For $u = 0$, $B_2(u) = 1$, $B_1(u) = B_3(u) = B_4(u) = 0$

For $u = 1$, $B_3(u) = -1$, $B_1(u) = B_2(u) = B_4(u) = 0$

For $u = 2$, $B_4(u) = 1$, $B_1(u) = B_2(u) = B_3(u) = 0$

Individual coordinates of the points on the curve in space can be computed as,

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

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

A generalized formula for the bending function is given as,

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

- The curve generated using the above formula is called the **Lagrange Interpolation curve**.

Limitations :

Q. State the limitations of Lagrange interpolation method.

- Curve wiggles between the control points.
- Curve segments are not smoothly joined.
- The curve may not lie within the convex hull of the control points

6.2 Types of Curve

Q. Enlist the types of curve representation.

Curves are categorized as follow :

1. Spline curves representation

- Bezier curve
- B-Spline curve

2. Fractal representation

- Hilbert curve
- Koch curve

6.2.1 Spline Curve Representation

Q. What is spline ?

Q. Explain interpolation and approximation with suitable diagram.

In this section, we will discuss various spline construction methods.

The term spline came from the concept of drafting. Draftsman uses flexible strip which bends in the direction of the weight (Refer Fig. 6.2.1). Bending of the strip is proportional to the magnitude of applied force.

Such spline may be long and very smooth, mathematically we can describe such curves by piecewise cubic curves using cubic polynomial equations.

A duck (Weight)

Ducks trace out curve

Ducks trace out curve

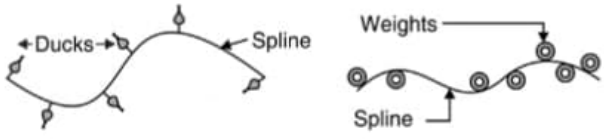


Fig. 6.2.1 : Spline generation through ducks

Type and shape of the spline curve depend on basis function (also called blending functions) and certain boundary conditions.

Basically, splines are continuous and smooth curves, used in digitizing drawing, specifying smooth motion path for animation, drafting engineering components etc.

Depending upon blending functions and boundary conditions, a curve generated from a given set of control points may or may not interpolate the control points.

If the curve passes through the control points, it is called **interpolation**. Interpolation curves are used in animation and specifying camera motion. It is also used in digitizing the coordinates.

If the curve does not pass through the control points and approximate the shape, it is called **approximation** or **extrapolation**. Such curves are used to estimate the shape of the object surface. Fig. 6.2.2 shows the geometry of both methods.

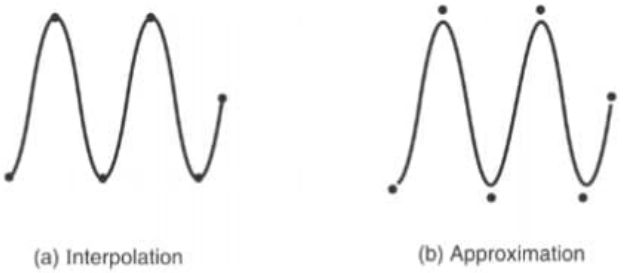


Fig. 6.2.2 : Interpolation v/s Approximation

6.2.1.1 Bezier Curve

Q. What is Bezier curve? Give the blending functions for Bezier curve.

Bezier curve was first invented by Pierre Bezier. Due to its nice properties, Bezier curves are highly used in engineering designs and CAD packages.

With cubic polynomials, Bezier curve can approximate any number of control points. In the Bezier curve, the degree of polynomial depends on a number of control points used to generate curve segment.

In Bezier curve, the degree of the polynomial is always one less than a number of control points. Fig. 6.2.3 shows Bezier curves with three, four and five control points having a degree of polynomial two, three and four, respectively.

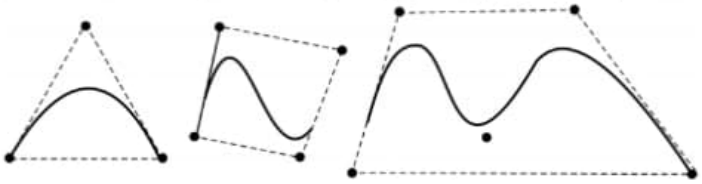


Fig. 6.2.3 : Bezier curves of degree 2, 3 and 4 with their convex hull

With n + 1 control points, parametric equation of Bezier curve approximating points p_0 to p_n,

$$P(t) = \sum_{i=0}^{n} p_i \, BEZ_{i,n}(t), \; 0 \le t \le 1$$

Blending functions of Bezier curves are directly derived from **Bernstein polynomials**.

$$BEZ_{i,n}(t) = C(n, i) t^i (1-t)^{n-i}$$

C(n, i) is the binomial coefficient :

$$C(n, i) = \frac{n!}{(n-i)!\, i!}$$

Bezier curve equation in each direction is written as,

$$x(t) = \sum_{i=0}^{n} x_i \, BEZ_{i,n}(t), \; 0 \le t \le 1$$

$$y(t) = \sum_{i=0}^{n} y_i \, BEZ_{i,n}(t), \; 0 \le t \le 1$$

$$z(t) = \sum_{i=0}^{n} z_i \, BEZ_{i,n}(t) \; 0 \le t \le 1$$

Properties of Bezier Curves :

Q. Enlist the properties of Bezier curve.

Degree of curve is one less than number of control points :

- Always interpolates first and last control points and approximates remaining two.
- The slope of the derivative at the beginning is along the line joining the first two points and slope of the derivative at the end is along the line joining last two points. Bezier curve always satisfies the convex hull property.
- At any parameter value t, sum all four Bezier blending function is always 1, i.e.,

$$\sum_{i=0}^{n} BEZ_{i,n}(t) = 1$$

- Polynomial smoothly follows the control points without much oscillation.
- Bezier curves do not have local control; repositioning one control point changes the entire curve.
- Curve is invariant under affine transformation.
- Basis functions are real.
- The curve exhibits variation diminishing property, i.e. any line intersects the Bezier curve at most as often as that line intersects the polygon interpolating control points.
- Bezier curve can fit any number of control points.
- Reversing the order of control points yield the same Bezier curve.

Applications of Bezier Curves :

Q. State few applications of Bezier curve.

Bezier curve is used in many design applications. Bezier curves are used to model shapes and surfaces of automobiles, engineering, architectures, and aeronautics objects. Few of the applications are listed where the Bezier curve is being used to model the shapes.

- Computer graphics
- Computer animation
- Engineering design
- CAD packages
- Modelling
- Graphics packages like photoshop
- Designing automobile parts

Cubic Bezier Curves :

Q. Write a short note on Cubic Bezier curve.

- Cubic Bezier curve is important to discuss. Cubic Bezier curves are the bridge between computation cost and smoothness.

- They are very much flexible and estimate shape nicely. As per the property of Bezier curve, we should specify four control points to generate a cubic Bezier curve.
- Blending functions for cubic Bezier curve is derived by putting n = 3 in equation of Bezier curve,

$$BEZ_{0,3}(t) = (1-t)^3$$

$$BEZ_{1,3}(t) = 3t(1-t)^2$$

$$BEZ_{2,3}(t) = 3t^2(1-t)$$

$$BEZ_{3,3}(t) = t^3$$

- Fig. 6.2.4 shows the plot of blending functions over parameter range $0 \le t \le 1$. For t = 0, only first blending function is non zero and it has value 1. And at t = 1, only last blending function is non zero and its value is 1.

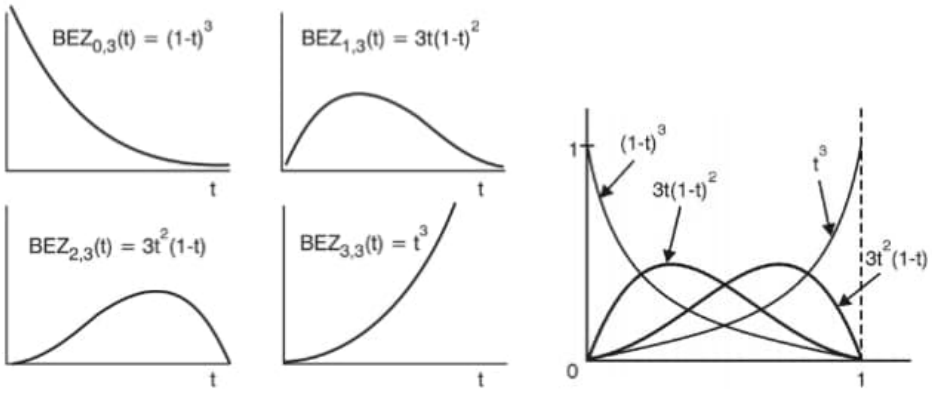


Fig. 6.2.4 : Bezier blending functions

- Thus Bezier curve always interpolates endpoints. For the rest of the values of t, all blending functions contribute to the shape of the curve.
- Blending functions $BEZ_{1,3}$ and $BEZ_{2,3}$ are maximum at t = 1/3 and t = 2/3 respectively. At any value of t, the summation of Bezier blending function is always 1.

Matrix Representation :

Q. Give the matrix representation of Bezier curve.

We can derive a matrix representation of a cubic Bezier curve using blending function representation as below :

$$P(t) = \sum_{i=0}^{n} p_i BEZ_{i,n}(t), \quad 0 \le t \le 1$$

$$= p_0 BEZ_{0,3}(t) + p_1 BEZ_{1,3}(t) + p_2 BEZ_{2,3}(t) + p_3 BEZ_{3,3}(t)$$

$$= p_0(1-t)^3 + p_1 3t(1-t)^2 + p_2 3t^2(1-t) + p_3 t^3$$

$$= (1 - 3t + 3t^2 - t^3)p_0 + (3t - 6t^2 + 3t^3)p_1 + (3t^2 - 3t^3)p_2 + t^3 p_3$$

$$P(t) = [t^3\ t^2\ t\ 1] \cdot \begin{bmatrix} -1 & 3 & -3 & -1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$P(t) = T \cdot M_{Bez} \cdot G_{Bez}$$

Where, M_{Bez} and G_{Bez} represent Bezier basis matrix and Bezier geometric matrix respectively.

Limitations of the Bezier Curve :

Q. What are the limitations of Bezier curve?

- Bezier curve approximates the control points so it cannot be used in the applications which require interpolating curve.
- Bezier curve does not have local control i.e. changing the position of one control point influences the entire curve.
- Degree of the curve depends on a number of the control point i.e. adding more control points increases the smoothness of the curve at the cost of processing higher degree polynomials.

Subdivision Method :

Q. How to generate Bezier curve using subdivision method.

Another method to generate a Bezier curve is a subdivision method. This method generates a Bezier curve by simply midpoint calculation. Let us derive the Bezier curve defined by control points P_1, P_2, P_3 and P_4. Join consecutive control points using straight lines. Suppose midpoint of lines joining points P_1P_2, P_2P_3 and P_3P_4 are L_2, H and R_3 respectively. Naming conventions are chosen to simplify the understanding of computation.

- Join the point L_2 and H and name its midpoint L_3. Similarly, join points H and R_3 and name its midpoint R_2. Join points L_3 and R_2 and assign label L_4 to its midpoint ($L_4 = R_1$).
- Recursively draw two Bezier curve segments between control points (L_1, L_2, L_3, L_4) and (R_1, R_2, R_3, R_4) respectively. Both these curve segments are independent Bezier curves.
- Numbers of recursive calls decide the smoothness of curve. We can derive the matrix representation for this subdivision as follows :

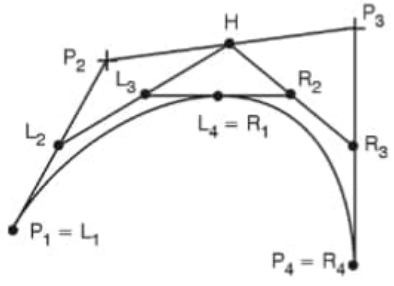


Fig. 6.2.5 : Bezier curve generation using subdivision method

From Fig. 6.2.5,

$$L_1 = P_1 = \frac{8P_1}{8}$$

$$L_2 = \frac{P_1 + P_2}{2} = \frac{4P_1 + 4P_2}{8}$$

$$H = \frac{P_2 + P_3}{2}$$

$$L_3 = \frac{L_2 + H}{2} = \frac{\frac{P_1 + P_2}{2} + \frac{P_2 + P_3}{2}}{2}$$

$$L_3 = \frac{P_1 + 2P_2 + P_3}{4} = \frac{2P_1 + 4P_2 + 2P_3}{8}$$

$$R_4 = P_4 = \frac{8P_4}{8}$$

$$R_3 = \frac{P_3 + P_4}{2} = \frac{4P_3 + 4P_4}{8}$$

$$R_2 = \frac{H + R_3}{2} = \frac{\frac{P_2 + P_3}{2} + \frac{P_3 + P_4}{2}}{2}$$

$$= \frac{P_2 + 2P_3 + P_4}{4} = \frac{2P_2 + 4P_3 + 2P_4}{4}$$

$$L_4 = R_1 = \frac{L_3 + R_2}{2}$$

$$= \frac{\frac{P_1 + 2P_2 + P_3}{4} + \frac{P_2 + 2P_3 + P_4}{4}}{2} = \frac{P_1 + 3P_2 + 3P_3 + P_4}{8}$$

Bezier left subdivision matrix is given as,

$$G_B^L = D_B^L \cdot G_B = \frac{1}{8}\begin{bmatrix} 8 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Wher, eD_B^L is Bezier left subdivision matrix

$$G_B^R = D_B^R \cdot G_B = \begin{bmatrix} 1 & 3 & 3 & 1 \\ 0 & 2 & 4 & 2 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Where, D_B^R is Bezier right subdivision matrix.

Example 6.2.1 : Obtain the curve parameters for drawing a smooth Bezier curve for the following points A (0,10), B (10,50), C (70, 40) and D (70, – 20). **(W-18, 6 Marks)**

Solution :

The curve passes through the endpoints (0, 0) and (2, 1)

Matrix representation of the Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & -1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} 0 & 10 \\ 10 & 50 \\ 70 & 40 \\ 70 & -20 \end{bmatrix}$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -110 & 0 \\ 150 & -150 \\ 30 & 120 \\ 0 & 10 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [-110t^3 + 150t^2 + 30t \quad -150t^2 + 120t + 10]$$

Let us find point on curve for parameter for step value 0.2. Curve interpolates first and last control points, so for t = 0 we shall get first control point and for t = 1, we shall get the fourth control point.

$$t = 0.0 \Rightarrow Q(0.0)$$
$$= [-110(0.0)^3 + 150(0.0)^2 + 30(0.0) \quad -150(0.0)^2 + 120(0.0) + 10]$$
$$= [0 + 0 + 0 \quad 0 + 0 + 10]$$
$$= [0 \quad 10] = \text{First control point}$$

$$t = 0.2 \Rightarrow Q(0.2)$$
$$= [-110(0.2)^3 + 150(0.2)^2 + 30(0.2) \quad -150(0.2)^2 + 120(0.2) + 10]$$
$$= [-0.88 + 6 + 6 \quad -6 + 24 + 10]$$
$$= [11.12 \quad 28]$$

$$t = 0.4 \Rightarrow Q(0.4)$$
$$= [-110(0.4)^3 + 150(0.4)^2 + 30(0.4) \quad -150(0.4)^2 + 120(0.4) + 10]$$
$$= [-7.04 + 24 + 12 \quad -24 + 48 + 10]$$
$$= [28.96 \quad 34]$$

$$t = 0.6 \Rightarrow Q(0.6)$$
$$= [-110(0.6)^3 + 150(0.6)^2 + 30(0.6) \quad -150(0.6)^2 + 120(0.6) + 10]$$
$$= [-23.76 + 54 + 18 \quad -54 + 72 + 10]$$
$$= [48.24 \quad 28]$$

$$t = 0.8 \Rightarrow Q(0.8)$$
$$= [-110(0.8)^3 + 150(0.8)^2 + 30(0.8) \quad -150(0.8)^2 + 120(0.8) + 10]$$
$$= [-56.32 + 96 + 24 \quad -96 + 96 + 10]$$
$$= [63.48 \quad 10]$$

$$t = 1.0 \Rightarrow Q(1.0)$$
$$= [-110(1.0)^3 + 150(1.0)^2 + 30(1.0) \quad -150(1.0)^2 + 120(1.0) + 10]$$
$$= [-110 + 150 + 30 \quad -150 + 120 + 10]$$
$$= [70 \quad -20] = \text{Last control point}$$

Value of t	Point on curve
t = 0.0	[0 10]
t = 0.2	[11.12 28]
t = 0.4	[28.14 34]
t = 0.6	[48.24 28]
t = 0.8	[63.48 10]
t = 1.0	[70 – 20]

Example 6.2.2 : Construct the Bezier curve of order three with control points $P_1(0, 0)$, $P_2(1, 3)$, $P_3(4, 2)$ and $P_4(2, 1)$. Generate at least five points on the curve.

Solution :

The curve passes through the endpoints (0, 0) and (2, 1)

Matrix representation of the Bezier curve is

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & -1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 3 \\ 4 & 2 \\ 2 & 1 \end{bmatrix}$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -7 & 4 \\ 6 & -12 \\ 3 & 9 \\ 0 & 0 \end{bmatrix}$$

$\therefore$ Equation of Bezier curve would be,

$$Q(t) = [-7t^3 + 6t^2 + 3t \qquad 4t^3 - 12t^2 + 9t]$$

We have to find at least five points on the curve. Let us find a point on the curve for parameter step value 0.2.

$t = 0.0 \Rightarrow Q(0.0)$

$= [-7(0.0)^3 + 6(0.0)^2 + 3(0.0) \qquad 4(0.0)^3 - 12(0.0)^2 + 9(0.0)]$

$= [0 + 0 + 0 \qquad 0 - 0 + 0]$

$= [0 \quad 0]$

$t = 0.2 \Rightarrow Q(0.2)$

$= [-7(0.2)^3 + 6(0.2)^2 + 3(0.2) \qquad 4(0.2)^3 - 12(0.2)^2 + 9(0.2)]$

$= [-0.056 + 0.24 + 0.6 \qquad 0.032 - 0.48 + 1.8]$

$= [0.784 \quad 1.352]$

$t = 0.4 \Rightarrow Q(0.4)$

$= [-7(0.4)^3 + 6(0.4)^2 + 3(0.4) \qquad 4(0.4)^3 - 12(0.4)^2 + 9(0.4)]$

$= [-0.448 + 0.96 + 1.2 \qquad 0.256 - 1.92 + 3.6]$

$= [1.712 \quad 1.424]$

$t = 0.6 \Rightarrow Q(0.6)$

$= [-7(0.6)^3 + 6(0.6)^2 + 3(0.6) \qquad 4(0.6)^3 - 12(0.6)^2 + 9(0.6)]$

$= [1.512 + 2.16 + 1.8 \qquad 0.864 - 4.32 + 5.4]$

$= [5.472 \quad 1.944]$

$t = 0.8 \Rightarrow Q(0.8)$

$= [-7(0.8)^3 + 6(0.8)^2 + 3(0.8) \qquad 4(0.8)^3 - 12(0.8)^2 + 9(0.8)]$

$$= [-3.584 + 3.84 + 2.4 \qquad 2.048 - 7.68 + 7.2]$$

$$= [2.656 \quad 1.568]$$

$$t = 1.0 \Rightarrow Q(1.0)$$

$$= [-7(1.0)^3 + 6(1.0)^2 + 3(1.0) \qquad 4(1.0)^3 - 12(1.0)^2 + 9(1.0)]$$

$$= [-7 + 6 + 3 \qquad 4 - 12 + 9]$$

$$= [2 \quad 1]$$

Value of t	Point on curve
t = 0.0	[0 0]
t = 0.2	[0.784 1.352]
t = 0.4	[1.712 1.424]
t = 0.6	[5.472 1.944]
t = 0.8	[2.656 1.568]
t = 1.0	[2 1]

Example 6.2.3 : Find the equation for the Bezier curve, which passes through control points (0, 0) and (– 3, 2) and controlled by (6, 4) and (3, 1). Also find points on curve for t = 0, 0.4, 0.8, 1.

Solution :

Assume that curve is defined by control points p_0, p_1, p_2 and p_3. Curve passes through the end points (0, 0) and (– 3, 2)

So, p_0 = (0, 0) and p_3 (– 3, 2). Let us consider p_1 = (6, 4) and p_2 = (3, 1)

Matrix representation of Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & -1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 6 & 4 \\ 3 & 1 \\ -3 & 2 \end{bmatrix}$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 6 & 11 \\ -27 & -21 \\ 18 & 12 \\ 0 & 0 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [6t^3 - 27t^2 + 18t \qquad 11t^3 - 21t^2 + 12t]$$

Let us find point on curve for given parameter value,

$$t = 0 \Rightarrow Q(0) = [6(0) - 27(0) + 18(0) \quad 11(0) - 21(0) + 12(0)]$$

$$= [0 \quad 0]$$

$$t = 0.4 \Rightarrow Q(0.4)$$
$$= [6(0.4)^3 - 27(0.4)^2 + 18(0.4) \quad 11(0.4)^3 - 21(0.4)^2 + 12(0.4)]$$
$$= [0.384 - 4.32 + 7.2 \quad 0.704 - 3.36 + 4.8]$$
$$= [3.264 \quad 2.144]$$
$$t = 0.8 \Rightarrow Q(0.8)$$
$$= [6(0.8)^3 - 27(0.8)^2 + 18(0.8) \quad 11(0.8)^3 - 21(0.8)^2 + 12(0.8)]$$
$$= [5.632 - 17.28 + 14.4 \quad 5.632 - 13.44 + 9.6]$$
$$= [2.752 \quad 1.792]$$
$$t = 1 \Rightarrow Q(1) = [6(1)^3 - 27(1)^2 + 18(1) \quad 11(1)^3 - 21(1)^2 + 12(1)]$$
$$= [6 - 27 + 18 \quad 11 - 21 + 12]$$
$$= [-3 \quad 2]$$

Value of t	Point on curve
t = 0.0	[0 0]
t = 0.4	[3.264 2.144]
t = 0.8	[2.752 1.792]
t = 1.0	[– 3 2]

6.2.1.2 B-Spline Curve

Q. Write a note on B-spline curve.

- In the Bezier curve, the degree of the polynomial is always one less than a number of control points. So we cannot have more than four points for cubic Bezier curves.
- Furthermore, change in the single control point has a global effect on the Bezier curve. B Spline curves are the most widely used class of curves for approximating the shape due to its following properties :
  - Degree of a polynomial is independent of a number of control points.
  - They have local control over the curve and surfaces.
  - However, derivation and generation of B spline are more complex than Bezier curves.
  - The b-spline curve with (n + 1) control points is expressed in form of blending function as,

$$P(t) = \sum_{i=0}^{n} p_k.B_{i,d}(t), \; t_{min} \le t \le t_{max} \text{ and } 2 \le d \le n + 1$$

- For Bezier curve, the range of t was always between 0 and 1. In B spline curve, t depends on types of B-spline.
- B-Spline bending functions $B_{i,d}$ are polynomials with degree d – 1, d can take any integer value between 2 and number of control points.
- We can achieve local control by defining the blending function on subinterval of the total range of parameter t.
- Blending functions of Bezier curve are described by Bernstein polynomials, whereas blending functions for B-spline curve are defined using Cox-deBoor recursive formula.

$$B_{i,1}(t) = \begin{cases} 1, & \text{if } t_i \le t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \frac{t - t_i}{t_{i+d-1} - t_i} B_{i,d-1}(t) + \frac{t_{i+d} - t}{t_{i+d} - t_{i+1}} B_{i+1,d-1}(t)$$

- Blending functions are defined over d subintervals of the total range of t. So change in one control point can affect maximum d piecewise small curves, it does have a global effect on the curve.
- The selected set of subinterval t_i is called know vector. We can choose any value for t_i in monotonically increasing order, i.e. $t_i \le t_{i+1}$. However, the value of t_{min} and t_{max} is controlled by a number of control points, parameter d and how we set up knot vector.

Properties of B-Spline Curve :

Q. What are the properties of B-Spline curve?

- Degree of polynomial does not depend on number of control points.
- B Spline curves do not interpolate any of the control points.
- The generated polynomial has degree d – 1 with inherent C^{d-2} parametric continuity.
- The curve has n + 1 blending function if it is described with n + 1 control points
- Size of knot vector is n + d + 1.
- Entire range of parameter t is divided in (n + d) subintervals.
- If the knot vector is defined as $\{t_0, t_1, t_2, \ldots, t_{n+d}\}$, B-spline curve spans only in the interval t_{d-1} to t_{n-1}.
- Each blending function $B_{i,d}$ spans over d subinterval of the entire range of t, starting from knot value t_i.
- B-Spline allows local control.
- C^2 continuity is inherent in B Spline.
- Each B-spline curve segment is affected by d control points.
- Change in one control point influence shape of at most d curve sections.
- B-Spline curve satisfies the property of convex hull, so they are tightly bound within the control point convex hull boundary.
- For any value of t in the range t_{d-1} to t_{n+1}, the sum of all blending function is 1, i.e.,

$$\sum_{i=0}^{n} B_{i,d}(t) = 1$$

According to the specification of knot vector, B-spline curves are classified into three categories :

- Uniform B-Splines
- Open uniform B-splines
- Non-uniform B-Splines

1. Uniform Periodic B-Spline :

Q. Write a short note on uniform periodic B-spline curve.

In uniform B-Splines, knot values in knot vector are equi-spaced, i.e. the difference between two consecutive knot values is constant. For example {– 3, – 2, – 1, 0, 1, 2, 3, 4} is uniform knot vector. Knot values are often normalized such that they fall in range 0 to 1, for example {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6}.

- A more convenient way is to represent the knot vector with integer knot values, starting from zero, i.e. {0, 1, 2, 3, 4, 5, 6, 7}.
- Blending functions of uniform B-spline are periodic and successive blending function is translated/shifted version of the previous one.

$$B_{i,d}(t) = B_{i+1,d}(t+\Delta t) = B_{i+2d},(t+2\Delta t) = \ldots\ldots$$

- Where, Δt is the difference between two successive knot values. Blending functions for uniform B-spline are shown in Fig. 6.2.6.

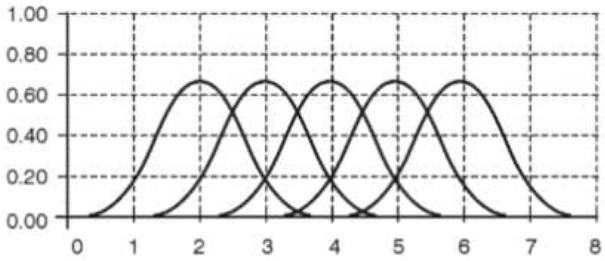


Fig. 6.2.6 : Blending functions for uniform B Spline

2. Open Uniform B-Spline :

Q. Write a short note on open uniform B-spline curve.

- Open uniform B spline is the bridge between uniform and non-uniform splines. Sometimes it is considered as a special case of uniform B spline; sometimes it is considered as a special case of non-uniform B-Spline.
- It differs from uniform spline in the specification of knot vector. Like uniform splines, knot vector in open uniform also has uniform spacing between successive knot values except first and last. End knot values are repeated d times.
- For d = 2 and n = 4, size of knot vector is n + 1 + d = 7. First and last knot value is repeated d = 2 times. So, starting with 0, knot vector is specified as, {0, 0, 1, 2, 3, 4, 4}
- For d = 4 and n = 5, size of knot vector is n + 1 + d = 10. First and last knot value is repeated d = 4 times. So, starting with 0, knot vector is specified as,

  {0, 0, 0, 0, 1, 2, 3, 3, 3, 3}

  We can normalize the range of knot value between 0 and 1.

  For d = 2 and n = 4, {0, 0, 0.25, 0.5, 0.75, 1, 1}

  For d = 4 and n = 5, {0, 0, 0, 0, 0.33, 0.66, 1, 1, 1, 1}
- We can generate integer valued knot vector for given n and d using following formula :

$$t_i = \begin{cases} 0, & \text{for } 0 \le i < d \\ i - d + 2, & \text{for } d \le i \le n \\ n - d + 2, & \text{for } i > n \end{cases}$$

- First knot has value 0 and last has value n – d + 2.
- Characteristics of the open B spline curve are very much similar to the Bezier curve. In fact, open B spline reduces to Bezier curve when the degree of the polynomial is one less than the control point, i.e. d = n + 1. For such cubic B spline, d = 4, n + d = 4, n + 1 = 4, so size of knot vector is 8. In this case, knot vector would be {0, 0, 0, 0, 1, 1, 1, 1}.

– Blending functions for open uniform B-spline with n = 4 and d = 3 are shown in Fig. 6.2.7.

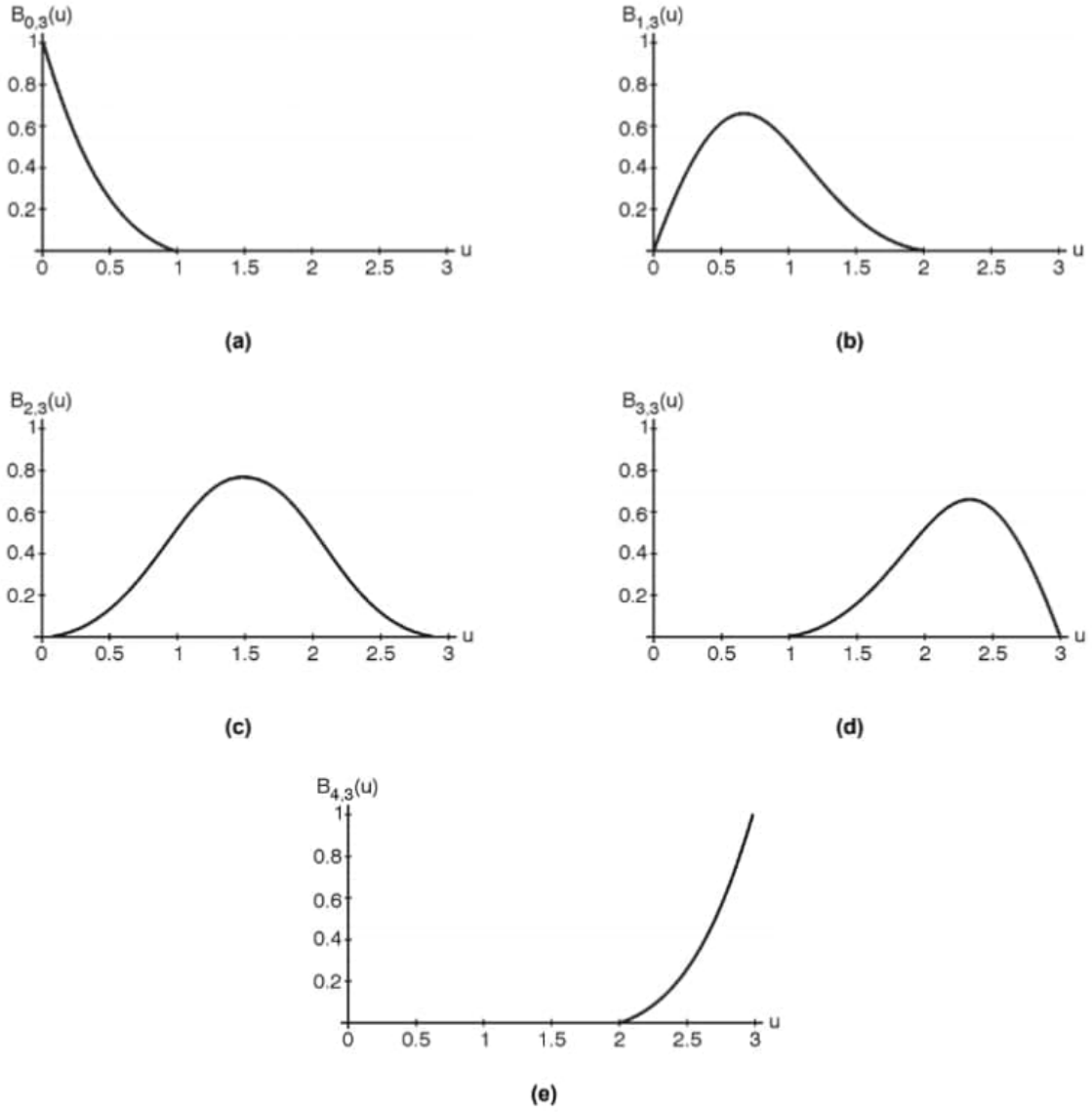


Fig. 6.2.7 : Blending functions for open uniform curve

3. Non-uniform B-Spline :

Q. Write a short note on non-uniform B-spline curve.

– Knot vector for this class of polynomial is least restrictive. Knot vector can repeat the knot values; spacing between successive knot values can be different. The only constraint is that knot value must be in increasing order.

Examples are,

{0, 1, 2, 3, 4, 5}

{0, 1, 1, 2, 3, 4, 4, 5}

{0, 0.1, 0.4, 0.4, 0.7, 0.9, 1, 1}

- Such curves are most flexible in controlling the shape. Due to the unequal spacing of knot value, we get different shapes blending functions at different intervals. By repeating the knot values, we can disturb the shape of the curve and even we can introduce a discontinuity in the curve. For each repeat, discontinuity of curve reduces by one. Derivation of blending functions of non-uniform B spline is the same as uniform and open uniform curves.
- With the help of n + 1 control points, we can use Cox-deBoor recursive formula to find blending function with specified degree and knot vector. Matrix representation can be used to avoid recursive computation of Cox-deBoor equation.
- Blending function for non-uniform B-Spline is shown in the Fig. 6.2.8.

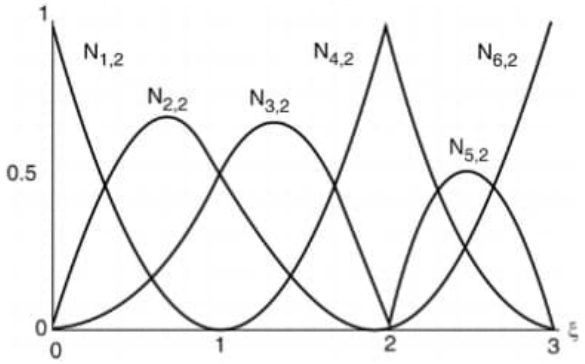


Fig. 6.2.8 : Blending functions for non-uniform curve

6.2.1.3 Bezier vs. B-Spline

Q. Differentiate the Bezier curve vs. B-Spline curve.

Sr. No.	Bezier Curve Generation	B-Spline Curve Generation
1.	Passes from the first and last control point.	Does not interpolate any control point.
2.	Does not have local control on a curve.	It has local control.
3.	Changing one control point affects the position of all points on the curve.	Changing one control point affects only a few points on the curve.
4.	Degree of Bezier curve is determined by the number of control points.	Degree of the B-Spline curve is independent of the number of control points.
5.	Need special treatment to join two Bezier curves with c_2 continuity.	Continuity is inherent in the B-Spline curve.
6.	Less flexible.	More flexible.
7.	During piecewise approximation, adds 3 control point to generate new curve segment.	During piecewise approximation, adds only 1 control point to generate new curve segment.

6.2.2 Fractal Representation

Q. What do you mean by fractals?

- Shapes like line, circle, ellipse, curves etc. are defined using Euclidean geometry or mathematical equation.

- Natural objects like plants, tree, mountain, waves etc. cannot be described by such mathematical notion.
- Natural shapes have many irregularities and self-similarities.
- Shapes generated using Euclidean geometry have a smooth surface. Euclidean geometry methods are not suitable for generating a realistic display of the natural scene.
- Natural objects are well described using **fractal geometry methods** - which are actually a procedure rather than an equation.

Fractal objects possess the following two properties :

- Infinite detail at every point.
- Self-similarity between object parts.
- Natural objects have infinite details; however, we should design a process which produces finite detail because on the computer we cannot display infinite detail.
- After certain levels of zoom in, Euclidean shapes lead to smooth drawing. While in the fractal object, if we keep zoom in, we continue to see the same detail as it appears in the original object. Fig. 6.2.9 describes the discussed concept.

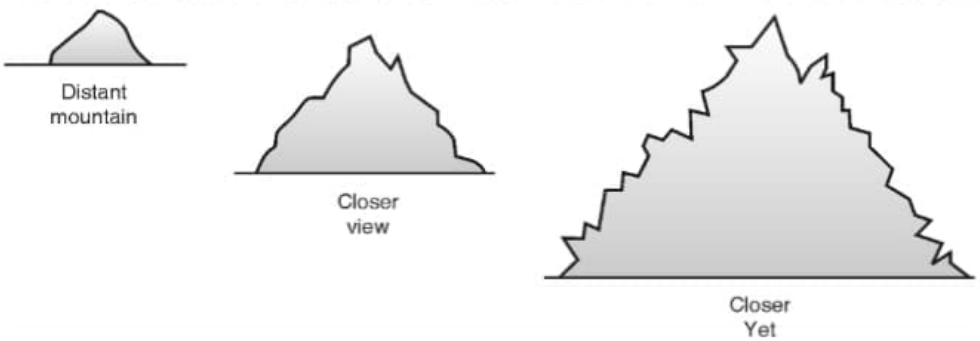


Fig. 6.2.9 : View of the mountain at different zoom level

- Zoom in effect on the monitor is achieved by selecting smaller and smaller window with the sameviewport. Less and less detail is mapped to the same size of the viewport.
- Amount of variation in object detail is described by a number called **fractal dimension**. Fractal dimension can be any real number. Fractal dimension is sometimes referred to as the **fractional dimension**.

Fractal Dimensions :

Q. Write a short note on fractal dimensions.

- Detail variation in a fractal is controlled by the fractal dimension number D. We can consider D as a measure of the roughness of the object.
- Objects with jaggy boundary have larger D value. We can write iterative procedure controlling detail by specifying D. However, it is difficult to estimate the value of D for real objects.
- Self-similar fractals are obtained by recursively applying the same scaling factor to Euclidean shapes.
- Fig. 6.2.10 shows the relationship between scaling factor and a number of parts for unit length line, unit square and unit cube with s = ½. With s = ½, the line is divided into two equal segments, the square is divided into four equal parts of equal area and cube is divided into eight subcubes of equal volume.

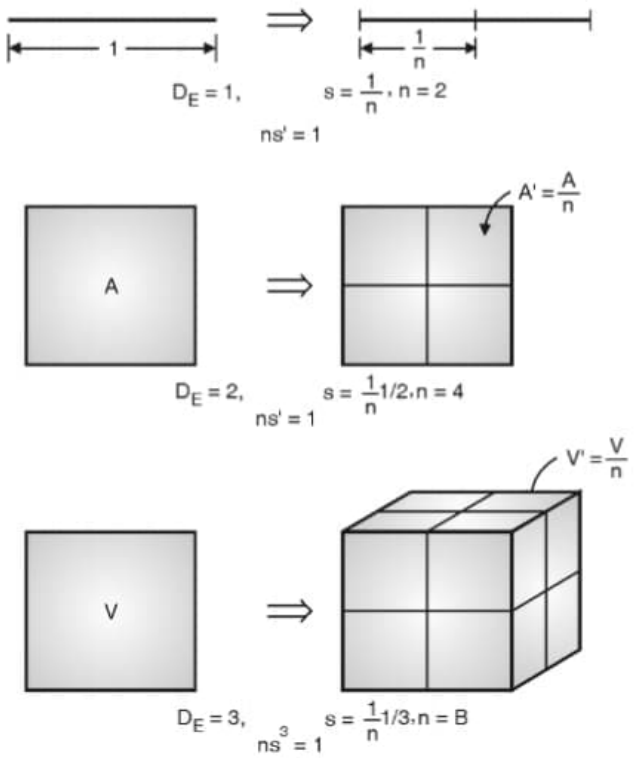


Fig. 6.2.10 : Subdivision of object with scaling factor s = ½ and Euclidean dimensions (D_E = 1, (a). D_E = 2 and (a). D_E = 3)

For these objects, the relationship between a number of subparts and the scaling factor is $n\ s^{DE} = 1$. For self-similar objects,

$$n \cdot s^D = 1$$

Solving the equation,

$$D = \frac{\ln n}{\ln (1/s)}$$

- D is called the fractal similarity dimension. For a self-similar object with different scaling factor for a different part, self-similarity dimension is obtained as,

$$\sum_{i=1}^{n} S_i^D = 1$$

Where, s_i is the scaling factor for i^{th} sub part of the object.

- For complex shapes like curves and surfaces, it is difficult to estimate the characteristics of subparts of the shape. The large shape is approximated by subdivides the shape into smaller simpler shapes, that is called **topological covering**. For example, smooth curves can be approximated using a very small line segment. The surface can be approximated using small polygons like triangles.
- Covering methods are used to determine object properties like length, area, the volume of an object by summing these properties smaller objects – forming the parent object. Covering methods also help in determining fractal dimensions (D).

6.2.2.1 Hilbert's Curve

Q. Write a short note on Hilbert's curve.

It was introduced by the German mathematician David Hilbert. This curve is also known as the **Hilbert space-filling curve**. The basic entity of this curve is U-shape as shown in Fig. 6.2.11(a).

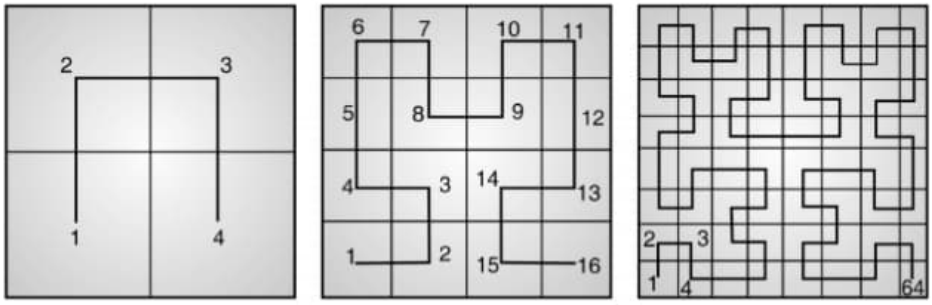


Fig. 6.2.11 : (a) First approximation, (b) Second approximation and (c) Third approximation

- Generation of Hilbert curve is quite simple. It visits the center of every square grid having size 2 × 2, 4 × 4, 8 × 8, 16 × 16, or any other size of $2^n \times 2^n$.
- The curve generation starts with a square. In a first approximation, the square is divided into 4 quadrants and the curve joins the centre of all quadrants by a straight line.
- In the second approximation, each quadrantis further subdivided into 2 × 2 grids, ending in 16 squares. Curve visits the centre of each small square in each quadrant before moving to the next quadrant (Refer Fig. 6.2.11(b))
- In third approximation, each of the 16 squares is further subdivided in 2 × 2 grids, ending in 64 squares. Curve visits the center of each small square before moving to higher level quadrant (Refer Fig. 6.2.11(c))
- On applying this process continuously,
  - The curve never crosses itself
  - Curve gets closer to a square containing it
  - With each subdivision, the length of the curve increases four times.
- There is no limit on length and hence there is no limit on curve length.
- At each subdivision, the scale of the square is reduced by factor 2, and length changed by factor 4.
- Ultimately, the Hilbert curve is a line with topological dimension $D_t = 1$, but as the curve gets scaled and twisted at every next approximation, it fills the square.
- The fractal dimension of the curve is given as,

$$N = 2^D$$

$$4 = 2^D$$

$$\text{So, } D = 2.$$

Thus, this curve has topological dimension 1 and fraction dimension 2.

6.2.2.2 Koch Curve

W-18

Q. Explain the Koch curve with diagram. **(W-18, 4 Marks)**

- Koch curve begins with the straight line. It divides the line into three segments. The middle segment is replaced by two sides of an equilateral triangle with the apex pointing outside. Now we have four segments. This same process is repeated on these four segments. General procedure for generating the Koch curve can be stated as :
  1. Divide a straight line into three equal parts.
  2. Draw an equilateral triangle on the middle segment, pointing in an outward direction with a base of the middle segment.
  3. Remove the line middle line segment.
- This procedure is repeated again and again to achieve more and more smoothness on the curve.
- To construct, deterministic, self-similar fractals, we start with geometric shapes called initiator. A subpart of the initiator is then repeatedly replaced by generator pattern.

Fig. 6.2.12 : Initiator and generator for Koch curve

Fig. 6.2.12 shows the initiator and generator pattern for the Koch curve.

- Each line segment of initiator is replaced by four segments of the generator. Gradually, the length of initiator tends to infinity, which more and more details are added to curve.
- After each iteration, the curve length is increased by one third. The number of line segments increases by a factor of four after each iteration. Length of each newly created line segment is $1/3^{rd}$ of the original line segment. The scaling factor is 1/3, so the fractal dimension is,

$$D = \frac{\ln(n)}{\ln(1/s)} = \frac{\ln(4)}{\ln(1/(1/3))} = \frac{\ln(4)}{\ln(3)} = 1.2619$$

- The total length of initiator increases by factor 4/3 in each iteration. Thus, the length of the curve after n iteration would be $(4/3)^n$. Fig. 6.2.13 illustrate the length calculation of the Koch curve.

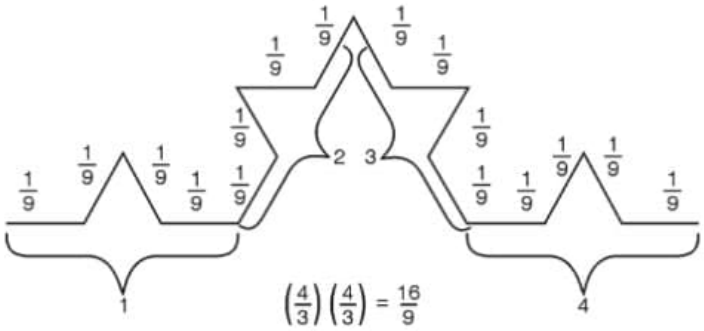


Fig. 6.2.13 : Length calculation after two iterations.

Fractals are non ending patterns. Each iteration adds finer detail. Fig. 6.2.14 shows the procedure of the Koch curve generation.

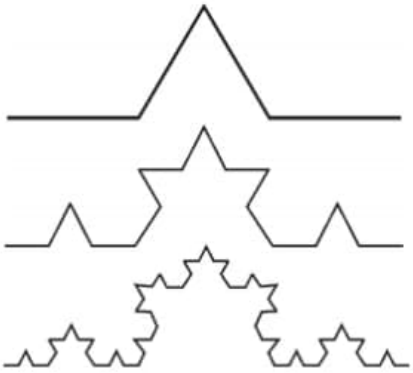


Fig. 6.2.14 : Generation of the Koch curve

6.3 Lab Programs

Program 6.3.1 : Write a program to draw following type of curve : Hilbert's curve. **(Lab 15)**

Solution :

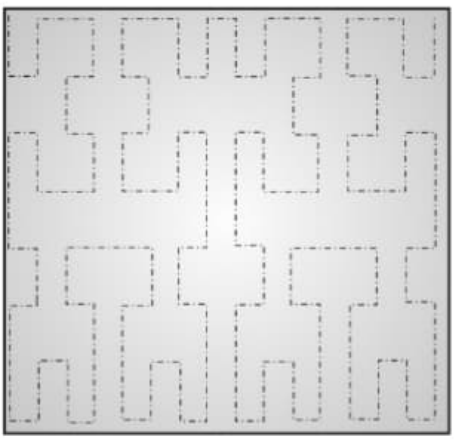
```
#include <stdio.h>
#include <conio.h>
#define N 16
#define K 3
#define MAX N * K

typedef struct
{
   int x;
   int y;
} point;
void Turn(int, point *, int, int );
void Divide(int, int, point *);
void main()
{
   clrscr();
   int d, x, y, cx, cy, px, py;
   char pts[MAX][MAX];
   point curr, prev;
   for (x = 0; x < MAX; ++x)
   {
        for (y = 0; y < MAX; ++y)
        {
             pts[x][y] = ' ';
        }
```

```
        prev.x = prev.y = 0;
        pts[0][0] = '.';
    }

    for (d = 1; d < N * N; ++d)
    {
        Divide(N, d, &curr);
        cx = curr.x * K;
        cy = curr.y * K;
        px = prev.x * K;
        py = prev.y * K;
        pts[cx][cy] = '.';

        if (cx == px )
        {
            if (py< cy)
            {
                for (y = py + 1; y < cy; ++y)
                {
                    pts[cx][y] = '|';
                }
            }
            else
            {
                for (y = cy + 1; y <py; ++y)
                {
                    pts[cx][y] = '|';
                }
            }
        }
        else
        {
            if (px< cx)
            {
                for (x = px + 1; x < cx; ++x)
                {
                    pts[x][cy] = '_';
                }
            }
```

```
            else
            {
                for (x = cx + 1; x <px; ++x)
                {
                    pts[x][cy] = '_';
                }
            }
        }
        prev = curr;
    }
    for (x = 0; x < MAX; ++x)
    {
        for (y = 0; y < MAX; ++y)
        {
            printf("%c", pts[y][x]);
        }
        printf("\n");
    }
    getch();
}

void Turn(int n, point *p, int rx, int ry)
{
    int t;
    if (!ry)
    {
        if (rx == 1)
        {
            p->x = n - 1 - p->x;
            p->y = n - 1 - p->y;
        }

        t = p->x;
        p->x = p->y;
        p->y = t;
    }
}

void Divide(int n, int d, point *p)
```

```
{
   int s = 1, t = d, rx, ry;
   p->x = 0;
   p->y = 0;
   while (s < n)
   {
        rx = 1 & (t / 2);
        ry = 1 & (t ^ rx);
        Turn(s, p, rx, ry);
        p->x += s * rx;
        p->y += s * ry;
        t /= 4;
        s *= 2;
   }
}
```

Output :



Program 6.3.2 : Write a program to draw following type of curve : Koch curve. **(Lab 16(a))**

Solution :

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

void KochCurve(int, int, int, int, int);
void main()
{
   int gdriver = DETECT, gm;
```

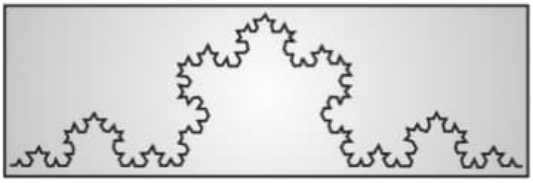
```
    initgraph(&gdriver, &gm, "");
    int x1, y1, x2, y2, Level = 3;
    //x1 = 10, y1 = 200, x2 = 600, y2 = 200;
    printf("Enter x1 :--> ");
    scanf("%d", &x1);
    printf("Enter y1 :--> ");
    scanf("%d", &y1);
    printf("Enter x2 :--> ");
    scanf("%d", &x2);
    printf("Enter y2 :--> ");
    scanf("%d", &y2);
    cleardevice();
    KochCurve(x1, y1, x2, y2, Level);
    getch();
}
void KochCurve(int x1, int y1, int x2, int y2, int Level)
{
    float angle = 60*3.14/180;
    int x3 = (2*x1+x2)/3;
    int y3 = (2*y1+y2)/3;
    int x4 = (x1+2*x2)/3;
    int y4 = (y1+2*y2)/3;

    int x = x3 + (x4-x3)*cos(angle)+(y4-y3)*sin(angle);
    int y = y3 - (x4-x3)*sin(angle)+(y4-y3)*cos(angle);

    if(Level > 0)
    {
        KochCurve(x1, y1, x3, y3, Level-1);
        KochCurve(x3, y3, x, y, Level-1);
        KochCurve(x, y, x4, y4, Level-1);
        KochCurve(x4, y4, x2, y2, Level-1);
    }
    else
    {
        line(x1, y1, x3, y3);
        line(x3, y3, x, y);
```

```
        line(x, y, x4, y4);
        line(x4, y4, x2, y2);
    }
}
```

Output :

```
Enter x1 :--> 10
Enter y1 :--> 200
Enter x2 :--> 600
Enter y2 :--> 200
```

(a) Coordinates of line



(b) Generated Koch Curve

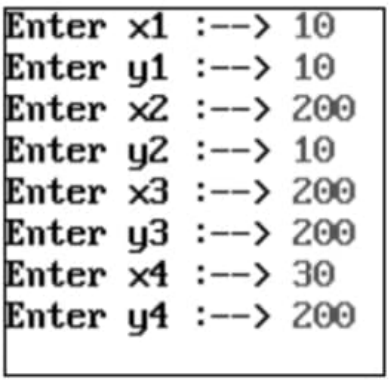
Program 6.3.3 : Write a program to draw following type of curve Bezier curve. **(Lab 16(b))**

Solution :

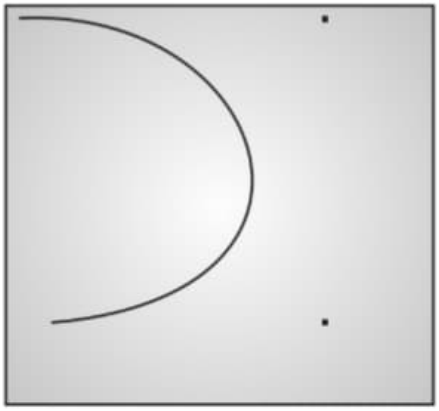
```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include <dos.h>
void Bezier(int [], int []);

void main()
{
    int gdriver = DETECT, gm;
    initgraph (&gdriver, &gm, "");
    int x[4], y[4], i;
     for (i=0; i<4; i++)
    {
        printf("Enter x%d :--> ", i+1);
        scanf ("%d", &x[i]);
        printf("Enter y%d :--> ", i+1);
        scanf ("%d", &y[i]);
     }
    cleardevice();
    Bezier (x, y);
    getch();
    closegraph();
}
```

```
void Bezier (int x[4], int y[4])
{
    int i;
     double t, xt, yt;
          for (t = 0.0; t < 1.0; t += 0.0005)
     {
        xt = pow (1-t, 3) * x[0] + 3 * t * pow (1-t, 2) * x[1] + 3 * pow (t, 2) * (1-t) * x[2] + pow (t, 3) * x[3];
        yt = pow (1-t, 3) * y[0] + 3 * t * pow (1-t, 2) * y[1] + 3 * pow (t, 2) * (1-t) * y[2] + pow (t, 3) * y[3];

        putpixel (xt, yt, RED);
        delay(1);
     }
      for (i=0; i<4; i++)
     {
        putpixel (x[i], y[i], 15);
     }
}
```

Output :



(a) Control point coordiantes



(b) Generated Bezier curve

Review Questions

Topic : Curve

Q. 1 Write DDA algorithm for arc generation.

Q. 2 Describe interpolation method for arc generation.

Q. 3 State the blending functions for Lagrange interpolation method.

Q. 4 What is Bezier curve? Explain with neat sketch.

Q. 5 Write a short note on cubic Bezier curve.

Q. 6 Derive the mathematical representation for Bezier curve and state its applications.

Q. 7 Explain how the Bezier curve is generated by four control point.

Q. 8 Explain blending function for Bezier curve.

Q. 9 Explain subdivision method for Bezier curve.

Q. 10 State the limitations of Bezier curve.

Q. 11 List properties of Bezier curve.

Q. 12 Write 'C' program to generate Bezier curve.

Q. 13 Discuss B spline curve generation.

Q. 14 List properties of B-Spline curve.

Q. 15 List and explain various knot vectors.

Q. 16 Differentiate: Bezier curve vs B-Spline curve.

Topic : Fractal

Q. 1 What is fractal dimension?

Q. 2 Explain Hilbert curve in detail.

Q. 3 Explain Koch curve in detail.

Q. 4 What are fractals? Derive an equation D = log N / log S. Outline the procedure of generating Koch curve.

Q. 5 Write 'C' program to generate Hilbert curve.

Q. 6 Write 'C' program to generate Koch curve.

Chapter Ends...

❑❑❑

Appendix A

Solved MSBTE Question Papers of Summer 2019 and Winter 2019

Summer 2019

Q. 1(a) Define aspect ratio. Give one example of an aspect ratio. **(2 Marks)**

Ans. :

Aspect ratio is the ratio of a number of vertical pixels to the number of horizontal pixels required to draw the same length of a line in both the direction. An aspect ratio of 9/16 means that a vertical line plotted with 9 points has the same length as a horizontal line plotted with 16 points. Generally, the distance between two vertical pixels is more than a distance between two horizontal pixels.

For example : Aspect ratio of our national flag is 3:2

Q. 1(b) List any four applications of computer graphics. **(2 Marks)**

Ans. :

Applications of computer graphics are listed here :

1. Computer Aided Design
2. Virtual Reality
3. Presentation Graphics
4. Computer Art
5. Entertainment
6. Education and Training
7. Visualization
8. Graphical user interface

Q. 1(c) Define virtual reality. List any two advantages of virtual reality. **(Section 1.11.1.1)** **(2 Marks)**

Ans. : Advantages :

1. Provides the realistic scenario.
2. Provides the safe and controlled simulation environment.
3. Suitable for different learning styles.
4. Makes learning more efficient and interesting.

Q. 1(d) List any two line drawing algorithms. Also, list two merits of any line drawing algorithm. **(2 Marks)**

Ans. :

Line drawing algorithms with their merits are listed here:

1. **DDA Line Drawing Algorithm**
   - It is simple and easy to understand.
   - It eliminates multiplications involved in explicit line drawing equation, $y = mx + c$.
2. **Bresenham's Line Drawing algorithm**
   - It involves integer calculation only
   - It is faster than DDA

Q. 1(e) Define convex and concave polygons. **(Sections 2.4.1.1 and 2.4.1.2)** **(2 Marks)**

Q. 1(f) What is homogeneous co-ordinate ? Why is it required ? **(Section 3.6)** **(2 Marks)**

Q. 1(g) Write the transformation matrix for y-shear. **(Section 3.9.2)** **(2 Marks)**

Q. 2(a) Compare Vector scan display and raster scan display. (Write any 4 points.) **(Section 1.8.4)** **(4 Marks)**

Q. 2(b) Rephrase the Bresenham's algorithm to plot $1/8^{th}$ of the circle and write the algorithm required to plot the same. **(4 Marks)**

Ans. :

Algorithm to draw a $1/8^{th}$ of the circle using Bresenham's circle drawing method is described here

```
Algorithm   BRESENHAM_CIRCLE(x, y, r)
                x ← 0
                y ← r
                S ← 3 – 2r
            EightWaySymmetry(x, y)
            while (x ≤ y) do
                x ← x + 1
            if S < 0      then
                S ← S + 4x + 6
            else
```

```
                y ← y – 1
                S ← S + 4 (x – y) + 10
            end
        EightWaySymmetry(x, y)
end

FUNCTION EightWaySymmetry(x, y)
putPixel(x, y)
putPixel(x, – y)
putPixel(– x, y)
putPixel(– x, – y)
putPixel(y, x)
putPixel(y, – x)
putPixel(– y, x)
putPixel(– y, – x)
```

Q. 2(c) Translate the polygon with co-ordinates A(3, 6), B(8, 11) and C(11, 3) by 2 units in X-direction and 3 units in Y-direction. **(4 Marks)**

Ans. :

Coordinates of given polygons are A(3, 6), B(8, 11) and C(11, 3). The polygon is to be shifted by 2 units in X-direction and 3 units in Y-direction, so $t_x = 2$ and $t_y = 3$.

The desired transformation operation is given as,

$$P' = T \cdot P$$

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

$$= \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 8 & 11 \\ 6 & 11 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 5 & 10 & 13 \\ 9 & 14 & 6 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A (3, 6)	A′ (5, 9)
B (8, 11)	B′ (10, 14)
C (11, 3)	C′ (13, 6)

Q. 2(d) Write the midpoint subdivision algorithm for line clipping. **(Section 5.3.2.2)** **(4 Marks)**

Q. 3(a) State the different character generation methods. Describe any one with diagram. **(Sections 2.8 to 2.8.3)** **(4 Marks)**

Q. 3(b) Obtain a transformation matrix for rotating an object about a specified pivot point. **(Section 3.5.2)** **(4 Marks)**

Q. 3(c) Describe Sutherland-Hodgeman algorithm for polygon clipping. **(Section 5.4.2)** **(4 Marks)**

Q. 3(d) 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. **(4 Marks)**

Ans. :

Curve passes through the end points (1, 1) and (3, 1)

Matrix representation of Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \; t^2 \; t \; 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 3 \\ 3 & 1 \end{bmatrix}$$

$$= [t^3 \; t^2 \; t \; 1] \begin{bmatrix} -4 & 0 \\ 3 & -6 \\ 3 & 6 \\ 1 & 1 \end{bmatrix}$$

$\therefore$ Equation of Bezier curve would be

$$Q(t) = [-4t^3 + 3t^2 + 3t + 1 \quad -6t^2 + 6t + 1]$$

We have to find at least five points on curve. Let us find point on curve for parameter step value 0.2.

$t = 0.0 \Rightarrow Q(0.0)$

$= [-4(0.0)^3 + 3(0.0)^2 + 3(0.0) + 1 \quad -6(0.0)^2 + 6(0.0) + 1]$

$= [0 + 0 + 0 + 1 \quad 0 - 0 + 1]$

$= [1 \quad 1]$

$t = 0.2 \Rightarrow Q(0.2)$

$= [-4(0.2)^3 + 3(0.2)^2 + 3(0.2) + 1 \quad -6(0.2)^2 + 6(0.2) + 1]$

$= [-0.032 + 0.12 + 0.6 + 1 \quad -0.24 + 1.2 + 1]$

$= [1.69 \quad 1.96]$

$t = 0.4 \Rightarrow Q(0.4)$

$= [-4(0.4)^3 + 3(0.4)^2 + 3(0.4) + 1 \quad -6(0.4)^2 + 6(0.4) + 1]$

$= [-0.26 + 0.48 + 1.2 + 1 \quad -0.96 + 2.4 + 1]$

$= [2.42 \quad 2.44]$

$t = 0.6 \Rightarrow Q(0.6)$

$= [-4(0.6)^3 + 3(0.6)^2 + 3(0.6) + 1 \quad -6(0.6)^2 + 6(0.6) + 1]$

$= [-1.44 + 1.8 + 1.8 + 1 \quad -2.16 + 3.6 + 1]$

$= [3.16 \quad 2.44]$

$t = 0.8 \Rightarrow Q(0.8)$

$= [-4(0.8)^3 + 3(0.8)^2 + 3(0.8) + 1 \quad -6(0.8)^2 + 6(0.8) + 1]$

$= [-2.048 + 1.92 + 2.4 + 1 \quad -3.84 + 4.8 + 1]$

$= [3.27 \quad 1.96]$

$t = 1.0 \Rightarrow Q(1.0)$

$= [-4(1)^3 + 3(1)^2 + 3(1) + 1 \quad -6(1)^2 + 6(1) + 1]$

$= [-4 + 3 + 3 + 1 \quad -6 + 6 + 1]$

$= [3 \quad 1]$

Value of t	Point on curve
t = 0.0	[1 1]
t = 0.2	[1.69 1.96]
t = 0.4	[2.42 2.44]
t = 0.6	[3.16 2.44]
t = 0.8	[3.27 1.96]
t = 1.0	[3 1]

Q. 4(a) Describe the vector scan display technique with neat diagram. **(Section 1.8.3)** **(4 Marks)**

Q. 4(b) Consider the line from (0, 0) to (4, 6). Use the simple DDA algorithm to rasterize this line. **(Example 2.2.1)** **(4 Marks)**

Q. 4(c) Consider a square A(1, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square by 45° in anti-clockwise direction followed by reflection about X-axis. **(4 Marks)**

Ans. :

Unit square is to be rotated by 45° in anti-clock wise direction followed by reflection about X-axis. The said transformation can be carried out by following sequence of operations.

$P' = Ref_{(Y=0)} \cdot R_{(\theta = 45^\circ)} \cdot P$

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

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

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.71 & 0 & -0.71 & 0 \\ 0.71 & 0 & 0.71 & 1.42 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.71 & 0 & -0.71 & 0 \\ -0.71 & 0 & -0.71 & -1.42 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original co-ordinates	Transformed co-ordinates
A (1, 0)	A′ (0.71, –0.71)
B (0, 0)	B′ (0, 0)
C (0, 1)	C′ (–071, –0.71)
D(1, 1)	D′ (0, –1.42)

Q. 4(d) Use the Cohen-Sutherland outcode algorithm to clip line $P_1(40, 15) - P_2(75, 45)$ and $P_3(70, 20) - P_4(100, 10)$ against a window A(50, 10), B(80, 10), C(80, 40) and D(50, 40). **(Example 5.3.1)** **(4 Marks)**

Q. 4(e) What is interpolation ? Describe the Lagrangian interpolation method. **(Sections 6.1.2 and 6.1.2.1)** **(4 Marks)**

Q. 5(a) Consider the line from (5, 5) to (13, 9). Use the Bresenham's algorithm to rasterize the line. **(6 Marks)**

Ans. :

Let us assume $(x_1, y_1) = (5, 5)$ and $(x_2, y_2) = (13, 9)$

$dx = |x_2 - x_1| = |13 - 5| = 8,$

$dy = |y_2 - y_1| = |9 - 5| = 4$

Initial decision parameter, $p_0 = 2dy - dx = 8 - 8 = 0$

$\Delta E = 2dy = 8$

(Increment if E pixel is selected)

$\Delta NE = 2(dy - dx) = 2(4 - 8) = -8$

(Increment if NE pixel is selected)

Table 1-Q. 5(a)

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	-	-	5	5	$p_k = p_0 = 2dy - dx = 0$
1	(5, 5)	$p_k \geq 0$, so select NE	5 + 1 = 6	5 + 1 = 6	$p_k = p_k + \Delta NE = 0 - 8 = -8$
2	(6, 6)	$p_k < 0$, so select E	6 + 1 = 7	6 + 0 = 6	$p_k = p_k + \Delta E = -8 + 8 = 0$
3	(7, 6)	$p_k \geq 0$, so select NE	7 + 1 = 8	6 + 1 = 7	$p_k = p_k + \Delta NE = 0 - 8 = -8$
4	(8, 7)	$p_k < 0$, so select E	8 + 1 = 9	7 + 0 = 7	$p_k = p_k + \Delta NE = -8 + 8 = 0$
5	(9, 7)	$p_k \geq 0$, so select NE	9 + 1 = 10	7 + 1 = 8	$p_k = p_k + \Delta NE = 0 - 8 = -8$
6	(10, 8)	$p_k < 0$, so select E	10 + 1 = 11	8 + 0 = 8	$p_k = p_k + \Delta NE = -8 + 8 = 0$
7	(11, 8)	$p_k \geq 0$, so select NE	11 + 1 = 12	8 + 1 = 9	$p_k = p_k + \Delta NE = 0 - 8 = -8$
8	(12, 9)	$p_k < 0$, so select E	12 + 1 = 13	9 + 0 = 9	$p_k = p_k + \Delta NE = -8 + 8 = 0$
9	(13, 9)	-	-	-	-

In table 1-Q. 5(a), column 2, i.e. (x_k, y_k) are the points to be displayed.

Q. 5(b) Apply the shearing transformation to square with A(0, 0), B(1, 0), C(1, 1), D(0, 1) as given below :

(i) Shear Parameter value of 0.5 relative to the line $Y_{ref} = -1$.

(ii) Shear Parameter value of 0.5 relative to the line $X_{ref} = -1$

(Example 3.9.5) **(6 Marks)**

Q. 5(c) Write a program in 'C' to generate Hilbert's curve. **(Program 6.3.1)** **(6 Marks)**

Q. 6(a) Write a program in 'C' for DDA circle drawing algorithm. **(6 Marks)**

Ans. :

```
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<math.h>
#include<graphics.h>
#define ROUND(a) ((int) (a+0.5))
void lineDDA(int xa,int ya, int xb, int yb)
{
 int dx=xb, dy=yb-ya, steps, k;
 float xi,yi, x=xa, y=ya;
 if(abs(dx)>abs(dy))
  steps=abs(dx);
 else
  steps = abs(dy);
  xi=dx/(float) steps;
  yi=dy/(float) steps;
  putpixel(ROUND(x),ROUND(y),CYAN);
  for(k=0;k<steps;k++)
   {
   x+=xi;
   y+=yi;
  putpixel(ROUND(x),ROUND(y),CYAN);
   }
}
void main()
{
        clrscr();
        int gdriver = DETECT, gmode, errorcode;
        initgraph(&gdriver, &gmode, "C:\\TC\\BGI");
        int xa=30, ya=300, xb=200, yb=50;
        lineDDA(xa,ya,xb,yb);
        getch();
        closegraph();
}
```

Q. 6(b) Perform a 45° rotation of a triangle A(0, 0), B(1, 1), C(5, 2).

(i) About the origin

(ii) About P(–1, –1)

(Example 3.7.4) **(6 Marks)**

Q. 6(c) Apply the Liang-Barsky algorithm to the line with co-ordinates (30, 60) and (60, 25) against the window : (X_{min}, Y_{min}) = (10, 10) and (X_{max}, Y_{max}) = (50, 50).

Ans. :

Here, $Xw_{min} = 10$, $Yw_{min} = 10$, $Xw_{max} = 50$, $Yw_{max} = 50$.

End points of line are A = (x_1, y_1) = (30, 60) and B = (x_2, y_2) = (60, 25).

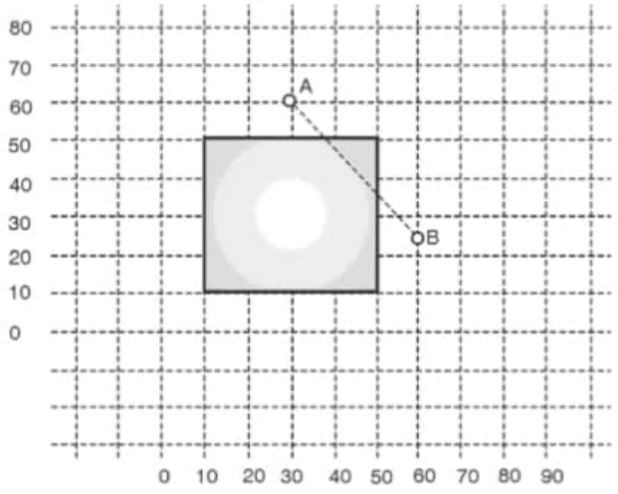


Fig. 1 - Q. 6(c)

Let us compute p_i, q_i and r_i for i = 1, 2, 3, 4

$$p_1 = -(\Delta x) = -(x_2 - x_1)$$
$$= -(60 - 30) = -30$$
$$p_2 = \Delta x = 30$$
$$p_3 = -\Delta y = -(y_2 - y_1)$$
$$= -(25 - 60) = 35$$
$$p_4 = \Delta y = -35$$

None of the p_i is zero, so line is neither horizontal, nor vertical. Let us compute q_i.

$$q_1 = x_1 - xw_{min} = 30 - 10 = 20$$
$$q_2 = xw_{max} - x_1 = 50 - 30 = 20$$
$$q_3 = y_1 - yw_{min} = 60 - 10 = 50$$
$$q_4 = yw_{max} - y_1 = 50 - 60 = -10$$

$$r_1 = \frac{q_1}{p_1} = -\frac{20}{30} = -0.67$$
$$r_2 = \frac{q_2}{p_2} = \frac{20}{30} = 0.67$$
$$r_3 = \frac{q_3}{p_3} = \frac{50}{35} = 1.43$$
$$r_4 = \frac{q_4}{p_4} = \frac{10}{35} = 0.29$$
$$u_1 = \max(0, \text{all } r_i \text{ having } p_i < 0)$$
$$= \max(0, r_1, r_4)$$
$$= \max(0, -0.67, 0.29) = 0.29$$
$$u_2 = \min(1, \text{all } r_i \text{ having } P_i > 0)$$
$$= \min(1, r_2, r_3)$$
$$= \min(1, 0.67, 1.43) = 0.67$$
$$x_1' = x_1 + \Delta x \cdot u_1 = 30 + (30) \times (0.29)$$
$$= 38.7 \cong 39$$
$$y_1' = y_1 + \Delta y \cdot u_1 = 60 + (-35) \times (0.29)$$
$$= 49.85 = 50$$
$$x_2' = x_1 + \Delta x \cdot u_2 = 30 + (30) \times (0.67)$$
$$= 50.1 \cong 50$$
$$y_2' = y_1 + \Delta y \cdot u_2 = 60 + (-35) \times (0.67)$$
$$= 36.55 = 37$$

∴ Visible line segment is (39, 50) to (50, 37)

Original line Segments	Visible line segments
A(30, 60) and B(60, 25)	A′(39, 50) and B′ (50, 37)

Winter 2019

Q. 1(a) Give two applications of computer graphics. **(Section 1.7)** **(2 Marks)**

Q. 1(b) List / name two line drawing algorithms. **(Section 2.2)** **(2 Marks)**

Q. 1(c) Explain the need of homogeneous co-ordinates matrix. **(Section 3.6)** **(2 Marks)**

Q. 1(d) Define Polygon Clipping. **(Section 5.4.1)** **(2 Marks)**

Q. 1(e) Draw Cubic Bezier Curve. **(Section 6.2.1.1)** **(2 Marks)**

Ans. :

Cubic Bezier curve is generated using four vertices. Example of such curve is shown in Fig. 1-Q. 1(e).

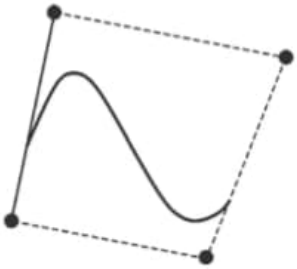


Fig. 1-Q. 1(e) : Cubic Bezier Curve

Q. 1(f) Define Bitmap graphics. **(Section 1.6.1) (2 Marks)**

Q. 1(g) List various character generation methods. **(Section 2.8) (2 Marks)**

Q. 2(a) Write short note on Augmented Reality. **(Sections 1.11.2 to 1.11.2.2) (4 Marks)**

Q. 2(b) Explain scan line algorithm of polygon clipping. **(Section 2.5.3) (4 Marks)**

Q. 2(c) Write 2D and 3D scaling matrix. **(4 Marks)**

Ans. :

2D scaling matrix :

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

3D scaling matrix :

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

S_x, S_y and S_z represents scaling parameters in x, y and z directions, respectively.

Q. 2(d) Explain midpoint subdivision line clipping algorithm. **(Section 5.3.2.2) (4 Marks)**

Q. 2(e) Explain interpolation technique in curve generation. **(Section 6.1.2) (4 Marks)**

Q. 3(a) Explain with diagram the technique of Raster Scan Display. **(Section 1.8.2) (4 Marks)**

Q. 3(b) Write procedure to fill polygon with flood fill. **(Section 2.5.2.2) (4 Marks)**

Q. 3(c) Explain 2D transformation with its types. **(Sections 3.3, 3.4, 3.5, 3.8 and 3.9) (4 Marks)**

Q. 3(d) Explain Koch curve with diagram. **(Section 6.2.2.2) (4 Marks)**

Q. 3(e) Explain Text Clipping. **(Sections 5.5 to 5.5.3) (4 Marks)**

Q. 4(a) Explain inside and outside test for polygon. **(Sections 2.4.2 to 2.4.2.2) (4 Marks)**

Q. 4(b) Explain composite transformation over arbitrary point. **(Section 3.7) (4 Marks)**

Q. 4(c) Use the Cohen-Sutherland algorithm to clip two lines P1(35, 10), – P2 (62, 40) and P3 (65, 20) – P4 (95, 10) against a window A (50, 10), B(80,10), C(80, 40), D(50,40). **(4 Marks)**

Ans. :

Spatial position of lines and rectangle is shown in Fig. 1-Q. 4(c).

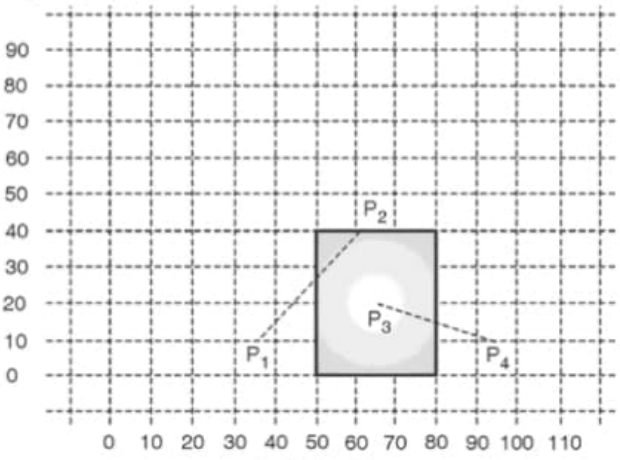


Fig. 1-Q. 4(c)

Line P_1P_2 : Given that, end points of line are $P_1(35, 10)$ and $P_2(62, 40)$

Outcode (P_1) = $0001 = A_1$

Outcode (P_2) = $0000 = B_1$

A_1 OR $B_1 = 0001$ OR $0000 = 0001$

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

A_1 AND $B_1 = 0001$ AND $0000 = 0000$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint A is 0001, that means line intersects left edge while moving from P_1 to P_2. Let us call the intersection of point P_1'. P_1' is on line $x = x_{min} = 50$. So its x-coordinate is 50, we have to compute only y coordinate of P_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{40 - 10}{62 - 35} = \frac{30}{27} = 1.11$$

$$\text{From } y = mx + c,\ c = y - mx$$

By putting one of the points of line AB in this equation, we can find intercept of line with Y axis. Let's put P_1 (35, 10) in above equation,

$$\therefore\ c = 10 - (1.11) \times (35) = 10 - 38.85$$
$$= -28.58$$

Put $x = x_{min} = 50$ in explicit line equation to compute corresponding y-coordinate

$$y = mx + c = 1.11 \times (50) - 28.58$$
$$= 55.5 - 28.58 = 26.92$$

Thus, $P_1' = (35, 26.92)$

So, line segment with end points $P_1'(35, 26.92)$ and $P_2(62, 40)$ is the visible segment.

Line P_3P_4 :

Given that, endpoints of the line are $P_3(65, 20)$ and $P_4(95, 10)$

Outcode $(P_3) = 0000 = C_1$

Outcode $(P_4) = 0010 = D_1$

C_1 OR $D_1 = 0000$ OR $0010 = 0010$

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

C_1 AND $D_1 = 0000$ AND $0010 = 0000$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{10 - 20}{95 - 65} = \frac{-10}{30} = -0.33$$

$$\text{From, } y = mx + c,\ c = y - mx$$

By putting one of the points of line P_3P_4 in this equation, we can find intercept of line with Y axis. Let's put $P_3(65, 20)$ in above equation,

$$\therefore\ c = 20 - (-0.33) \times (65) = 20 + 21.45 = 41.45$$

Outcode of end point P_4 is 0010, that means line intersects right edge. Let us call the intersection of point P_4'.

$$\text{From, } y = mx + c,$$

By putting the value of x of a right edge in this equation, we can find the intersection y coordinate of P_4'.

$$\therefore\ y = (-0.33 \times 95) + 41.45$$
$$= -31.35 + 41.45 = 10.1$$

So, line segment with end points $P_3(60, 20)$ and $P_4'(10.1, 10)$ is the visible segment.

Original line Segments	Visible line segments
$P_1(35, 10)$ and $P_2(62, 40)$	$P_1'(35, 26.92)$ and $P_2'(62, 40)$
$P_3(65, 20)$ and $P_4(95, 10)$	$P_3'(65, 20)$ and $P_3'(80, 10.1)$

Q. 4(d) Write DDA Arc generation algorithm. **(Section 6.1.1)** **(4 Marks)**

Q. 5(a) Use Bresenham's line drawing algorithm to rasterize line from (6, 5) to (15, 10). **(6 Marks)**

Ans. :

Let us assume $(x_1, y_1) = (6, 5)$ and $(x_2, y_2) = (15, 10)$

$$dx = |x_2 - x_1| = |15 - 6| = 9,$$
$$dy = |y_2 - y_1| = |10 - 5| = 5$$

Initial decision parameter, $p_0 = 2dy - dx = 10 - 9 = 1$

$$\Delta E = 2dy = 10$$

(Increment if E pixel is selected)

$$\Delta NE = 2(dy - dx) = 2(5 - 9) = -8$$

(Increment if NE pixel is selected)

Table 1-Q. 5(a)

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	-	-	6	5	$p_k = p_0 = 2dy - dx = 1$
1	(6, 5)	$p_k \geq 0$, so select NE	6 + 1 = 7	5 + 1 = 6	$p_k = p_k + \Delta NE$ $= 1 - 8 = -7$
2	(7, 6)	$p_k < 0$, so select E	7 + 1 = 8	6 + 0 = 6	$p_k = p_k + \Delta E$ $= -7 + 10 = 3$
3	(8, 6)	$p_k \geq 0$, so select NE	8 + 1 = 9	6 + 1 = 7	$p_k = p_k + \Delta NE$ $= 3 - 8 = -5$
4	(9, 7)	$p_k < 0$, so select E	9 + 1 = 10	7 + 0 = 7	$p_k = p_k + \Delta E$ $= -5 + 10 = 5$
5	(10, 7)	$p_k \geq 0$, so select	10 + 1 =	7 + 1 =	$p_k = p_k + \Delta NE$

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
		NE	11	8	$= 5 - 8 = -3$
6	(11, 8)	$p_k < 0$, so select E	11 + 1 = 12	8 + 0 = 8	$p_k = p_k + \Delta E$ $= -3 + 10 = 7$
7	(12, 8)	$p_k \geq 0$, so select NE	12 + 1 = 13	8 + 1 = 8	$p_k = p_k + \Delta NE$ $= 7 - 8 = -1$
8	(13, 9)	$p_k < 0$, so select E	13 + 1 = 14	9 + 0 = 9	$p_k = p_k + \Delta E$ $= -1 + 10 = 9$
9	(14, 9)	$p_k \geq 0$, so select NE	14 + 1 = 15	9 + 1 = 10	$p_k = p_k + \Delta NE$ $= 9 - 8 = 1$
9	(15, 10)	-	-	-	-

In Table 1-Q. 5(a), column 2, i.e. (x_k, y_k) are the points to be displayed.

Q. 5(b) Find the transformation of triangle A(1, 0) B(0, 1) C(1, 1) by, **(6 Marks)**

(i) Rotating 30° about the origin.

(ii) Translating one unit x and y direction and then rotate 45° about origin.

Ans. :

1. Rotation by 30° about origin

The said transformation is carried out as

$$P' = R_{(\theta = 30^\circ)} \cdot P$$

$$= \begin{bmatrix} \cos(30^\circ) & -\sin(30^\circ) & 0 \\ \sin(30^\circ) & \cos(30^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.87 & -0.5 & 0 \\ 0.5 & 0.87 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.87 & -0.5 & 0.37 \\ 0.5 & 0.87 & 0.37 \\ 0 & 0 & 1 \end{bmatrix}$$

2. Translating one unit x and y direction and then rotate 45° about origin.

The said transformation is carried out as

$$P' = R_{(\theta = 45^\circ)} \cdot T \cdot P$$

$$= \begin{bmatrix} \cos(45^\circ) & -\sin(45^\circ) & 0 \\ \sin(45^\circ) & \cos(45^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.71 & -0.71 & 0 \\ 0.71 & 0.71 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0.71 & -0.71 & 0 \\ 2.13 & 2.13 & 2.84 \\ 0 & 0 & 1 \end{bmatrix}$$

Q. 5(c) Write C program for Hilbert's curve. **(Program 6.3.1)** **(6 Marks)**

Q. 6(a) Explain character generation methods :

(i) Stroke. **(Section 2.8.1)**

(ii) Starburst. **(Section 2.8.2)**

(iii) Bitmap. **(Section 2.8.3)** **(6 Marks)**

Q. 6(b) Apply shearing transformation to square with A(0, 0), B(1, 0), C(1, 1) and D(0, 1) as shear parameter value of 0.5 relative to the line $Y_{ref} = -1$ and $X_{ref} = -1$. **(Example 3.9.5)** **(6 Marks)**

Q. 6(c) Explain Cyrus beck line clipping algorithm. **(Section 5.3.2.3)** **(6 Marks)**

❑❑❑

Note

Note