Syllabus

Mumbai University
Revised syllabus (Rev-2016) from Academic Year 2017-18

Computer Graphics

Course Code	Course Name	Credits
CSC404	Computer Graphics	4

Course Objectives

1. To equip students with the fundamental knowledge and basic technical competence in the field of computer graphics.
2. To emphasize on implementation aspect of Computer Graphics Algorithms.
3. To prepare the student for advance areas like Image Processing or Computer Vision or Virtual Reality and professional avenues in the field of Computer Graphics.

Course Outcomes : At the end of the course student should be able to

1. Understand the basic concepts of Computer Graphics.
2. Demonstrate various algorithms for scan conversion and filling of basic objects and their comparative analysis.
3. Apply geometric transformations, viewing and clipping on graphical objects.
4. Explore solid model representation techniques and projections.
5. Understand visible surface detection techniques and illumination models.

Prerequisite : Knowledge of C Programming, Basic Data Structures and Mathematics.

Module No.	Detail Syllabus	Hours
1.	**Introduction and Overview of Graphics System :** • Definition and Representative uses of computer graphics, classification of application areas, Overview of coordinate systems ,definition of scan conversion, rasterization and rendering. • Raster scan & random scan displays, Flat Panel displays like LCD and LED , architecture of raster graphics system with display processor, architecture of random scan systems. **(Refer Chapter 1)**	03
2.	**Output Primitives :** • Scan conversions of point,line, circle and ellipse : DDA algorithm and Bresenham algorithm for line drawing, midpoint algorithm for circle, midpoint algorithm for ellipse drawing (Mathematical derivation for above algorithms is expected) • Aliasing , Antialiasing techniques like Pre and post filtering , super sampling , and pixel phasing). • Filled Area Primitive: Scan line Polygon Fill algorithm, Inside outside tests, Boundary Fill and Flood fill algorithm. **(Refer Chapters 2 and 3)**	12

Module No.	Detail Syllabus	Hours
3.	**Two Dimensional Geometric Transformations** • Basic transformations : Translation , Scaling , Rotation • Matrix representation and Homogeneous Coordinates • Composite transformation • Other transformations : Reflection and Shear • Raster method for transformation. **(Refer Chapter 4)**	06
4.	**Two Dimensional Viewing and Clipping** • Viewing transformation pipeline and Window to Viewport coordinate transformation • Clipping operations – Point clipping , Line clipping algorithms : Cohen – Sutherland , Midpoint subdivision , Liang – Barsky , Polygon Clipping Algorithms : Sutherland – Hodgeman, Weiler – Atherton. **(Refer Chapter 5)**	08
5.	**Three Dimensional Object Representations , Geometric Transformations and 3D Viewing** • Boundary Representation and Space partitioning representation: Polygon Surfaces , Bezier Curve , Bezier Surface , B-Spline Curve , Sweep Representation, Constructive Solid Geometry ,Octree, Fractal-Geometry : Fractal Dimension, Koch Curve. • 3D Transformations :Translation, Rotation , Scaling and Reflection. • Composite transformations :Rotation about an arbitrary axis • 3D transformation pipeline • Projections – Parallel , Perspective.(Matrix Representation) • 3D clipping **(Refer Chapter 6)**	12
6.	**Visible Surface Detection** • Classification of Visible Surface Detection algorithm • Back Surface detection method • Depth Buffer method • Depth Sorting method • Scan line method • Area Subdivision method **(Refer Chapters 7 and 8)**	04
7.	**Illumination Models and Surface Rendering** • Basic Illumination Models : Diffused reflection, Phong Specular reflection Model • Halftone and Dithering techniques • Polygon Rendering :Constant shading , Gouraud Shading , Phong Shading. **(Refer Chapter 9)**	03

□□□

Lab Code	Lab Name	Credit
CSL402	Computer Graphics Lab	1

Lab Objectives

1. To emphasize on implementation aspect of Computer Graphics Algorithm.
2. To prepare students for advanced areas like Animation, image processing ,virtual reality etc

Lab Outcomes : At the end of the course , the students should be able to

1. Explore the working principle, utility of various input/ output devices and graphical tools.
2. Implement various output and filled area primitive algorithms using C/ OpenGL
3. Apply transformation and clipping algorithms on graphical objects.
4. Implementation of curve and fractal generation.
5. Develop a Graphical application based on learned concept.

Contents :

Scan conversions: lines, circles, ellipses. Filling algorithms, clipping algorithms. 2D and 3D transformation. Curves. Visible surface determination. Simple animations. Application of these through exercises in C/C++/ Open GL

List of Desirable Experiments :

1. Study and apply basic opengl functions to draw basic primitives. (*)
2. Implement sierpinsky gasket using openGL.
3. Implement DDA Line Drawing algorithms and Bresenham algorithm(*)
4. Implement midpoint Circle algorithm(*)
5. Implement midpoint Ellipse algorithm
6. Implement Area Filling Algorithm: Boundary Fill, Flood Fill ,Scan line Polygon Fill (*)
7. Implement Curve : Bezier for n control points , B Spline (Uniform) (atleast one)(*)
8. Implement Fractal (Koch Curve)
9. Character Generation : Bit Map method and Stroke Method
10. Implement 2D Transformations: Translation, Scaling, Rotation, Reflection, Shear.(*)
11. Implement Line Clipping Algorithm: Cohen Sutherland / Liang Barsky.(*)
12. Implement polygon clipping algorithm (atleast one)
13. Program to represent a 3D object using polygon surfaces and then perform 3D transformation.
14. Program to perform projection of a 3D object on Projection Plane : Parallel and Perspective.(*)

□□□

Module 1

Module 2

Module 3

Module 4

List of Experiments

❑❑❑

Module 1

CHAPTER 1

Introduction to Computer Graphics

Chapter 1 gives an overview of the Computer Graphics and tells why computer graphics is important and discusses number of its applications. It also discusses different interactive devices and introduction to OpenGL. The chapter is divided into nine major sections.

1.1 **Introduction to Computer Graphics** discusses the importance of computer graphics and different applications of computer graphics.

1.2 **Elements of Pictures** explain how a particular point is displayed on screen and use of frame buffer.

1.3 **Overview of Co-ordinate Systems** explains how to plot a point in two dimension and three dimension graphics system

1.4 **Basics of Computer Graphics** explain the properties of display adapters and discuss different display modes which are frequently used. It also discusses about aspect ratio.

1.5 **Graphics Display Devices** discuss various types of interactive devices, display devices, data generating devices which are commonly used for computer graphics.

1.6 **Primitive Operations** tells the basic commands which are used for computer graphics.

1.7 **Display File and its structure** tells how to store the image and display it on output device, it also gives idea about how exactly display file interpreter works.

1.8 **Raster and Random Scan** explains the use of it and their difference with advantages and disadvantages.

1.9 **Introduction to OpenGL** briefly explains operation and features of OpenGL along with basic primitives.

Syllabus Topic : Definition and Representative Uses of Computer Graphics, Classification of Application Areas

1.1 Introduction to Computer Graphics

Q. What is computer graphics ? (3 Marks)

- Anything which is not a text on computer is treated as a computer graphics. Nowadays almost all computers are using graphical user interfaces. This is because users are not interested in doing any operation by giving textual command. Rather the operations are done by using different icons and GUI's.
- The term **Computer Graphics** has several meanings like the representation and manipulation of pictorial data by a computer, the various technologies used to create and manipulate such pictorial data, the images so produced, and the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics.
- Today computers and computer generated pictures are becoming integral part of our daily life. Computers are used in many versatile fields like television, Animation films, medical science, simulation fields, etc. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines.
- The computer is a data processing machine or we can say it is a tool which is used for storing, manipulating and displaying data. We can collect or generate data and process it according to users need. Let us say data is entered from keyboard in the form of 2 nos., say 10 and 20.
- Now we can process this data by performing some operation on it, say addition, so that we will get result as 10 + 20 = 30. Here 30 is a result. It means here we have performed addition process on input data. This data is also called as information. This information can help us to make decisions and understand the world and control the operations.

If the data elements are less in numbers then we can easily perform any operation. But as the volume of information increases a problem arises. Problems could be of type, more time required to process data, accuracy problem, understanding may be difficult etc. So, how can this large information be effectively and efficiently transferred between machine and human?

A machine can easily generate hundred lines of code or thousand entries for a table. But such a code or table may be worthless if the human reader does not have time to go through it or understand it. If the reader is not going to read all the hundred lines of codes and analyze it then he will be doubtful about the output of that code. So even if the code may be correct as reader is not reading it sincerely, he is doubtful about the code. At this stage **Computer Graphics** is very useful.

- Computer Graphics is the study of techniques to improve communication between human and machine. The word Computer Graphics means pictures, graph or scene is drawn with the help of computer. We can draw a graph which may replace that huge table of thousand entries and allow the reader to note the relevant patterns and characteristics at a glance. Here Computer represents the data provided to it in pictorial form. The input given by the user, often alter the output presented by the machine.
- A dialog can be established through the graphics medium. This is termed as **Interactive Computer Graphics**. Here user interacts with machine.e.g. Histogram
- Here we will provide input data to machine as result of F.E. to B.E.

For F.E. result is 70%

For S.E. result is 80%

For T.E. result is 65%

For B.E. result is 90%

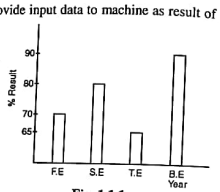


Fig. 1.1.1

- If we draw histogram with this database it will be easy to understand and analyze the result of F.E. to B.E. i.e. which class is having maximum result, which class is having minimum results etc. See Fig. 1.1.1. So, just by working at one diagram we can answer many questions. By changing the input parameters we can modify the output picture i.e. if F.E. result increases from 70% to 80% then certainly the histogram is going to change.
- Similarly the other examples could be Pie-charts, Cardiogram etc. Cardiogram takes heart pulses as input and converts that input on graph paper. Computer graphics allows communication through pictures, charts and diagrams, as there is old adage "Thousand words can be replaced by a single picture".

1.1.1 Applications of Computer Graphics

→ **(May 2016)**

Q. State various applications of Computer Graphics. MU - May 2016, 5 Marks

Q. What are different Applications of computer graphics ? (5 Marks)

- There are many areas where computer generated pictures are used. The applications of computer graphics are in a variety of field such as,

- Engineering/Scientific Software, Business Software.
- T.V. channels, space simulation training.
- PCB designing, map preparation.
- User Interface, Animation.
- Office Automation.
- Making Charts, Image Processing.
- Desktop Publishing.
- CAD/CAM.
- Art and Commerce.
- Process Controlling.
- 'Visual Effects' in Movies and Computer Games.

1.2 Elements of Pictures

- We cannot represent an infinite number of points on a computer, just as we cannot represent an infinite quantity of numbers. The machine is finite and we are limited to a finite number of points making up each line. The maximum number of distinguishable points which a line may have is a measure of the resolution of the display device.
- The greater the number of points, the higher is the resolution. Since we must build our lines from a finite number of points, each point must have some size and so is not really a point at all. It is a pixel. The pixel is the smallest addressable screen element. It is the smallest piece of the display screen which we can control. Pixel is a basic element of the picture.
- Each pixel has name or address. The names which identify pixels correspond to the coordinates which identify points. Computer graphics images are made by setting the intensity and color of the pixels which compose the screen. We draw line segments by setting the intensities that is the brightness of a string of pixels between a starting pixel and an ending pixel.
- We can think of the display screen as a grid, or array, of pixels. We shall give integer coordinate values to each pixel. Starting at the left with 1, we shall number each column. Starting at the bottom with 1, we shall number each row. The coordinate (i, j) will then give the column and row of a pixel. Each pixel will be centered at its coordinates. See Fig. 1.2.1.

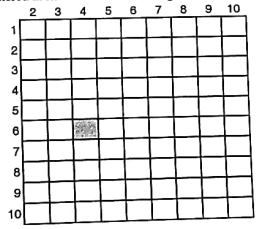


Fig. 1.2.1

- We may wish to place the intensity values for all pixels into an array in our computer's memory. Our graphics display device can then access this array to determine the intensity at which each pixel should be displayed. This array, which contains an internal representation of the image, is called the frame buffer. It collects and stores pixel values for use by the display device.
- The information in the memory buffer typically consists of color values for every pixel on the screen. Color values are commonly stored in 1-bit binary (monochrome), 4-bit, 8-bit, 16-bit high color and 24-bit true color formats. The total amount of the memory required to drive the frame buffer depends on the resolution of the output signal, and on the color depth.
- Frame buffers differ significantly from the vector displays that were common prior to the advent of the frame buffer. With a vector display, only the vertices of the graphics primitives are stored. The electron beam of the output display is then commanded to move from vertex to vertex, tracing an analog line across the area between these points.
- With a frame buffer, the electron beam is commanded to trace a left-to-right, top-to-bottom path across the entire screen, the way a television renders a broadcast signal. At the same time, the color information for each point on the screen is pulled from the frame buffer, creating a set of pixels.

Rotating Memory Frame Buffer

- This is the situation when we use Screen resolution less than the maximum screen resolution supported by the Visual Display Unit. In case of maximum screen resolution the frame buffer memory is fully utilized and only one page is available for display. For such a resolution, when it is less than the maximum resolution, more than one page is available.
- This we often use for animation using page flipping. We keep drawing on to one page which is hidden and flash it altogether when the drawing is complete with the existing one on the screen. This avoids showing of the partially drawn images. For any particular case even 3-4 pages may be available. So the system is termed as "Rotating frame buffer" in Computer Graphics. We keep rotating the pages to generate animation.
- A different situation where the frame buffer is literally rotated to generate a rotated display of any image is also sometimes referred to as Rotating frame buffer. Just changing the signs and interchanging the X-Y coordinates may generate such an effect. See Fig. 1.2.2.
- A common implementation of frame buffer is a **random access** semiconductor memory. Frame buffers can also be implemented using Shift Registers. Each Shift Register contributes one pixel in a horizontal scan line. Fig. 1.2.2 shows a shift Register implementation of frame buffer.

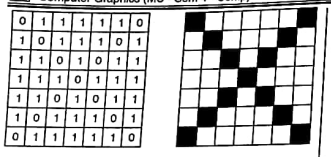


Fig. 1.2.2

- **Shift Register Frame Buffer** memory has the disadvantage of low levels of interactivity. In workstations, it is efficient to add a graphics processor with a separate frame buffer memory. The system performance can be improved by this architecture as it meets the update requirements of the frame buffer.

Syllabus Topic : Overview of Co-ordinate Systems

1.3 Overview of Co-ordinate Systems

- A co-ordinate system is a one which defines every point by a pair of numerical values, which are the distances from a fixed point in a perpendicular direction. To represent a coordinates we need reference lines. Each reference line is referred as an axis. The point where two axes are meeting is called as its origin. These axes are perpendicular to each other.
- One can use the same principle to specify the position of any point in three-dimensional space by three co-ordinates, its signed distances to three mutually perpendicular planes. In general, one can specify a point in a space of any dimension.

Co-ordinates in two dimensions

- In a two dimensional system both the axes are perpendicular to each other having single unit of length for both axes. The axis which is horizontal is referred as X axis and the axis which is vertical is referred as Y axis. The common point where these two axes meet is referred as origin.
- To represent any point we have to specify it's both X and Y co-ordinate values. Suppose a point P is represented as (2,3) then it will interpreted as the value of X is 2 units and value of Y is 3 units. So we have to plot a point P, 2 units away from the vertical axis and 3 units away from the horizontal axis. The details are shown in Fig. 1.3.1.
- The common point of both X and Y axis is referred as origin, which is also sometimes called as starting point. This origin is usually set to the value (0,0). We may represent any point in any of the four quadrants. In first quadrant both X and Y values are positive. In second quadrant X is negative and Y is positive, whereas in third quadrant both X and Y values are negative. In fourth quadrant X is positive and Y is negative.

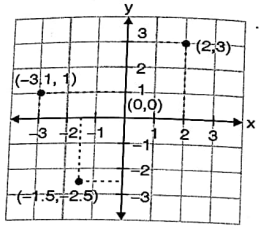


Fig. 1.3.1

Co-ordinates in three dimensions

- Coordinates for three dimensions is just the extension of two dimensions. Here we are having three major axes instead of two like in 2D system. In 3D we are having X, Y and Z axis. All these three axes are perpendicular to each other. In 3D usually X axis represents the width of the object, Y axis represents the height of the object and Z axis represents depth of the object.
- If we want to represent any point in 3D we have to specify (x,y,z) values of that point. Suppose we represent a point A(3,2,1) then it means point A is 3 units away from the origin in X direction, 2 units away from origin in Y direction and 1 unit away from origin in Z direction. Point A(3,2,1) is displayed in Fig. 1.3.2.

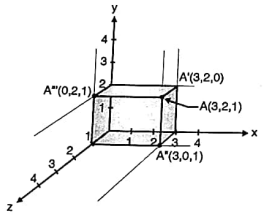


Fig. 1.3.2

- In 3D also we can have negative X or negative Y or negative Z axis coordinates. Accordingly we need to plot the specified point. In addition to this we are using Homogeneous Coordinate system and Normalized Device Coordinate Systems also. These coordinate systems will be seen in next chapters.

1.4 Basics of Graphics

1.4.1 Video Adapters

- A video adapter or display adapter is an integrated circuit card in a computer that provides digital to analog converter, video RAM and a video controller so that data can be sent to a computers display.

Types of display adapters

- Display adapters are characterized by,

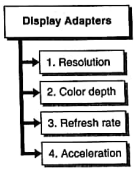


Fig. C1.1 : Display Adapters

➜ 1. Resolution

- It refers to the number of dots on the screen. It is expressed as a pair of numbers that give the number of dots on a horizontal line and the number of such vertical lines. Four resolutions are in common use today.
  - o 640 × 480
  - o 800 × 600
  - o 1024 × 768
  - o 1280 × 1024
- Computer display generates colors by combining amounts of Red, Green and Blue. These colors are controlled by 3 wires in the display cable. Each has a variable amount of voltage represented by a number from 0 to 255. This produces upto 16 million possible colors.

➜ 2. Color Depth (number of colors)

- It is determined by the number of bits assigned to hold color value.
  - o 1 bit – 2 colors (black and white)
  - o 4 bits – 16 colors
  - o 8 bits – 256 colors
  - o 16 bits – 32 thousand colors
  - o 24 bits – 16 million (high color)
  - o 32 bits – latest (true color)
- The display adapter stores a value (4 to 32 bits) in memory for every dot on the screen. The amount of storage needed is determined by multiplying the number of dots (resolution) by the memory required for each dot. e.g. for VGA having resolution 640 × 480 with 4 bit color 256 KB memory is required. SVGA having resolution 800 × 600 with 8 bits color, will need 512 KB memory.

➜ 3. Refresh rate

- It is nothing but the speed by which a particular dot on the screen is getting printed. Initially VGA display monitors were using 60 Hz frequency, but it was producing flickering effect, so later on it uses 70 Hz, to lower down this flickering effect. We can set this refresh rate by making use of drivers of a particular display adapter.

➜ 4. Accelerator

- Accelerator chip is an integrated chip existing on the display adapter. It is used to draw readymade shapes like drawing lines and boxes, filling the color in the box, filling background color and managing the mouse pointer. The CPU has to send a command only to perform a task and rest will be done by this Accelerator chip. Suppose we need to draw a line then we will send only the end points and the color of the line, the pixels on that line from starting point to end point are drawn by the Accelerator chip by a specified color.
- There is no need to calculate the intermediate points in the line by the CPU. In that way it accelerates the speed of drawing the objects. The accelerator chip also reduces the data transfer rate among the CPU and display device.

1.4.2 Modes of Resolution

- There are many video adapters which supports several different modes of resolution. Generally all these modes are grouped in two broad categories:

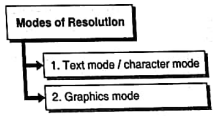


Fig. C1.2 : Modes of Resolution

➜ 1. Text Mode /Character mode

- When a display screen is divided into number of rows and columns of boxes then the mode of display is referred as a text mode. In text mode each referred box is capable to hold a single character only.
- Almost all the standard video adapters which are used for computer graphics, supports this mode. In this mode the screen is divided into 25 rows and 80 columns. In this scenario the screen of the display is considered as an array of blocks, where each block stores a single ASCII character. See Fig. 1.4.1.

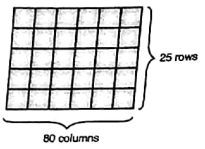


Fig. 1.4.1

➜ **2. Graphics Mode**

- In addition to text mode, most video adapters support a graphics mode, in which the display screen is divided into an array of pixels.
- Here we are introducing new term pixel.
- Pixel is the smallest addressable screen element. It is the smallest piece of the display screen which we can control. Each pixel has a name or address, so that we can uniquely identify that. Computer Graphics images are made by setting the intensity and color of the pixels.

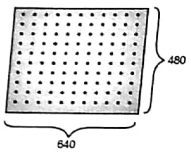


Fig. 1.4.2

- The output in graphics mode looks more perfect and smooth. We can display unlimited variety of shapes and fonts by making use of graphics mode. But in case of text mode we are getting many limitations. We can't increase the font size or change the shape of the character. Also we can't achieve the free hand shapes. All this issues can be achieved by graphics mode.

☞ **Graphics Mode function**

- We will cover some of the standard library functions here. As we have seen that, there are two different modes, namely text mode and graphics mode. In text mode it is possible to display or capture only text in terms of ASCII. But in graphics any type of figure can be displayed, captured and animated.
- "graphics.h" and "graphics.lib" are the two files which we need to carry out any graphics related operation. These are the standard header and library files. To execute the graphics program we need a display device, such as CGA, EGA, VGA, etc., which is compatible with graphics mode functions. Let us have a look at how to write a simple graphics program.

```
/* Sample example to draw line */
#include<stdio.h>
#include<graphics.h>
        void main()
        {
                int gd=DETECT,gm;
                initgraph(&gd, &gm, "c:/tc/bgi ");
                line(200,200,100,100);
                closegraph();
                restorecrtmode();
        }
```

- GRAPHICS.H, header file contains definitions of all the graphic functions and constants. While GRAPHICS.LIB file contains standard graphic functions.Here we are dealing with graphic mode function. By default system sets Text Mode. To switch from text mode to graphic mode, we have function called as "initgraph".
- **Initgraph :** Initgraph() is a readymade function which is used to initialize the graphic mode for our system. It selects the best resolution for our PC and assigns that value to mode in variable gm. The variables 'gd' and 'gm' are the two integer variables which are used for storing the graphics driver and graphics mode for your system. When we are writing, gd=DETECT, it means the highest possible value available for the detected driver is assigned to 'gd'. We can assign some fixed value also for the variable 'gd' and 'gm'. Since we are passing the address of the constants we are using '&' symbol for initgraph().
- **Path ("C:\\tc\\bgi") :** It specifies the directory path where initgraph looks for graphics drivers (*.BGI) first. If files are not there then initgraph will look for the current directory of your program. If it unable to find it then it will generate an error. We can leave it blank (" ") if the *.BGI files are within the working directory.
- **Line( ) :** inbuilt line function takes X1,Y1 and X2,Y2 values with respect to top left corner of the screen and third co-ordinate. In this example we have passed X1 = 200,Y1 =200 and X2 = 100 and Y2 = 100 in terms of pixels as arguments. The line function will draw a line for the said coordinates on the screen.
- **Closegraph( ) :** The closegraph() command is used to come out of graphics mode and sets the display mode to its original text mode. If we don't use this function then we may not be able to see the cursor blinking. Because cursor is a text mode character and not the graphics mode character. And characters could be displayed as small and bold on output window.
- **Restorcrtmode( ) :**This function is used to restore the original video mode.

- There are several display modes that can be found in personal computer.In 1970s, the computer systems were using the monochrome monitors. These monitors were just good for word processing and text based operations, because they were displaying text in black and white color only.
- **Color Graphics Adapter (CGA)** is introduced by IBM in 1981. CGA was able to display four colors. It was having maximum resolution of 320 × 200 i.e. 320 pixels horizontally and 200 pixels vertically. CGA was a good choice for simple computer games such as solitaire and checkers, But it fails to play and display higher resolution games and smooth pictures respectively.
- After that in 1984, **Enhanced Graphics Adapter (EGA)** display was introduced by the IBM. It was supporting upto 16 colors with a resolution of 640 × 350. It was much better than the earlier CGA. Both text and graphics appear much sharper on the EGA compared to the CGA. But still EGA did not offer sufficient image resolution for high-level applications such as graphic design and desktop publishing.
- In 1987, IBM introduced the **Video Graphics Array (VGA)** display system. Today almost all displays and video adapters adhere to VGA. The maximum resolution depends on the number of color displayed. You can choose between 16 colors with 640 × 480 or 256 colors with 320 × 200. All IBM compatible computers supports the VGA standard. The VGA generates analog RGB output in red, green and blue.
- In 1990, IBM introduced the **Extended Graphics Array (XGA)** and XGA-2 which offers 800 × 600 pixel resolution in true color (16 million colors) and 1024 × 768 resolution in 65,536 colors respectively. These two resolution levels are the most popular in use today by individuals and small business.
- The **Video Electronics Std. Association (VESA)** has established a standard Programming interface for **Super Video Graphics Array (SVGA)** displays, called VESA BIOS Extension. Typically, an SVGA display can support upto 16 million colors, although the amount of video memory in a particular computer may limit the actual number of displayed colors to something less than that. Image-Resolution specifications vary, typically 800 × 600. In general, the larger the diagonal screen measure of an SVGA monitor, the more pixels it can display horizontally and vertically.
- Recently, new specifications have arisen. These include **Super Extended Graphics Array (SXGA)** and **Ultra Extended Graphics Array (UXGA).** The SXGA is generally used in reference to screens with 1280 × 1024 resolution; UXGA refers to a resolution of 1600 × 1200.

Display mode	Resolution
VGA	640 × 480
SVGA	800 × 600
XGA	1027 × 768
SXGA	1280 × 1024
UXGA	1600 × 1200

☞ **VGA and SVGA**

- **VGA :** VGA Video Card contains all the circuitry needed to produce VGA graphics and like all expansion cards, it plugged into a slot on motherboard, via an 8 bit interface. Today, the VGA card is not much used and usually serves as a "spare". VGA offers clean images at higher resolutions. The standard VGA can produce as many as 256 colors at a time with 262,144 colors.
- **SVGA :** It refers to the group of video cards. SVGA was developed by third party companies in order to compete with IBM's XGA.

Adapter type	Resolution	Colors
XGA	600 × 800	16 million
	1024 × 768	65,636
SVGA	600 × 480	65,536
	600 × 800	256
	1024 × 768	16
CGA	320 × 200	4
	640 × 200	2
EGA	640 × 350	16
VGA	640 × 480	16
	320 × 200	256

- SVGA is much more advanced than VGA. In most cases, one SVGA card can produce millions of colors at a choice of resolutions. But the abilities depends on card and the manufacturer. Since SVGA is a loose term created by several companies, there is no actual standard to SVGA.

Syllabus Topic : Architecture of Raster Graphics System with Display Processor, Architecture of Random Scan Systems

1.5 Graphics Display Devices

1.5.1 Interactive Devices

- Input devices are also called as interactive devices which allow us to communicate or interact with computer.
- Using this one can feed in the data. Input devices are :

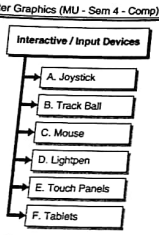


Fig. C1.3 : Interactive / Input devices

➜ 1.5.1(A) Joystick

- A joystick is nothing but a small vertical lever which is usually called as stick mounted on the base and is used to move the cursor on the screen.
- It consists of two potentiometers attached to a single lever. As we move the lever, the potentiometer setting gets changed and the x and y co-ordinates on screen get changed. Joystick is generally used for computerized games.

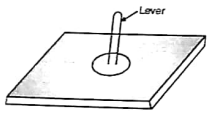


Fig. 1.5.1

➜ 1.5.1(B) Track Ball

- Trackball is some sort of an inverted mouse where the ball is held inside a rectangular box. Here, the positioning of the cursor on the screen is associated with the orientation of the trackball.
- A potentiometer captures the trackball orientation which is calibrated with the translation of the cursor on the screen. Fig. 1.5.2 gives the view of Track ball.

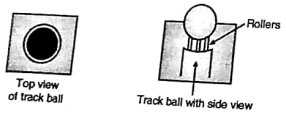


Fig. 1.5.2

➜ 1.5.1(C) Mouse

- Mouse is an integral part of the graphical user interface. There is a cursor in the shape of an arrow usually associated with mouse. There is a ball and set of rollers at the bottom of the mouse to indicate the amount and direction of movement.
- Several instructions or commands in a software are activated using a mouse. There are two or three buttons on top side of mouse. Some mouse are having scrolling button also on top side. The commands may be executed by pressing the mouse button. See Fig. 1.5.3.

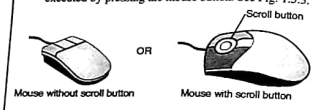


Fig. 1.5.3

➜ 1.5.1(D) Lightpen

- Lightpen consists of photocell mounted in a pencil like case. It is a pencil shaped device which is used to point a particular position on the screen. The pen will send a pulse whenever the phosphor below it is illuminated. When the light pen senses the phosphor under it is illuminated, it can interrupt the display processor to record the co-ordinate position of the electron beam.
- Lightpens are not much popular because of several disadvantages. One disadvantage is, when a lightpen is pointed at the screen, part of the screen image is obscured by the hand and pen. Another disadvantage is lightpens are not detecting positions in black areas. Sometimes lightpens give false readings due to background lighting in room.

➜ 1.5.1(E) Touch Panels

- Touch panels allow displayed objects or screen positions to be selected with a touch of a finger. A typical application of touch panels is for the selection of processing options that are represented with a graphical icon.
- Optical touch panels employ a line of infrared Light Emitting Diodes (LEDs) along one vertical and one horizontal edge of the frame. The opposite vertical and horizontal edge contains light detectors. These detectors are used to record which beams are interrupted when the panel is touched. The two crossing beams that are interrupted identify the horizontal and vertical co-ordinates of the screen position selected. Positions can be selected with an accuracy of about 0.25 inch. With closely spaced LEDs, it is possible to break two horizontal or two vertical beams simultaneously. In this case an average

position between the two interrupt beams is recorded. The LEDs operate at infrared frequencies, so the light is not visible to a user.

- An electrical touch panel is constructed with two transparent plates separated by a small distance, one of the plate is coated with a conducting material and the other plate is coated with a resistive material. When the outer plate is touched, it is forced into contact with the inner plate. This contact creates a voltage drop across the resistive plate that is converted to the co-ordinate values of the selected screen position. In acoustical touch panels, high-frequency sound waves are generated in the horizontal and vertical directions across a glass plate. Touching the screen causes part of each wave to be reflected from the finger to the emitters. The screen position at the point of contact is calculated form a measurement of the time interval between the transmission of each wave and its reflection to the emitter.

➜ 1.5.1(F) Tablets

- These are used along the display device to input / update graphical data directly to the computer system, e.g., for free hand sketching, tracing a map, engineering drawing, electrical circuits, etc. The tablet is the most common locator device. It is a flat surface on which the user draws with cursor or stylus. Thus, one can interact with the device as if writing with the pen on paper. Usually the proximity of the stylus to the tablet surface is also sensed. When used in conjunction with a CRT display, feedback from the CRT face is provided by means of a small tracking symbol called a cursor, which follows the movement of the stylus on the tablet surface. Tablet provides two-dimensional coordinate information. The values returned are in tablet coordinates. These coordinates are converted into user or world coordinates by a software.
- The feedback of one's action is available on the display by the cross hairs or a square whose movement is locked with the motion of the stylus, when pressed on the tablet. The resolution of such a device can be as high as 0.1%. A number of mechanisms have been developed for translating the position and direction of cursor movement on the tablet into a signal which can be used by the computer.

1.5.2 Data Generating Devices

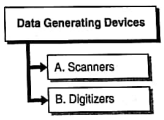


Fig. C1.4 : Data Generating Devices

➜ 1.5.2(A) Scanners

- Now a day scanners are heavily used as input devices. A scanner is a peripheral for scanning documents, i.e. converting a paper document to a digital image. We can scan drawings, graphs, pictures, text or photos and store it in computer which acts as input for further processing i.e. Instead of typing some document by using keyboard we can directly scan that document also. So scanner is also one of the input device only.
- The scanners use the optical scanning mechanism to scan the information. The scanner records the gradation of gray scales or color and stores them in the array. Scanner stores the image in specific file format. Once the image is scanned, it can be processed or we can apply transformations such as rotate, scale etc. to that image.
- Fig. 1.5.4 shows a typical scanner.

Fig. 1.5.4

☞ **Types of Scanner**

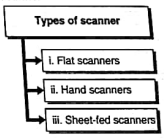


Fig. C1.5 : Types of scanner

- There are generally three types of scanner:

➜ **(i) Flat scanners :** They allow us to scan a document by placing it flat against a glass panel. This is the most common type of scanner.

➜ **(ii) Hand scanners :** They are smaller in size. These scanners must be moved manually in successive sections over the document in order to scan the whole document.

➜ **(iii) Sheet-fed scanners :** feed the document through a lighted slot in order to scan them, similar to fax machines.

☞ **Operating principle for scanner**

- The operating principle for a scanner is as follows :
  - The scanner moves over the document line by line.
  - Each line is broken down into "basic dots" which correspond to pixels.

- A captor analyzes the color of each pixel.
- The color of each pixel is broken down into 3 components (red, green, blue).
- Each color component is measured and represented by a value. For 8-bit quantification, each component will have a value between 0 and 225 inclusive.

➜ 1.5.2(B) Digitizers

- Digitizer is a common device for drawing, painting or interactively selecting co-ordinate positions on an object. See Fig. 1.5.5. These devices can be used to input co-ordinate values in either a two-dimensional or a 3-dimentional space.

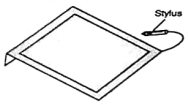


Fig. 1.5.5

- Typically a digitizer is used to scan over a drawing or object and to input a set of discrete co-ordinate positions, which can be joined with straight-line segments to approximate the curve or surface shapes. One type of digitizer is the graphics tablet which is used to input two-dimensional co-ordinates by activating a hand cursor or stylus at selected positions on a flat surface. Stylus is a pencil-shaped device that is pointed at positions on the tablet.

1.5.3 Display Devices

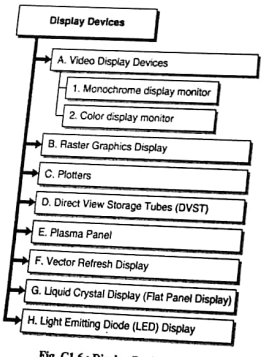


Fig. C1.6 : Display Devices

- The basic question in computer graphics is how to locate and display points and line segments. There are several hardware devices which may be used to display images.

➜ 1.5.3(A) Video Display Devices

- There are two types of video monitors :

1. Monochrome display monitor
2. Color display monitor

➜ 1. Monochrome display monitor

- Let us first see the Monochrome display monitor (Black and white).
- It mainly consist a Cathode Ray Tube (CRT) along with related circuits. The CRT is a vacuum glass tube with the display screen at one end and connectors to the control circuits at other end. See Fig. 1.5.6. Phosphor is coated on the inside of the display screen which emits lights for a period of time when hit by a beam of electrons.
- The color of light and time period varies from one type of phosphor to another. The light given off by the phosphor during exposure to the electron beam is known as *fluorescence*, the continuing glow given off after the beam is removed is known as **phosphorescence** and the duration of phosphorescence is known as the **phosphors persistence.**

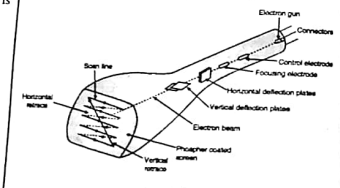


Fig. 1.5.6

- Opposite to phosphor-coated screen i.e. on other end there is an electron gun which is heated to send out electrons. The electrons are regulated by the control electrode and forced by the focusing electrode into a narrow beam striking the phosphor coating at small spots.
- When this electron beam passes through the horizontal and vertical deflection plates, it is bent or deflected by the electric field between the plates. The horizontal plate controls the beam to scan from left to right and retrace from right to left.
- The vertical plates control the beam to go from the first scan line at the top to the last scans line at the bottom

and retrace from the bottom back to the top. These actions are synchronized by the control circuits so that the electron beam strikes each and every pixel position in a scan line by scan line fashion.

- The intensity of the light emitted by the phosphor coating is a function of the intensity of the electron beam. The control circuits shut off the electron beam during horizontal and vertical retraces. The intensity of the beam at a particular pixel position is determined by the intensity value of the corresponding pixel in the image being displayed.

➜ 2. Color Display Monitor

- In color display there are 3 electron guns instead of one with one electron gun for each primary color. See Fig. 1.5.7. The phosphor coating on the inside of the display screen consists of dot patterns of three different types of phosphors. These phosphors are capable of emitting Red, Green and Blue light. The distance between the centers of the dot patterns is called the pitch of the color CRT. A thin metal screen called a shadow mask is placed between the coating and electron guns.
- The tiny holes on the shadow mask constrain each electron beam to hit its corresponding phosphor dots. When viewed from a certain distance, light emitted by the 3 types of phosphors blends together to give us a broad range of colors.

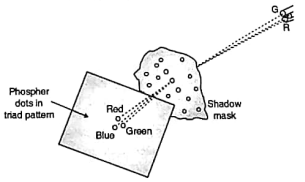


Fig. 1.5.7

➜ 1.5.3(B) Raster Graphics Display

- If we want to get a good resolution say 1000 lines on both x and y directions, then the screen would have resolution 1000 × 1000 i.e. 1 million pixels. And each pixel needs at least one bit of intensity information, light or dark and further bits are needed if shades of different colors are desired.
- Thus, if we want to store the information for each pixel in computers memory a lot of memory may be required. This is in fact what is done in *raster graphics display*. A portion of system memory which is used to hold the pixels is called as *frame buffer*.

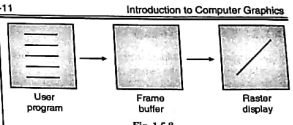


Fig. 1.5.8

- In raster display, which is shown in Fig. 1.5.8, the frame buffer may be examined to determine what is currently being displayed. Surfaces or lines may be displayed on raster display devices. The raster terminal can also display color images.
- One of the problems with the raster display is the time which may be required to alter every pixel whenever the image is changed. Another disadvantage is the cost of the required memory. So, in the past, the cost of memory made the raster display less promising.
- So one approach was to let the display medium remember the image, instead of computer Memory. This is what *plotters* do.

➜ 1.5.3(C) Plotters

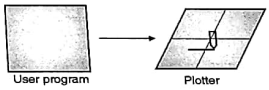


Fig. 1.5.9

- A pen is lowered onto the paper and moved under the direction of a vector generation algorithm. (See Fig. 1.5.9)
- Once the line is drawn, the ink on the paper remembers it and computer need not consider it further.
- But once a line is drawn, it cannot be easily removed.
- If we change the image on plotter by removing a line, we have to get new paper and redraw the whole picture.
- For this reason plotters are not best devices for interactive graphics.

➜ 1.5.3(D) Direct View Storage Tubes (DVST)

- CRT is a low cost and most widely used device. The terminals or devices use special CRT called *direct view storage tubes (DVST)* which behave much similar to plotter. An electron beam is directed at the surface of the screen. The position of the beam is controlled by electric or magnetic fields within the tube. Once the screen phosphors of this special tube by the electron beam, they stay lit.
- Similar to plotter one cannot alter the DVST image except by erasing the entire screen and drawing it again. This can be faster than plotter but still not efficient and is a time consuming process.

➜ 1.5.3(E) Plasma Panel

- A display device which stores the image but allows selective erasing is the *plasma panel.*
- It contains a neon gas at a low pressure sandwiched between horizontal and vertical grids of fine wires. (See Fig. 1.5.10) A large voltage difference between a horizontal and vertical wire will cause the gas to glow as it does in a neon street sign.
- A lower voltage will not start a glow but will maintain a glow once started. Normally the wires have this low voltage between them.
- To set a pixel, the voltage is increased momentarily on the wires that intersect the desired point. To make pixel off, the voltage on the corresponding wires is reduced until the glow cannot be maintained.
- Plasma panels are very durable and are often used for military applications.

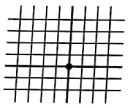


Fig. 1.5.10

➜ 1.5.3(F) Vector Refresh Display

- Another approach to the display of graphical information is the *vector refresh display.* Sometimes it is also called as *Random Scan Systems.*
- It also stores the image in computer's memory but it tries to be more efficient than raster display.
- To specify the line segment we need co-ordinates of end points. But the raster frame buffer stores not only the end points but also all pixels in between. The vector refresh display stores only the commands necessary for drawing the line segment.
- In vector refresh display input is saved, instead of output. These commands are saved in Display files.

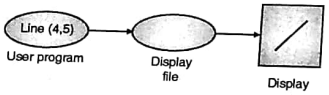


Fig. 1.5.11

- The commands in display files are examined and lines are drawn by appropriate algorithm (See Fig. 1.5.11). Vector refresh display allows alteration of the image.
- The disadvantage is images are composed of line segments not surfaces or pictures and a complex display may flicker because it will take long time to analyze and draw it.
- The concept of display file is very useful. It provides an interface between the image specification process and image display process. It also defines compact description of image, which may be used for later display.
- When the display file idea is applied for the devices other than refresh display then that file is called as pseudo display file or metafiles.

Syllabus Topic : Flat Panel Displays like LCD and LED

➜ 1.5.3(G) Liquid Crystal Display (Flat Panel Display)

- We probably use items containing an **LCD (Liquid Crystal Display)** every day. They are all around us, namely, in laptop computers, digital clocks and watches, microwave ovensand many other electronic devices. LCDs are common because they offer some real advantages over other display technologies. They are thinner and lighter and draw much less power than Cathode Ray Tube (CRTs). We get the definition of LCD from the name "Liquid Crystal" itself. The name "liquid crystal" sounds like a contradiction. We think of a crystal as a solid material, and a liquid is obviously different.
- We learned in school that there are three common states of matter: solid, liquid or gaseous. Solids act the way they do because their molecules always maintain their orientation and stay in the same position with respect to one another. The molecules in liquids are just the opposite: They can change their orientation and move anywhere in the liquid. But there are some substances that can exist in an odd state that is sort of like a liquid and sort of like a solid. When they are in this state, their molecules tend to maintain their orientation, like the molecules in a solid, but also move around to different positions, like the molecules in a liquid. This means that liquid crystals are neither a solid nor a liquid.
- Liquid crystals are closer to a liquid state than a solid. It takes a fair amount of heat to change a suitable substance from a solid into a liquid crystal, and it only takes a little more heat to turn that same liquid crystal into a real liquid.
- LCD panels are fairly simple to understand. The signal comes in and, as with a CRT, the signal from the video controller is decoded and understood by a display controller on the monitor itself. The controller has two things to control - the electrics of the pixels and the light source. See Fig. 1.5.12.

Fig. 1.5.12

- The actual image on a TFT is made up of a matrix of pixels. Unlike with CRTs, there's no complex equation of dot pitch and image area to try and calculate - the native resolution of the monitor is simply the number of pixels contained in the matrix. If it's a 17" monitor, chances are there are 1280 pixels in the matrix horizontally, and 1024 vertically.
- Each pixel is made up of three sub-pixels, which have red, green and blue filters in front of them, just as each pixel on a CRT has RGB phosphors. The sub-pixels are made up of a group of liquid crystal molecules. These molecules are suspended between transparent electrodes and are mashed between two polarizing filters.
- The two filters are exact opposites of each other. As the light from the light source behind the first filter comes in, the filter effectively whites it out - which means that if it was to pass through the liquid crystals with no interaction, the filter on the other side would polarize it back to black, leaving no color being emitted. In fact, alternate current - leaving the crystals 'dead in the water' - is how black is created on a panel.
- However, if the electrodes apply current to the liquid crystals they twist and change the way that the light is passed through, altering its polarization and this then results in the correct color coming out of the second polarizing filter and being displayed to the user.
- The backlight itself is a cold cathode. Depending on how expensive the display is, there will be either a single cathode at the top, or one at the top and one at the bottom, or two at the top and two at the bottom for optimum brightness and clarity. These cathodes are diffused through a layer of plastic and then through multiple layers ofdiffusing material of the kind you might find on a flashgun diffuser for photography.

➜ 1.5.3(H) Light Emitting Diode (LED) Display

- A LED display, or light emitting diode display, is a flat panel display that uses light emitting diodes as the video display. Basically LED monitors are the LCD monitors with a LED backlight to power up the LCD panel. It means that LEDs are placed behind or around the LCD panel to enhance the luminosity and video definition of the monitor screen.
- In LED monitors all the concepts are same except the backlight. In LCD monitors Cold Cathode Fluorescent Light (CCFL) is used for backlight, whereas LED's are used as backlight in LED monitors.
- There are three different types of LED monitors available based on the manner how the diodes are arranged in the monitor. These are – Direct LEDs, Edge LEDs and RGB LEDs. Both Edge and Direct LED display monitors use white diodes that are used to illuminate the LCD panel to produce the improved picture quality. Fig. 1.5.13 show Direct LED monitor where white diodes are placed all over the panel to produce higher quality image. Direct LEDs are also called as full array LED backlighting. Generally Direct LEDs are used in the HD TV.

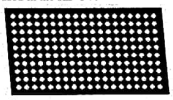


Fig. 1.5.13

- In case of Edge LED displays, LEDs are used only on the borders of the LCD panel. Refer Fig.1.5.14. Edge LEDs is mainly used in the generation of computer screens.

Fig. 1.5.14

- Out of these three type of LEDs,RGB LED display is better as it uses red, green and blue diodes to generate all the images with amazing contrast ratio.

Difference between LED and LCD

Sr. No.	LED	LCD
1.	Contrast and Black level are better than LCD	Contrast and Black level are Poor than LED
2.	Color accuracy of RGB LEDs are better than LCD	Color accuracy of Direct LED and Edge LED are almost same as that of LCD
3.	LED displays consume less power as compared to LCD	LCD displays consume more power as compared to LED
4.	LEDs are used for backlight	Cold Cathode Fluorescent Light (CCFL) is used for backlight

Sr. No.	LED	LCD
5.	LED displays are more LCD monitors are not slimmer and brighter slimmer and brighter than than LCD.	LED
6.	More costlier than LCD.	Cheaper than LED

1.6 Primitive Operations

- Regardless of the differences in display devices, most graphics systems offer a similar set of graphics primitive commands. Now we will see the primitive commands to draw line segment. To draw any line segment we need 2 points. Let's say we have drawn a line AB. See Fig. 1.6.1.

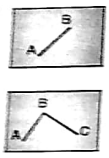


Fig. 1.6.1

- If we want to join another line BC to this line segment AB then it will become ABC because we are adding BC at end of AB line segment. i.e. often the segments drawn are connected end to end.
- It means the final point of first segment will be the starting point for next segment. In the example which is shown in Fig. 1.6.2, segment AB is drawn first then BC then CD and so on i.e. Final point of segment AB, which is B, is acting as starting point for next segment BC.
- Usually to draw single line segment we need two points. similarly to draw two lines we need four points. But we can draw two line segments by just three points only because here one point is common.

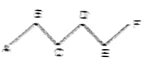


Fig. 1.6.2

- In this case then command becomes : draw a line from the current position to the specified point i.e. say command as *line-abs (x, y)*.
- Here x and y are actual final co-ordinates. It means we are going to draw a line from current position to point (x, y). See Fig. 1.6.3. Such a commands are called as *Absolute line command*.

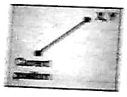


Fig. 1.6.3

- There is another type also, which is called as *Relative line command*.
- In relative line command we indicate how far to move in x and y direction from the current position to reach the final point. See Fig. 1.6.4.
- The command used will be line-Rel (Dx, Dy)
- where Dx and Dy are not actual co-ordinates, but they indicate relative distance from current x and y point to final point.

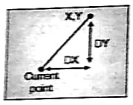


Fig. 1.6.4

- But it may happen that we want to draw two disconnected segments as shown in Fig. 1.6.5.
- This can be accomplished by the same mechanism if we draw these two segments as connected by a middle segment BC which happens to be invisible as shown in Fig. 1.6.6.

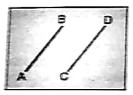


Fig. 1.6.5

- So we need a command to move the current position without drawing line between BC or by drawing invisible line BC. See Fig. 1.6.6. For this purpose we have to use command, *Move-abs (x, y)*

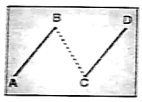


Fig. 1.6.6

- Where (x, y) indicates new position for the current point. Similarly we are having another command.
- *Move-Rel (Dx, Dy)* Where Dx and Dy indicates relative position.

1.7 Display File and its Structure

Q. Explain Display File Structure. (5 Marks)
Q. What is a Display file structure ? (5 Marks)

- We are aware that in the raster graphics displays pictures are stored in frame buffer which holds information about all pixels whereas in vector refresh displays we use commands to store the picture which we want to display. So in vector refresh displays we are storing commands i.e. input instead of output. The file in which we are storing these commands to draw picture is called **Display file.**
- In this display file we are going to store command by proper way. Each command in display file, we are breaking it into two parts : *opcode* and *operand.*
- Opcode is nothing but a command, which will be like *Move* or *Line* etc. and operand will be parameters.

E.g. *move (10, 10)*

- Here we will break this command in opcode and operand as,

Opcode	Operand
Move	10, 10

- Uptill now we have seen only two primitive command *Move* and *Line*. So initialize opcode 1 for *Move* command and opcode 2 for *Line* command.

Opcode	Command
1	*Move*
2	*Line*

- As operand is having both x and y parameters we have to separate them. So we will store the x co-ordinate in x operand and y co-ordinate in y operand.
  So now our command *[move (10, 10)]* will become.

Opcode (op)	x-op	y-op
1	10	10

- This is what about one command. But in display file there will be many commands. So we will use arrays to store opcodes and operands.

Opcode(op)array	x-op array	y-op array

- Now we can access a particular command from this display file very easily.
  e.g.

Opcode (op)array	x-op array	y-op array
1	10	10

- Suppose we want to access second command then we will just say
  op [2], x-op [2], y-op [2].
- Now let us take an example :
  Suppose we want to draw rectangle ABCD. Then we will perform following steps :

Step I : Move to point A which may be (x_1, y_1). See Fig. 1.7.1.
So our command will be
Move (x_1, y_1)
We will represent this command in opcode and operand form by $(1, x_1, y_1)$

Fig. 1.7.1

where 1 means *Move* command
x_1 indicates x co-ordinate of point A
y_1 indicates y co-ordinate of point A

Step II : Draw line from A to B
So command will be *line* (x_2, y_2)
where (x_2, y_2) are the co-ordinates of point B.(See Fig. 1.7.2)
Again we will represent this command in the form of opcode and operand as $(2, x_2, y_2)$
where 2 stands for line command.

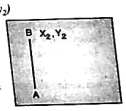


Fig. 1.7.2

Step III : Draw line from B to C
So command will be *line*(x_3, y_3)
where (x_3, y_3) will be the co-ordinates of point C.
(See Fig. 1.7.3)
Now we will represent this command as $(2, x_3, y_3)$
where 2 stands for line command.

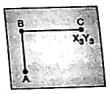


Fig. 1.7.3

Step IV : Draw line from C to D
So command will be
line (x_4, y_4)
where (x_4, y_4) will be co-ordinates of point D. (See Fig. 1.7.4)
And the command will be $(2, x_4, y_4)$
where 2 stands for line command.

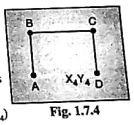


Fig. 1.7.4

Step V : Draw line from D to A
So command will be *line* (x_1, y_1)
where (x_1, y_1) represents point A.
i.e. we have to close the rectangle.
See Fig. 1.7.5. As we are drawing line so we have represent this command as $(2, x_1, y_1)$
where 2 stands for line command.

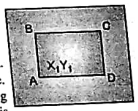


Fig. 1.7.5

Now our display file for this rectangle ABCD will look like.

Opcode	x-op	y-op
1	x_1	y_1
2	x_2	y_2
2	x_3	y_3
2	x_4	y_4
2	x_1	y_1

Now this display file is containing what to display on screen.

1.7.1 Display File Interpreter

Q. Explain the need for display file interpreter. (5 Marks)

- The display file will contain the information necessary to construct the picture. Saving the instructions usually takes much less storage than saving the picture itself. These instructions are nothing but a sample programs.
- Each instruction like move or line indicates some action for display device. So we need something to convert these instructions into actual images. That conversion is done by display file interpreter. It interprets each command from display file into some action on display. (See Fig. 1.7.6)
- Interpreter may be a part of machine which executes these instructions. The result of execution is image. In some graphics systems there is separate computer which handles this job which is called **display processor**.

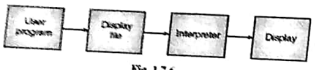


Fig. 1.7.6

- Display file interpreter serves as an interface between our graphics program and the display device.

Syllabus Topic : Raster Scan and Random Scan Displays

1.8 Raster and Random Scan

→ **(May 2014)**

Q. Write short note on : Raster techniques.
MU - May 2014, 10 Marks

Q. Explain in brief Raster-scan display System. (10 Marks)

- There are two different ways of scanning video displays. These are called as raster and random scans. In raster scan the electron beam follows a fixed path as shown in Fig. 1.8.1.
- The electron beam starts at top left corner of the screen and moves horizontally to the right. This defines a scan line. During the scan the intensity of the beam is modulated according to the pattern of the desired image along the line.
- At the right corner of the screen, the beam becomes off and moved back to the left edge of the screen at the starting point on next line. This is shown as dotted line, which is called as horizontal retrace.
- This way of scanning is continued till the bottom right corner is reached. At this point, one scan is completed. The beam is then repositioned at the top left corner of the screen for starting another scan. This movement of beam from bottom right corner to top left corner is called as vertical retrace.

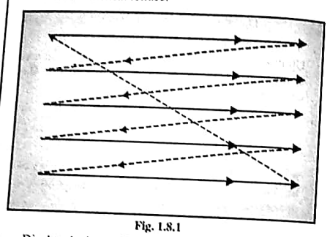


Fig. 1.8.1

- Display devices with random scan operate by directing the electron beam to only those parts of the screen where the picture is to be drawn. This is also called as vector graphics. This vector graphics depends on the ability of hardware to generate line vectors. Creation of diagrams using vector graphics becomes easier, so can be used in engineering and scientific drawings, whereas the raster graphics can be used in animation.
- In random scan pen plotters and direct storage view tubes (DVST) devices are used, whereas in raster scan cathode ray tubes (CRT) are used. Since the cost of devices used for random scan is much higher than the cost of devices for raster scan, people are using raster scan devices heavily.

Advantages of Random scan

- Very high resolution, limited only by monitor.
- Easy animation, just draw at different positions.
- Requires less memory.

Disadvantages of Random Scan

- Requires intelligent electron beam, i.e. processor controlled.
- Limited screen density before have flicker, can't draw a complex image.
- Limited color capability.

- For random scan the host system generates a display file of graphics commands which is executed by the display processor. The display processor then just reads the frame buffer and turns the electron beams on/off.

1.8.1 Differentiation of Random and Raster Scan

→ **(Dec. 2014, Dec. 2015, May 2017)**

Q. Differentiate between Random scan and Raster scan display. MU - Dec. 2014, Dec. 2015, 5 Marks

Q. Compare random scan and raster scan technique. MU - May 2017, 5 Marks

Q. Clearly differentiate between random scan and raster scan system. (5 Marks)

Sr. No.	Random Scan	Raster Scan
1.	Random scan operate by directing the electron beam to only those parts of the screen where the picture is to be drawn.	The electron beam starts at top left corner of the screen and moves horizontally to the right.
2.	Creation of diagrams using random scan becomes easier, so can be used in engineering and scientific drawings.	Raster graphics can be used in animation.
3.	Pen plotters and direct storage view tubes (DVST) devices are used.	Cathode ray tubes (CRT) are used.
4.	The cost of devices used for random scan is much higher.	The cost of devices used for raster scan is much cheaper.
5.	Requires less memory.	Requires more memory.
6.	Requires intelligent electron beam, i.e. processor controlled.	No such requirement.

1.9 Introduction to OpenGL

→ **(May 2015, Dec. 2015)**

Q. Write short note on : OpenGL.
MU - May 2015, 5 Marks, Dec. 2015, 10 Marks

- **OpenGL** (**Open** Graphics **L**ibrary) is a standard specification defining a cross-language, cross platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL's main purpose is to render two and three dimensional objects into a frame buffer.
- OpenGL is a low level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. In its basic operation, OpenGL accepts primitives such as points, lines and polygons, and converts them into pixels via a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

1.9.1 OpenGL Operation

- The Fig.1.9.1 Shows the block diagram of OpenGL data processing method. The commands enter from the left and proceed through a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during various processing stages.

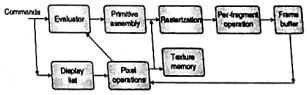


Fig. 1.9.1

The processing stages in basic OpenGL operation are as follows:

- **Display list :** Rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.
- **Evaluator :** The evaluator stage of processing provides an efficient way to approximate curve and surface geometry by evaluating polynomial commands of input values.
- **Primitive assembly :** OpenGL processes geometric primitive points, line segments, and polygonsall of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for rasterization.
- **Rasterization :**The rasterization stage produces a series of frame-buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced is fed into the last stage, per-fragment operations.
- **Per-fragment operations :** These are the final operations performed on the data before it is stored as pixels in the framebuffer.
- Per-fragment operations include conditional updates to the framebuffer based on incoming and previously stored z values (for z buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.
- Data can be input in the form of pixels rather than vertices. Data in the form of pixels, such as might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The

pixel data is either stored as texture memory, for use in the rasterization stage; or Rasterized, with the resulting fragments merged into the framebuffer just as if they were generated from geometric data.

1.9.2 Functions and Features of OpenGL

Functions

There are many functions provided by OpenGL for graphics system. Some of the important graphics functions of OpenGL are as follows :

- **Display list :** The contents of display list are preprocessed and executed more efficiently than the same set of OpenGL commands executed in immediate mode.
- **Feedback :** This is the mode where OpenGL will return the processed geometric information to the applications as compared to rendering them into frame buffer.
- **Alpha Bending :** It is used to create transparent objects.
- **Pixel Operations :** There are different operations carried on pixels such as storing, mapping, zooming, transformations, etc.
- **Texture mapping :** It is used to create realistic effects in images. In this process we are applying different graphics primitives on images.
- **Color index mode :** Instead of storing the values of RGB components color buffer stores color indices.
- **Polynomial evaluators :** It supports NURBS (Non Uniform Rational B-Spline)

Features

In addition to above functionality OpenGL offers following some of the important features also.

- **Scaling and Rotation :** The transformations like scaling and rotation can be achieved very easily without any problem.
- **Performance :** Higher level performance is achieved when OpenGL is used on an accelerated device.
- **Transparency :** We can make some portion of the graphics as a transparent area with the help of OpenGL.
- **Alpha Blending:** It allows graphics to be drawn semi transparently.
- **Rendering in 3D:** OpenGL offers ability to render 3D graphics. By using OpenGL we can draw the objects which are away from the user as smaller as compared to the objects which are near to viewer.
- **Primitive operations:** OpenGL provides basic drawing primitive operations such as point, line, polygon. etc.
- **Hardware:** Whenever the speed of operation needs to be increased, OpenGL use hardware.

1.9.3 OpenGL Basic Drawing Primitives

→ (Dec. 2014, Dec. 2016)

Q. Write short note : Open GL basic primitives. **MU - Dec. 2014. 10 Marks**

Q. Explain OpenGL basic primitives **MU - Dec. 2016. 5 Marks**

- OpenGL supports various basic primitive types such as point, line and polygons. Sequence of vertices is used to specify these primitives. There are ten different primitive types. OpenGL interprets each primitive using following guidelines:
- **GL_POINTS :** used to render a point.
- **GL_LINES :** used to draw unconnected line segments. Each group of two vertices are drawn by OPENGL.
- **GL_LINE_STRIP :** used to draw a sequence of connected line segments i.e. joining the first and second point then second and third and so on.
- **GL_LINE_LOOP :** used to close the line strip. Usually it is used to join the first and last point.
- **GL_TRIANGLES :** used to draw the individual triangles. Triangle is formed from the three vertices.
- **GL_TRIANGLE_STRIP :** used to draw a sequence of triangles that share edges. Usually this is used when we need to draw a triangle having first, second and third edge followed by a common edge as second edge with another triangle i.e. second, third and fourth edge.
- **GL_TRIANGLE_FAN :** Whenever we need to share edges as well as vertices we are making use of this primitive. One of the common vertex is shared by all the triangles.
- **GL_QUADS :** used to draw a separate convex quadrilaterals. Whenever there are four vertices a convex quad is formed.
- **GL_QUAD_STRIP :** used to draw a sequence of quadrilaterals which shares their edges.
- **GL_POLYGON :** used to draw a solid convex polygon. To use this primitive the number of vertices and edges must be at least three.

1.9.4 OpenGL Interaction with Mouse and Keyboard

There are three libraries of utility functions in OpenGL which are Basic GL, GLUT and GLU. Out of these GLUT (GL Utility Toolkit) provides a generic interface to the window system.

Mouse event

- There are three types of mouse events which are recognized by GLUT. Whenever a mouse button is pressed or released, each time a mouse click event is generated. The screen coordinates of the cursor along with status of the mouse button i.e. pressed or released, are passed to the callback function which is registered using glutMouseFunc().

- When the mouse is moved with button in pressed state from one position to another, a motion event is generated. A passive motion event is generated when the button is not pressed and the mouse is moved. The new coordinates of the cursor are generated by the glutMotionFunc() and glutPassiveMotionFunc().

Keyboard Event

- There are two types of keyboard events. When a key on keyboard is pressed a normal keyboard event is generated. For the release of the key there is no event generated. The screen coordinate of the cursor and the ASCII character code for the key pressed are passed to the callback function registered using glutKeyboardFunc(). All the function keys such as F1 to F12 generates special key event. For such function keys an enumerated value is assigned to the callback function registered using glutSpecialFunc.

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 What are the different applications of Graphics system ?

Ans. : The answer for such a question should includes following points.

- What is computer graphics
- How it is useful in different areas
- How it can be used in our daily life
- Student must write at least 3-4 lines for each of the above mentioned point.

Q. 2 What are the characteristics of display adapter ?

Ans. : The answer for this question should include the detail explanation of following characteristics :

- Resolution
- Color depth
- Refresh rate
- Acceleration

Q. 3 Write a short note on any output device.

Ans. : Here we have to explain one of the following device in detail

- Monitor (both monochrome and color)
- Raster graphics display
- Plotters
- DVST
- Plasma panels
- Vector refresh display

While explaining any device mentioned above we have to draw a suitable diagram and explain the working of that device. Also don't forget to mention the advantages and disadvantages of this device. If possible mention the application of that device also.

Q. 4 Explain display file and its structure.

Ans. : Here we have to explain

- What is display file
- Why it is required
- What is its structure
- How the display file interpreter comes in picture by using block diagram.
- Explain the display file by taking any simple example.
- Draw the diagrams wherever needed.

Q. 5 Differentiate between raster and random scan.

Ans. : Here we have to explain

- What is raster scan and random scan
- Draw the diagram for raster scan.
- Different devices used for raster and random scan.

Q. 6 What is frame buffer ? How long would it take to load a 1280 by 1024 frame buffer with 12 bits per pixel if transfer rate is 1Mbps ?

Ans. : Picture definition is stored in a memory area called the refresh buffer or frame buffer. This memory area holds the set of intensity values for all the screen points. Because eight bits constitute a byte, frame-buffer sizes of the systems are as follows :

1280 × 1024 × 12 bits = 15728640 bits = 15.73 Mbits. Since the transfer rate is1 Mbps and we need to transfer 15.73 Mbits data we need approximately 16 sec.

Q. 7 Consider a raster system with resolution of 1280 by 1024. What size of frame buffer is needed for given system to store 24 bits per pixel ?

Ans. : The frame buffer size is calculated as

1280 × 1024 × 24 bits /8=3932 kB.

Since 8 bit constitutes a byte, the frame buffer size will be 3932 k Bytes.

Q. 8 Write a short note on Normalized device co-ordinates.

Ans. : Answer for this question should include following points :

- Size of screen in NDC with diagram

- Explanation of conversion formulas from NDC to screen co-ordinates and vice versa.
- Take an example to explain the working of NDC.
- Explain the limitations of NDC and different techniques to overcome them.
- Draw suitable diagrams wherever possible.

1.10 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Definition and Representative uses of Computer Graphics, Classification of Application Areas**

Q. What is computer graphics ? *(Refer Section 1.1)* **(3 Marks)**

Q. State various applications of Computer Graphics. *(Refer Section 1.1.1)* **(May 2016, 5 Marks)**

Q. What are different Applications of computer graphics ? (Refer Section 1.1.1) **(5 Marks)**

☞ **Syllabus Topic : Flat Panel Displays like LCD and LED**

Q. Explain Display File Structure. *(Refer Section 1.7)* **(5 Marks)**

Q. What is a Display file structure ? *(Refer Section 1.7)* **(5 Marks)**

Q. Explain the need for display file interpreter. *(Refer Section 1.7.1)* **(5 Marks)**

☞ **Syllabus Topic : Raster Scan and Random Scan Displays**

Q. Write short note on : Raster techniques. *(Refer Section 1.8)* **(May 2014, 10 Marks)**

Q. Explain in brief Raster-scan display System. *(Refer Section 1.8)* **(10 Marks)**

Q. Differentiate between Random scan and Raster scan display. *(Refer Section 1.8.1)* **(Dec. 2014, Dec. 2015, 5 Marks)**

Q. Compare random scan and raster scan technique. *(Refer Section 1.8.1)* **(May 2017, 5 Marks)**

Q. Clearly differentiate between random scan and raster scan system. *(Refer Section 1.8.1)* **(5 Marks)**

Q. Write short note on : OpenGL. *(Refer Section 1.9)* **(May 2015, 5 Marks, Dec. 2015, 10 Marks)**

Q. Write short note : Open GL basic primitives. *(Refer Section 1.9.3)* **(Dec. 2014, 10 Marks)**

Q. Explain OpenGL basic primitives *(Refer Section 1.9.3)* **(Dec. 2016, 5 Marks)**

□□□

CHAPTER 2

Module 2

Output Primitives

Section Nos.	Name of the Topic
2.1	Introduction
2.2	Line and Line Segments
2.3	Line Generation Algorithms
2.4	Thick Line Generation
2.5	Circle Generation Algorithms
2.6	Ellipse Generation
2.7	Arcs and Sectors
2.8	Aliasing and Antialiasing
2.9	Character Generation

Line, circle and ellipse are the basic elements in computer graphics. These are important enough that we devote this chapter to methods for drawing them. There are nine major sections in this chapter.

2.1 **Introduction** briefly tells the process of scan conversion and representation a point.

2.2 **Line and Line Segment** explains equation of line and intersection of lines.

2.3 **Line Generation Algorithms** explains different ways by which we can draw lines. In this section we have explained DDA and Bresenham line generation algorithms.

2.4 **Thick Line Generation** explains the procedure to generate thick lines.

2.5 **Circle Generation Algorithm** explains techniques by which we can draw circle. Here we have explained midpoint and Bresenham's circle generation algorithm.

2.6 **Ellipse Generation** explains the midpoint ellipse generation method.

2.7 **Arcs and Sector** tells how to use circle and line generation algorithms to draw arcs and sectors.

2.8 **Aliasing and Antialiasing Techniques** explains different drawbacks and the solution for those drawbacks, when we are displaying an image.

2.9 **Character Generation** explains different methods to generate characters by using computer graphics

Module 2

CHAPTER

2

Output Primitives

Section Nos.	Name of the Topic
2.1	Introduction
2.2	Line and Line Segments
2.3	Line Generation Algorithms
2.4	Thick Line Generation
2.5	Circle Generation Algorithms
2.6	Ellipse Generation
2.7	Arcs and Sectors
2.8	Aliasing and Antialiasing
2.9	Character Generation

Line, circle and ellipse are the basic elements in computer graphics. These are important enough that we devote this chapter to methods for drawing them. There are nine major sections in this chapter.

2.1 **Introduction** briefly tells the process of scan conversion and representation a point.

2.2 **Line and Line Segment** explains equation of line and intersection of lines.

2.3 **Line Generation Algorithms** explains different ways by which we can draw lines. In this section we have explained DDA and Bresenham line generation algorithms.

2.4 **Thick Line Generation** explains the procedure to generate thick lines.

2.5 **Circle Generation Algorithm** explains techniques by which we can draw circle. Here we have explained midpoint and Bresenham's circle generation algorithm.

2.6 **Ellipse Generation** explains the midpoint ellipse generation method.

2.7 **Arcs and Sector** tells how to use circle and line generation algorithms to draw arcs and sectors.

2.8 **Aliasing and Antialiasing Techniques** explains different drawbacks and the solution for those drawbacks, when we are displaying an image.

2.9 **Character Generation** explains different methods to generate characters by using computer graphics

Syllabus Topic : Definition of Scan Conversion, Rasterization, Rendering

2.1 Introduction

- In last chapter we have studied many display devices. To make use of these devices to display the objects like line, circle, polygons, etc., we need some special procedures. But whatever may be the procedure used, the system is generating the images by turning the pixels ON or OFF. The process in which the object is represented as the collection of discrete pixels is called **scan conversion.**
- Scan conversion of a point is nothing but turning ON a particular point or pixel. We know that pixel is a smallest element of the screen which can be controlled i.e. either On / Off. If we wish to display a point P(3.3, 5.5), then it means we want to illuminate this pixel. We can say that point P(x, y) is represented by an integer part of 'x' and integer part of 'y'. so we can use the command as

putpixel (int x, int y, Color);

- Normally we use right handed coordinate system, however in case of computer system, the system turns out to be left handed system.

2.2 Line and Line Segments

- We can specify a point by giving pair of numbers (x, y) where x represent horizontal distance and y represents vertical distance. To specify a line we need two such points.
- Every line is described by the equation such that if a point (x, y) satisfies the equations, then that point is on the line. If the two points used to specify a line are (x_1, y_1) and (x_2, y_2) then the equation of line will be,

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1} \quad \ldots(2.2.1)$$

- This equation says that the slope between any point on the line and point(x_1, y_1) is the same as the slope between (x_1, y_1) and (x_2, y_2). There are many equivalent forms for this equation.

Multiplying by denominators we get,

$$(y - y_1)(x_2 - x1) = (y_2 - y_1)(x - x_1) \quad \ldots(2.2.2)$$

by solving this for y,

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

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

i.e. $$y = \frac{y_2 - y_1}{x_2 - x_1} * (x - x_1) + y_1$$

$$\therefore y = \underbrace{\frac{y_2 - y_1}{x_2 - x_1}}_{m} x \underbrace{- \frac{y_2 - y_1}{x_2 - x_1} x_1 + y_1}_{b}$$

$$y = m \cdot x + b \quad \ldots\text{slope intercept form}$$

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

$$b = y_1 - m x_1$$

Slope m is the change in height divided by the change in width for any two points on the line. The **intercept b** is the height at which the line crosses the y-axis. We can get y intercept by having (0, b). Refer Fig. 2.2.1. A different form of line equation is called **general form.** Consider the Equation (2.2.2).

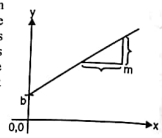


Fig. 2.2.1

$$(y_2 - y_1)(x - x_1) = (y - y_1)(x_2 - x_1)$$

$$(y_2 - y_1)x - (x_2 - x_1)y + x_2 y_1 - x_1 y_2 = 0$$

$$rx + sy + t = 0 \quad \ldots(2.2.3)$$

Here possible values of r, s, t are,

$r = y_2 - y_1 \qquad s = -(x_2 - x_1) \qquad t = x_2 y_1 - x_1 y_2$

If we compare Equation (2.2.3) with slope intercept form i.e. y = mx + b

$$rx + sy + t = 0$$

$$sy = -rx - t$$

$$y = \underbrace{-\frac{r}{s}}_{m} x \underbrace{- \frac{t}{s}}_{b}$$

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

2.2.1 Line Segments

- There is a difference between line and line segment. The lines can extend forever both forward and backward. But in graphics we would like to display only a piece of lines. Let's consider only those points on a line which lie between two end points P_1 and P_2. See Fig. 2.2.2. This is called line segment.

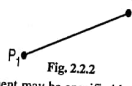


Fig. 2.2.2

- A line segment may be specified by its two end points. From these end points we can determine the equation of the line. If the end points are $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ and these yield equation y = mx + b then another point $P_3 = (x_3, y_3)$ lies on the segment if

$y_3 = mx_3 + b$

x_3 is in between x_1 and x_2.

y_3 is in between y_1 and y_2.

- There is one more useful form of the line equation called the **parametric form.** In this case x and y-values on the line are given in terms of a parameter u. Suppose we want the line segment between (x_1, y_1) and (x_2, y_2), then we need the x co-ordinate to move uniformly from x_1 to x_2. This may be expressed by the equation,

$$x = x_1 + (x_2 - x_1)u$$

where u = 0 to 1

- When u = 0, x = x_1. As u increases to 1, x moves uniformly to x_2. Similarly we must have the y co-ordinate moving from y_1 to y_2 at the same time as x changes.

$$y = y_1 + (y_2 - y_1)u$$

- Similarly to find the length of line segment we can make use of Pythagorean theorem. See Fig. 2.2.3.Suppose point P_1 is (x_1, y_1) and P_2 is (x_2, y_2). Now if we want to find length (L) of line segment $P_1 P_2$ then

$$L^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$$

$$L = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Similarly we can find midpoint of a line segment very easily.

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

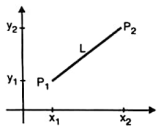


Fig. 2.2.3

Intersection of two lines

- We can determine a point where two lines can cross. By two lines crossing we mean they share some common point. That point will satisfy the equations for the both lines.
- Suppose there are two lines line 1 and line 2 and both lines are intersecting at point (x_i, y_i). Then we can represent line 1 by equation,

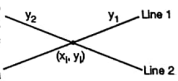


Fig. 2.2.4

$y = m_1 x + b_1$ and equation for line 2 as

$y = m_2 x + b_2$

- where m_1 and m_2 are slopes of line 1 and line 2 respectively. If point (x_i, y_i) is shared by both the lines then the point (x_i, y_i) must satisfy both the line equations.

$\therefore y_i = m_1 x_i + b_1$ and $y_i = m_2 x_i + b_2$

$\therefore$ Equating both over y_i.

$$m_1 x_i + b_1 = m_2 x_i + b_2$$

$$m_1 x_i - m_2 x_i = b_2 - b_1$$

$$x_i = \frac{b_2 - b_1}{m_1 - m_2} \quad \ldots(2.2.4)$$

Equating the value of x_i in any one line equation we will get y_i as,

$$y_i = m_1 x_i + b_1 = m_1\left(\frac{b_2 - b_1}{m_1 - m_2}\right) + b_1$$

$$= \frac{m_1 b_2 - m_1 b_1}{m_1 - m_2} + b_1$$

$$= \frac{m_1 b_2 - m_1 b_1 + m_1 b_1 - m_2 b_1}{m_1 - m_2}$$

$$y_i = \frac{m_1 b_2 - m_2 b_1}{m_1 - m_2}$$

$\therefore$ point $\left(\frac{b_2 - b_1}{m_1 - m_2}, \frac{m_1 b_2 - m_2 b_1}{m_1 - m_2}\right)$ is intersection point

Syllabus Topic : Scan Conversions of Point, Line, Circle and Ellipse - DDA Algorithm and Bresenham Algorithm for Line Drawing

2.3 Line Generation Algorithms

- The process of "Turning ON" the pixels for a line is called **line generation** or scan conversion of a line.
- When we want to draw a line that means we have to make the pixels on that line to ON condition i.e. we have to change the intensity of the pixels present on that line. For this task, we have two different algorithms.

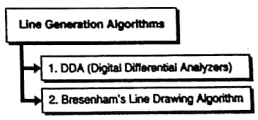


Fig. C2.1 : Line Generation Algorithms

- Both the algorithms are based on *increment method.* Let us see this increment method now.
- To draw a line we must know the two endpoints. Here we are selecting the pixels which lie near the line segment. See Fig. 2.3.1. We could try to turn on the pixels through which line is passing. But it will not be easy to find all such pixels and when we are using line drawing for big project or in animation where picture may change frequently, in that case it is very inefficient.

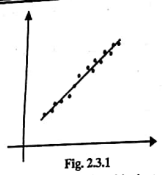
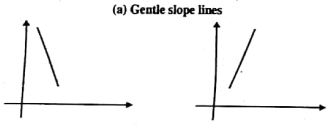


Fig. 2.3.1

- So the alternative way for this is to step along the columns of the pixels and for each column finalize which row is closest to the line. We could then turn ON the pixel in that row and column. We know how to find the row because we can place the x-value corresponding to the column into the line equation and solve for y. This will work for lines with slopes between –1 and 1 (i.e. lines which are closer to horizontal lines). Such lines are called as lines with *gentleslope*. But for *steep slope* case it will leave gaps. So we step along the rows and solve for the columns. Fig. 2.3.2 shows different lines.

(a) Gentle slope lines



(b) Steep slope lines

Fig. 2.3.2

- For *gentle slopes* ($-1 < m < 1$) there are more columns than rows. These are the line segments, where the length of the x component, $Dx = (x_b - x_a)$ is longer than the y component, $Dy = (y_b - y_a)$, i.e. $|Dx| > |Dy|$. For this case we step across the columns and solve for the rows. See Fig. 2.3.3. As there are more number of columns than rows and we are moving along columns we are getting more number of pixels i.e. on every column we are getting some pixel. So it will not leave gaps.
- For the *sharp slopes* where $|Dx| \leq |Dy|$, we step across the rows and solve for the columns. See Fig. 2.3.3.

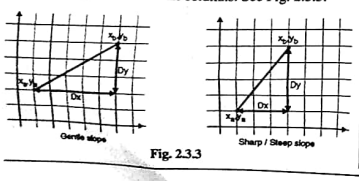


Fig. 2.3.3

- Consider the gentle slope case where we step across the columns. Each time we move from one column to the next, the value of x changes by 1. But if x always changes by exactly 1, then by what value y should change? Answer is, y will always change by exactly m (the slope). How? Let us say (x_i, y_i) is starting point. As we are considering gentle slope case means we are moving along columns i.e. x-axis, next point will be on column $(x_i + 1)$.
- If $(x_i, y_i) = (2, 3)$ then $x_i + 1 = 2 + 1$ and at that time what will be $(y_i + 1)$, that we have to find out. When we are saying that (x_i, y_i) is starting point means (x_i, y_i) satisfies equation of line i.e. $y_i = mx_i + b$. Now we know $(x_i + 1)$ and for that we have to find out $(y_i + 1)$ in such a way that $(x_i + 1, y_i + 1)$ will satisfy the equation of line i.e. $y_i + 1 = m(x_i + 1) + b$

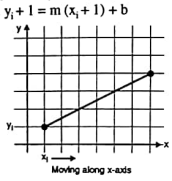


Fig. 2.3.4

$$\therefore (y_i + 1) - y_i = m(x_i + 1) + b - mx_i - b$$
$$= m[(x_i + 1) - x_i] = m(1) = m$$

- This means that as we step along the columns, we can find the new value of y for the line by just adding m or slope to the y position at the previous column. See Fig. 2.3.4.

➔ 2.3.1 DDA (Digital Differential Analyzers) Algorithm

➔ **(May 2015, May 2016, May 2017)**

Q. What are the disadvantages of DDA algorithm ?
MU - May 2015, May 2017, 5 Marks

Q. Specify the disadvantages of DDA algorithm.
MU - May 2016, 5 Marks

- Digital differential analyzer is based on incremental method. The slope intercept equation for a straight line is,

$$y = m \cdot x + b$$

Where m is slope and b is y-intercept. We can determine the value of m as,

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

$(y_2 - y_1)$ is nothing but change in y-values which is represented as Dy.

Similarly $(x_2 - x_1)$ is represented as Dx.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{Dy}{Dx}$$

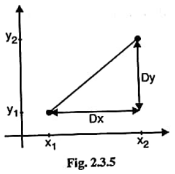


Fig. 2.3.5

Refer Fig. 2.3.5.

For lines with slope (m) is < 1 i.e. lines with positive gentle slopes, we are moving in x-direction by uniform steps of calculating the corresponding y-value by using.

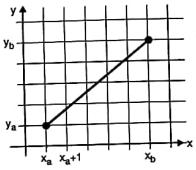


Fig. 2.3.6

$$\frac{Dy}{Dx} = m$$

$$Dy = m \cdot Dx$$

$$(y_a + 1) - (y_a) = m \cdot [(x_a + 1) - (x_a)]$$

But as we are moving in x-direction by 1 unit i.e. distance between two columns;

$(x_a + 1) - (x_a) = 1$. See Fig. 2.3.6.

$$\therefore (y_a + 1) - (y_a) = m(1)$$
$$\therefore y_a + 1 = m + y_a$$

i.e. $$y_{new} = \text{slope} + y_{old} \quad \ldots(2.3.1)$$

Similarly when the slope is > 1 i.e. lines with positive steep slopes, we are moving in y-direction by uniform steps and calculating the corresponding x-value. See Fig. 2.3.7.

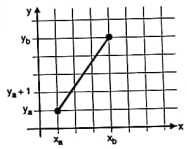


Fig. 2.3.7

$$\frac{Dy}{Dx} = m$$

$$\text{i.e. } Dx = \frac{Dy}{m}$$

$$(x_a + 1) - (x_a) = \frac{(y_a + 1) - (y_a)}{m}$$

But $[(y_a + 1) - (y_a)]$ is 1 unit i.e. distance between two rows.

$$\therefore (x_a + 1) - (x_a) = \frac{1}{m}$$

$$\therefore x_a + 1 = \frac{1}{m} + x_a$$

i.e. $$x_{new} = \frac{1}{\text{slope}} + x_{old} \quad \ldots(2.3.2)$$

When the slope (m) is = 1 then as x changes, values of y also changes. See Fig. 2.3.8.

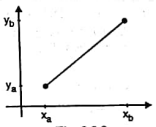


Fig. 2.3.8

i.e. $\frac{Dy}{Dx} = m$

but $m = 1$

$\therefore Dy = Dx$

$\therefore$ $y_{new} = y_{old} + 1$ and $x_{new} = x_{old} + 1$

- Equation (2.3.1) and (2.3.2) are based on assumption that lines are to be processed from left end point to right side i.e. $x_{start} < x_{end}$. If this process is reversed i.e. $x_{start} > x_{end}$ then $y_{new} = y_{old} - m$ and $x_{new} = x_{old} - \frac{1}{m}$.

Since slope (m) can be any real value (floating point number) between 0 to 1, the calculated y-value for Gentle slope or calculated x-value for steep slope must be rounded to the nearest integer, because display device is having co-ordinates as integer only. See Fig. 2.3.9.

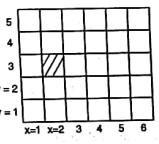


Fig. 2.3.9

- We can represent integer point as (2, 3) but we cannot represent any fractional point as (2.3, 3.1). So we need a separate function which will round that float value to integer. For this generally ceiling and flooring functions are used.
- **Ceil :** This is a function which returns smallest integer which is greater than or equal to its argument. Suppose the value is 8.6 and if we are passing this value to ceil function then that function will return 9.
- **Floor :** This is a function which returns largest integer which is less than or equal to its argument. If the value is 8.2 and if we are passing this value to floor function then it will return 8.

There is no doubt that because of this floor and ceil functions the point is diverting from actual location.

☛ Steps for DDA algorithm

1) Accept two end points: (x_a, y_a) and (x_b, y_b)
2) Find out horizontal and vertical difference between end points.
   $Dx = x_b - x_a$ and $Dy = y_b - y_a$
3) Difference between greater magnitude determines the value of parameter steps.
4) Determine the offset needed at each step i.e. to generate the next pixel loop through this step times.
5) Before displaying every point round off the point to its nearest integer value.

6) If $|Dx| > |Dy|$ and $x_a < x_b$. Then the values of the increments in the x and y-directions are 1 and m respectively. See Fig. 2.3.10.

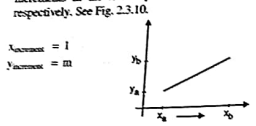


Fig. 2.3.10

7) If $|Dx| > |Dy|$ and $x_a > x_b$. Then the decrements –1 and –m are used to generate new point. See Fig. 2.3.11.

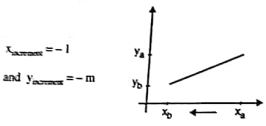


Fig. 2.3.11

8) Similarly if $|Dx| < |Dy|$ and $y_a < y_b$

Then the values of increments in x and y-direction are 1/m and 1 respectively.

Refer Fig. 2.3.12.

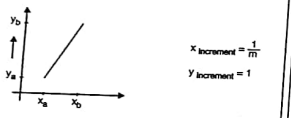


Fig. 2.3.12

9) Similarly if $|Dx| < |Dy|$ and $y_a > y_b$

Then the values of increments in x and y-direction are $-\frac{1}{m}$ and – 1.

Refer Fig. 2.3.13.

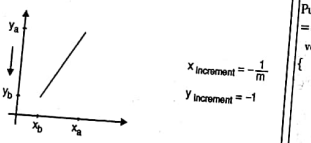


Fig. 2.3.13

Program for DDA line generation

```
/*************************************
Program :
A C program to draw a line by using DDA linegeneration
algorithm
*************************/
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<dos.h>
#include<graphics.h>
void dda(int x1,int y1,int x2,int y2);
/*=========================
Function Name : Main
Purpose : To initialize graphics mode and call DDA
function
=============*/
void main()
{
    int gd = DETECT, gm; /*detect the graphics drivers
automatically*/

    int x1,y1,x2,y2;
 initgraph(&gd,&gm,"c:\\tcplus\\bgi"); /*Initialise to graphics
mode*/
  cleardevice();
  cout<< " DDA Line generation algorithm";
  cout<< " \n enter the starting co-ordinates for drawing
line";
  cin>>x1>>y1;
  cout<<"\n enter the ending co-ordinates";
  cin>>x2>>y2;
  dda(x1,y1,x2,y2);
  cout<< "\n Thank you";
  getch();
  closegraph();
}
/*=========================
Function Name : dda
Purpose : To draw a line using DDA algorithm
============*/
 void dda(int x1,int y1,int x2,int y2)
{
    int i,dx,dy,steps;
    float x,y;
    float xinc,yinc;
    dx = (x2-x1);
    dy = (y2-y1);
if(abs(dx)>=abs(dy)) /* decide whether to move along x-
direction */
```



```
    steps = dx;      /* or y-direction */
    else
        steps = dy;
xinc =(float) dx/steps; /*calculate the increment value for x &
y*/
    yinc =(float) dy/steps;
    x = x1;
    y = y1;
putpixel (x,y,WHITE);      /* plot first point */
    for (i=1;i<steps;i++)
        {
            x = x + xinc;
            y = y + yinc;
/* convert the floating value to its nearest interger value.i.e.
same as use of floor or ceil function */
            x1 = x + 0.5 ;
            y1 = y + 0.5 ;
      putpixel (x1,y1,WHITE);  /* plot the points */
        }
}
```

Output

```
DDA Line generation algorithm
Enter the starting co-ordinates for drawing line
200
300
enter the ending co-ordinates
450
400
Thank you
```

Advantages of DDA

- The DDA algorithm is a faster method for calculating pixel positions than the direct use of line equation $y = mx + b$.
- It is very easy to understand.
- It requires no special skills for implementation.

Disadvantages of DDA

- Because of round off, errors are introduced and causes the calculated pixel position to drift away from the true line path.
- Because of floating point operations the algorithm is time-consuming.

Example

Suppose if we want to draw a line from (1, 1) to (5, 3).

So, $x_1 = 1$, $y_1 = 1$, $x_2 = 5$ and $y_2 = 3$.

$\therefore Dx = (x_2 - x_1) = (5 - 1) = 4$

$Dy = (y_2 - y_1) = (3 - 1) = 2$

As $|Dx| > |Dy|$ the line is of gentle slope category.

$\therefore$ Steps = abs (Dx) = 4

$x_{increment} = \frac{Dx}{steps} = \frac{4}{4} = 1$

$y_{increment} = \frac{Dy}{steps} = \frac{2}{4} = 0.5$

First point we know i.e. x_1, y_1 so plot it.

$\therefore x_{new} = x_{old} + x_{increment} = 1 + 1 = 2$

$y_{new} = y_{old} + y_{increment} = 1 + 0.5 = 1.5$

But over here we have to round off 1.5 as 2 for displaying that point.

For next iteration

$x_{new} = x_{old} + x_{increment} = 2 + 1 = 3$

$y_{new} = y_{old} + y_{increment} = 1.5 + 0.5 = 2$

Now loop steps are tabulated as follows :

I	x	y	Plot
1	1	1	1, 1
2	2	1.5 = 2	2, 2
3	3	2	3, 2
4	4	2.5 = 3	4, 3
5	5	3	5, 3

- For calculation purpose we are using original value but for display only, we are rounding the values.
- If we do this till x_{new} and y_{new} becomes same as end point then we will get line same as shown in Fig. 2.3.14.

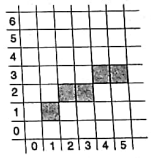


Fig. 2.3.14

Example 2.3.1

Explain DDA line drawing algorithm. Consider the line from (1,1) to (5,6). Use DDA line drawing algorithm to rasterize this line.

Solution :

Given : $x_1 = 1,\ Y_1 = 1$

$x_2 = 5,\ Y_2 = 6$

$\therefore Dx = x_2 - x_1 = 5 - 1 = 4$

$Dy = y_2 - y_1 = 6 - 1 = 5$

Since IDyI > IDxI the line is of steep slope category

$\therefore$ Steps = abs (Dy) = 5

$x_{increment} = \frac{Dx}{Steps} = \frac{4}{5} = 0.8$

$y_{increment} = \frac{Dy}{Steps} = \frac{5}{5} = 1$

As we know 1st point, let's plot it as (1, 1)

$\therefore x_{new} = x_{old} + x_{increment} = 1 + 0.8 = 1.8$

$y_{new} = y_{old} + y_{increment} = 1 + 1 = 2$

But here we need to round off 1.8 as 2 for displaying that point

$\therefore$ for next iteration

$x_{new} = x_{old} + x_{increment} = 1.8 + 0.8 = 2.6$

$y_{new} = y_{old} + y_{increment} = 2 + 1 = 3$

Like this we will calculate points till y_{new} becomes equal to y_2 i.e. 6

The following table summarizes the points.

X	Y	Plot
1	1	1, 1
1.8	2	2, 2
2.6	3	3, 3
3.4	4	3, 4
4.2	5	4, 5
5	6	5, 6

→ 2.3.2 Bresenham's Line Generation (Drawing) Algorithm

→ (Dec. 2014, Dec. 2015)

Q. Derive Bresenham's line drawing algorithm for lines with slope. **MU - Dec. 2014, 10 Marks**

Q. Derive Bresenham's line drawing algorithm for lines with slop < 1. **MU - Dec. 2015, 5 Marks**

Q. Write Bresenham's line drawing algorithm. Also write mathematical derivations for the same. **10 Marks**

- There is one more algorithm to generate lines which is called as Bresenham's line algorithm. The distance between the actual line and the nearest grid location is called error and it is denoted by G.
- Suppose line is as shown in Fig. 2.3.15. After displaying first point (0, 0) we have to select next point. Now there are two candidate pixels (1, 0) and (1, 1). Out of these two pixels we have to select one pixel.

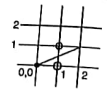


Fig. 2.3.15

- This selection of pixel will depend on the slope of the line. If the slope of line is greater than 0.5 then, we are selecting upper pixel i.e. (1, 1) and if slope is less than 0.5 then we are selecting next pixel as (1, 0).
- As we are interested in only checking whether the point is below 0.5 or above 0.5, we will put condition as if the slope of line is greater than or less than 0.5.

☞ Example

- Suppose the slope of line shown in Fig. 2.3.16 is 0.4. After displaying 1st point as (0, 0) we will add slope of line to error factor G, so that G will become 0.4.
- As we are considering gentle slope case we are moving along x-axis so next pixel will be x = 1 and at that time y will be decided by G. If G > 0.5 then y = 1 else y = 0. But here as G = 0.4 so we have to select next point as (1, 0).

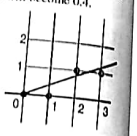


Fig.2.3.16

- Again for next point x is increased by 1 and it becomes x = 2, for that G will become G = G + m i.e. 0.8. Now this time G > 0.5 so upper pixel is more near to desired one. That is why we have to select upper point i.e. (2, 1).
- The beauty of Bresenham's algorithm is it is implemented entirely with integer numbers and the integer numbers are much more faster than floating-point arithmetic. But in previous section we have seen that we are checking slope with 0.5. i.e. floating point variable. So how we can make use of integers for this test?

☞ For Gentle slope case

- Consider P as height of pixel or error. Our condition is whether P > 0.5 or not.

  If P > 0.5 ...(2.3.3)

- But here 0.5 is floating point so we have to convert floating point to integer. For that we will multiply both sides by 2.

$\therefore 2P - 1 > 0$ ...(2.3.4)

- In this equation we have removed fractional part 0.5. But still it may contain fractional part from the denominator of slope. Because every time we are updating P by,

  P = P + slope or P = P + slope – 1

i.e. here we are adding slope to P

But slope is nothing but $\frac{Dy}{Dx}$.

- So the fractional part may come due to Dx. To eliminate this we will multiply the Equation (2.3.4) by Dx.

$\therefore 2PDx - Dx > 0$

Now we will define G = 2PDx – Dx ...(2.3.5)

$\therefore$ Our test will become as G > 0

- Now we will see how this G can be used to decide which row or column to select.

As $G = 2PDx - Dx$

$\therefore G + Dx = 2PDx$

$\therefore \frac{G + Dx}{2Dx} = P$

Now if we consider P = P + slope, then

We will get $\frac{G + Dx}{2Dx} = \frac{G + Dx}{2Dx} + \frac{Dy}{Dx}$

If we further solve this equation then we will get

$G = G + 2Dy$ ...(2.3.6)

- Similarly if we consider P=P + slope – 1, then

We will get

$G = G + 2Dy - 2Dx$ ...(2.3.7)

- To calculate this G we need only addition and subtraction and no multiplication or division. Another thing is now, this G is not containing any fractional part also.
- So, we can use test G > 0 to determine when a row boundary is crossed by a line instead of P > 0.5.
- As the initial value of P is slope, so we have to find initial value of G also. We know from Equation (2.3.5) that

$G = 2DxP - Dx$

- Put initial value of P as Dy/Dx to find initial value of G.

$\therefore G = 2Dx\left(\frac{Dy}{Dx}\right) - Dx$

$\therefore G = 2Dy - Dx$

- For each column we check G. If it is positive we move to the next row and add(2 Dy – 2 Dx) to G, because it is equivalent to P = P + slope – 1, otherwise we will keep the same row and add (2 Dy) to G.

☞ Steps for Bresenham's line drawing algorithm for Gentle slope (m < 1)

1) Accept two endpoints from user and store the left endpoint in (x_0, y_0) as starting point.
2) Plot the point (x_0, y_0)
3) Calculate all constants from two endpoints such as Dx, Dy, 2Dy, 2Dy – 2Dx and find the starting value for the G as G = 2 Dy – Dx.
4) For each column increment x and decide y-value by checking G > 0 condition. If it is true then increment y-value and add (2Dy – 2Dx) to current value of G otherwise add (2Dy) to G and don't increment y-value. Plot next point. See Fig. 2.3.17.

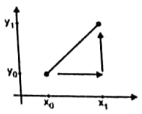


Fig. 2.3.17

5) Repeat step 4 till Dx times.

For steep slope cases just interchange the rolls of x and y i.e. we step along y-direction in unit steps and calculate successive x-values nearest the line path.

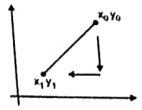


Fig. 2.3.18

If the initial position for a line with positive slope is right endpoint, as shown in Fig. 2.3.18, then we have to decrease both x and y as we step from right to left.

☞ Program for Bresenham's line generation

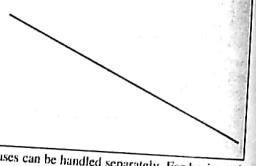
```
/*************************************
Program :
A C program to draw a line by using Bresenham's line
generation algorithm
*************************/
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
#include<dos.h>
#include<graphics.h>
void bresen(int x1,int y1,int x2, int y2);
int sign (float arg);
/*============================
Function Name  : Main

Purpose        : To initialize graphics mode and call
                 Bresenham function
============================*/
void main()
{
int gd = DETECT, gm;
int x1,y1,x2,y2;
init graph (&gd,&gm,"c:\\tcplus\\bgi");
```

```
cleardevice();
cout<< " Bresenhams line drawing ";
cout<< "\n enter the starting point's co-ordinates for line ";
cin>>x1>>y1;
cout<<"\n enter the ending points co-ordinates";
cin>>x2>>y2;
bresen(x1,y1,x2,y2);
cout<< "\n Thank you";
getch();
closegraph();
}
/*============================
Function Name   : bresen
Purpose         : To draw a line using bresenham's line
                  drawing algorithm
============================*/
void bresen(int x1,int y1,int x2,int y2)
{
    int s1,s2,exchange,y,x,i;
    float dx,dy,g,temp;
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    x = x1;
    y = y1;
    s1 = sign(x2-x1);
    s2 = sign(y2-y1); /* interchange dx & dy depending on
the slope of the line */
    if(dy>dx)
    {
        temp = dx;
        dx = dy;
        dy = temp;
        exchange = 1;
    }
    else
        exchange = 0;
    g = 2 * dy - dx;
    i = 1;
    while(i<=dx)
    {
        putpixel(x,y,WHITE);
        delay(10);
        while(g>=0)
        {
            if(exchange==1)
                x = x + s1;
            else
                y = y + s2;
            g = g + 2dy - 2dx;
        }
        if(exchange==1)
            y = y + s2;
        else
            x = x + s1;
            g = g + 2 * dy;
            i++;
    }
}
int sign(float arg)
    {
    if (arg<0)
        return -1;
        else if(arg==0)
            return 0;
                else return 1;
    }
```

Output

```
Bresenhams line drawing,
Enter the starting point's co-ordinates for line drawing.
200
200
enter the ending points co-ordinates
250
450
Thank you
```



- Special cases can be handled separately. For horizontal lines we should not increase y. We have to just increase/decrease x by 1 every time till endpoint.
- For vertical lines we should not increase x we have to just increase/decrease y by 1 every time till endpoint.
- Similarly for diagonal lines we will increase/decrease both x and y by 1 every time till endpoint.

Example

- Plot a line by using Bresenham's line generation algorithm from (1, 1) to (5, 3). See Fig. 2.3.19.

Given :

$x_1 = 1, y_1 = 1$

$x_2 = 5, y_2 = 3$

$\therefore Dx = x_2 - x_1 = 4$

$Dy = y_2 - y_1 = 2$

$G = 2Dy - Dx$

$= 2(2) - (4)$

$G = 0$

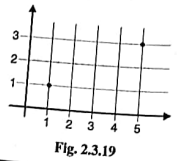


Fig. 2.3.19

- Plot 1st point (1, 1) here as |Dx| > |Dy| that means line is Gentle slope, so here we have to move on x till x_2 i..e. 5.

After ploting 1st point as (1, 1) increase x by 1

$\therefore x = 2$

Here G = 0

We have to increase y by 1 and update G as,

$G = G + 2(Dy - Dx)$

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

- So, plot next point as (2, 2) then again increase x by 1. Now it will become x = 3.

Here G = – 4.

So, don't increase y just update G only as

$G = G + 2Dy$

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

So, plot (3, 2) go on doing this till x reaches to x_2.

We will get points as,

X	Y
1	1
2	2
3	2
4	3
5	3

- So, the final points on line will be as shown in Fig. 2.3.20.

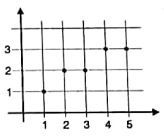


Fig. 2.3.20

- Bresenham algorithm is much more efficient than DDA, it requires only integers and requires only addition and subtraction operations.

Example 2.3.2

Consider the line from (2, 7) to (5, 5). Use Bresenham's line drawing algorithm to rasterize this line.

Solution :

Given : $x_1 = 2, y_1 = 7$

$x_2 = 5, y_2 = 5$

$\therefore Dx = |x_2 - x_1| = |5 - 2| = (3)$

$Dy = |y_2 - y_1| = |5 - 7| = (2)$

$G = 2Dy - Dx = 2(2) - 3 = 1$

Plot 1st point (2, 7)

Since |Dx| > |Dy| the line is having gentle slope.

So we have to move along x till x_2 i.e. 5; after plotting 1st point as (2,7), increase x by 1.

$\therefore x = 3$

Here G > 0

$\therefore$ We have to decrease y by 1 because $y_1 > y_2$ and update G as

$G = G + 2(Dy - Dx)$

$= 1 + 2(2 - 3) = 1 + 2(-1) = 1 - 2 = -1$

$\therefore$ Plot the next point as (3,6)

Now again increasing x by 1, x becomes 4; here G = – 1

$\therefore$ Don't modify y and update G as

$G = G + 2Dy = -1 + 2(2) = 3$

So, plot next point as (4,6). Go on doing this till x becomes x_2.

$\therefore$ Here the points we are getting will be

X	Y
2	7
3	6
4	6
5	5

Example 2.3.3

Consider the line from (4, 9) to (7, 7).

Draw a line using Bresenham's line drawing algorithm.

Solution :

Given : $x_1 = 4, y_1 = 9$

$x_2 = 7, y_2 = 7$

$\therefore Dx = |x_2 - x_1| = |7 - 4| = 3$

$Dy = |y_2 - y_1| = |7 - 9| = 2$

$G = 2Dy - Dx = 2(2) - 3 = 4 - 3 = 1$

Let's plot 1st point as (4, 9)

Since |Dx| > |Dy| the line is of Gentle slope category, so we have to move along x till x_2 reaches.

After potting 1st point as (4, 9) increase x by 1

$\therefore x = 5$

Here G > 0

So update y by – 1 because $y_1 > y_2$ and modify G as

$G = G + 2(Dy - Dx)$

$= 1 + 2(2 - 3) = 1 + 2(-1)$

$= -1$

$\therefore$ Plot next point as (5, 8)

Now again increasing x by 1, x becomes 6. Here G = – 1

$\therefore$ Don't modify y and update G as

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

$= -1 + 4 = +3$

$\therefore$ Plot next point as (6, 8)

Now again increase x by 1, x becomes 7, Here G = 3

i.e. $G > 0$

$\therefore$ update y by – 1 because y1 > y2 and modify G as

$G = G + 2(Dy - Dx)$

$= 3 + 2(2 - 3) = 3 + 2(-1) = 1$

$\therefore$ Plot next point as (7, 7) which is our another end point

Hence we get following points on line (4, 9) to (7, 9) as

X	Y
4	9
5	8
6	8
7	7

Example 2.3.4

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

Solution :

Given : $x_1 = 4,\ y_1 = 4$

$x_2 = 12,\ y_2 = 9$

$\therefore Dx = |x_2 - x_1| = |12 - 4| = 8$

$Dy = |y_2 - y_1| = |9 - 4| = 5$

$G = 2Dy - Dx = 2(5) - 8 = 10 - 8 = 2$

Let's plot 1st point as (4,4)

Since |Dx| > |Dy| the line is of Gentle slope category

So we have to move along x till x_2 i.e. 12

After plotting 1st point as (4, 4), increase x by 1.

$\therefore$ x becomes equal to 5

Here $G > 0$

So update y by 1 and modify G as

$G = G + 2(Dy - Dx) = 2 + 2(5 - 8) = 2 + 2(-3)$

$= 2 - 6$

$= -4$

$\therefore$ Plot next point as (5, 5)

Now again increasing x by 1, x becomes 6, Here $G = -4$

$\therefore$ Don't modify y and update G as

$G = G + 2Dy = -4 + 2(5) = -4 + 10 = 6$

Plot next point as (6, 5). Go on doing this till x becomes x_2.

Hence we get following points on a line (4, 4) to (12, 9)

X	Y
4	4
5	5
6	5
7	6
8	6
9	7
10	7
11	8
12	9

Example 2.3.5

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

Solution : Refer Section 2.3.2.

Given : $x_1 = 0 \qquad y_1 = 0$

$x_2 = 6 \qquad y_2 = 6$

$\therefore Dx = |x_2 - x_1| = |6 - 0| = 6$

$Dy = |y_2 - y_1| = |6 - 0| = 6$

$G = 2Dy - Dx = 2(6) - 6 = 12 - 6 = 6$

Lets plot 1st point as (0, 0)

Since Dx = Dy the line is in gentle slope category. The line is having slope as

$$\frac{Dy}{Dx} = \frac{6}{6} = 1$$

It means we have to move along x direction from x_1 to x_2 and find the corresponding y value.

After plotting 1st point as (0, 0), increase x by 1.

$\therefore$ x becomes 1

Here $G > 0$, so update y by 1 and modify G as

$G = G + 2(Dy - Dx) = 6 + 2(6 - 6) = 2$

$\therefore$ Plot next point as (1, 1)

Now again increasing x by 1, x becomes 2, Here $G = 6$

$\therefore$ Since $G > 0$, so update y by 1 and modify G as

$G = G + 2(Dy - Dx) = 6 + 2(6 - 6) = 6$

$\therefore$ Plot next point as (2, 2)

Go on doing this till x becomes x_2. Hence we get following points on a line(0, 0) to (6, 6) as

X	Y
0	0
1	1
2	2
3	3
4	4
5	5
6	6

Example 2.3.6

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

Solution : Refer Section 2.3.2.

Given : $x_1 = 2, \qquad y_1 = 2$

$x_2 = 6, \qquad y_2 = 5$

$\therefore Dx = x_2 - x_1 = 4 \qquad Dy = y_2 - y_1 = 3$

$G = 2Dy - Dx = 2(3) - (4)$

$G = 2$

Plot 1st point (2, 2) here as |Dx| > |Dy| that means line is Gentle slope, so we have to moveon x till x_2 i..e. 6.

After plotting 1st point as (2, 2) increase x by 1

$\therefore$ x = 3 Here G= 2

We have to increase y by 1 and update G as,

$G = G + 2(Dy - Dx) = 2 + 2(3 - 4) = 0$

So, plot next point as (3, 3) then again increase x by 1. Now it will become x = 4.

Here G = 0.

We have to increase y by 1 and update G as,

$G = G + 2(Dy - Dx) = 0 + 2(3 - 4) = -2$

So, plot next point as (4, 4) then again increase x by 1. Now it will become x = 5.

Here $G = -2$.

So, don't increase y just update G only as

$G = G + 2Dy = -2 + 2(2) = 2$

So, plot (5, 4) go on doing this till x reaches to x_2.

We will get points as,

X	Y
2	2
3	3
4	4
5	4
6	5

2.3.3 Comparison of DDA and Bresenham's Line Drawing Algorithm

Sr. No.	DDA	Bresenham
1.	Based on increment method.	Based on increment method.
2.	Use floating point arithmetic.	Use only integers.
3.	Slower than Bresenham	Faster than DDA
4.	Use of multiplication and division operations.	Use of only Addition and Subtraction operations.
5.	To display pixel we need to use either floor or ceil function.	No need of floor or ceil function for display.
6.	Because of floor and ceil function error component is introduced.	No error component is introduced.
7.	The co-ordinate location is same as that of Bresenham.	The co-ordinate location is same as that of DDA.

2.4 Thick Line Generation

➔ (May 2013)

Q. Explain the method to draw a thick line using Bresenham's algorithm. MU - May 2013, 5 Marks

- We can produce thick lines also, whose thickness is greater than one pixel. To produce a thick line segment, we can run two line drawing algorithms in parallel to find the pixels along the line edges.
- As we are moving to next pixel of line 1 and line 2, we must also turn on all the pixels which lie between the boundaries. See Fig. 2.4.1.

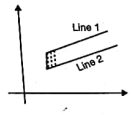


Fig. 2.4.1

- To be more specific, if we want to draw a gentle slope line from (x_a, y_a) to(x_b, y_b) with thickness w, as shown in Fig. 2.4.2., then the corner points of thick line become.

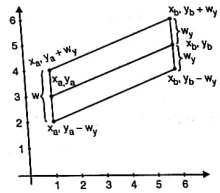


Fig. 2.4.2

$(x_a, y_a + w_y)$, $(x_b, y_b + w_y)$, $(x_a, y_a - w_y)$, $(x_b, y_b - w_y)$

Where w_y can be calculated as,

$$w_y = \frac{w-1}{2}\left(\frac{((x_b - x_a)^2 + (y_b - y_a)^2)^{1/2}}{|x_b - x_a|}\right)$$

- w_y is the amount by which the boundary line is moved from the center.
- (w – 1) factor is the desired width minus one pixel thickness, we automatically receive from drawing the boundary. We divide this by 2 because half the thickness will be used to offset the top boundary and other half to move the bottom boundary. The factor containing x and y-values is needed to find the amount to shift up and down in order to achieve proper width w.

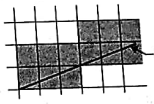


Fig. 2.4.3

- The above mentioned is one of the general method. For special thing we can have other alternatives also. Such as if thickness required is 2 and slope of line is < 1 i.e. line in gentle slope category. See Fig. 2.4.3. Then we can simply plot pixel or draw another line next to actual line.
- If thickness is ≥ 3 then alternately plot the pixels above and below the actual line.
- If slope is > 1 i.e. line is steep slope category then pick the pixels which are at right and left of the line. i.e. we are plotting 2 pixels on right side and one pixel on left of line. See Fig. 2.4.4.

Fig. 2.4.4

Syllabus Topic : Midpoint Algorithm for Circle

2.5 Circle Generating Algorithms

- A circle is a set of points that are placed at a given distance 'r' from centre (x_c, y_c). See Fig. 2.5.1. This distance relationship is expressed by Pythagorean Theorem as,

$$(x - x_c)^2 + (y - y_c)^2 = r^2$$

- We could use this equation to calculate the position of points on circumference of circle by stepping along x-axis from $(x_c - r)$ to $(x_c + r)$ and calculating the corresponding y-values at each position by using,

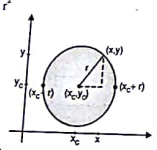


Fig. 2.5.1

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

- We derived this equation from the equation of circle only.

$$\text{i.e. } (y - y_c)^2 = r^2 - (x - x_c)^2$$

$$y^2 = y_c^2 + r^2 - (x - x_c)^2$$

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

- But this is not best method for generating circle as one of the problem with this method is that it leaves uneven spaces and needs more computations at each step. See Fig. 2.5.2.
- We can minimize this problem of spacing by interchanging the position of x and y.
- But there is another way to solve this spacing problem. And that is by using polar co-ordinates r and θ. We use *parametric polar form* to yield the pair of equation as,

Fig. 2.5.2

$$x = x_c + r \cos\theta$$

$$y = y_c + r \sin\theta$$

- With this we are displaying points at fixed angular step size. The step size chosen for θ will depend on application. See Fig. 2.5.3.
- We can reduce the computations by considering the symmetry of circles. By using this symmetry property there is no need to find each and every point on the boundary of the circle

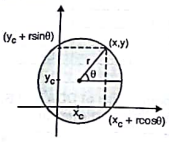


Fig. 2.5.3

- We can divide the circle in 4 quadrants or in 8 octants also. Here we will find only one octant's co-ordinates and then we will just replicate that one octant's co-ordinates to rest seven octants.
- See Fig. 2.5.4. Suppose we have calculated a point (x, y) then from this (x, y) we will plot rest seven points as (x, y), (y, x), (– x, y),(– y, x), (x, – y), (y, – x),(– x, – y), (– y, – x). So using this symmetry proper circle generation becomes fast.

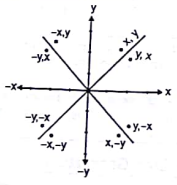


Fig. 2.5.4

- With Cartesian equation of Pythagorean we have to perform multiplication and square root operations, while in parametric equation we have to perform multiplications and trigonometric calculations.

- Both these methods require more computational time. So we have another method of *Bresenham's* which totally deals with integers and performs only addition and subtraction, so it is fast also. This method is applicable for different types of curves also. There is one more approach which is called as *midpoint circle generation*. This approach generates the same pixel positions as the Bresenham's circle algorithm. So first we will see midpoint circle generation algorithm.

2.5.1 Midpoint Circle Generation Algorithm

→ **(May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, Dec. 2016)**

Q. Specify midpoint circle algorithm. **MU - May 2014, 5 Marks**

Q. Explain midpoint circle algorithm. Use the same to plot the circle, whose radius is 10 units. **MU - Dec. 2014, Dec. 2015, May 2016, Dec. 2016, 10 Marks**

Q. Explain the midpoint circle generating algorithm. **MU - May 2015, 8 Marks**

Q. Explain mid-point circle algorithm. In order to support your explanation, show mathematical derivation. **(8 Marks)**

Q. Derive Mid point circle algorithm. **(10 Marks)**

- Here we have to determine the closest pixel position to the specified circle's path at each step. For this we have to accept radius 'r' and center point x_c and y_c. See Fig. 2.5.5. We will first see for center point as origin (0, 0).
- Then each calculated point is to be moved to proper position by adding x_c and y_c to corresponding x and y-value. In first quadrant we are moving in x-direction with starting point as x = 0 to ending point x = y.
- To apply a midpoint method, we define a circle function.

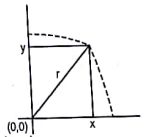


Fig. 2.5.5

$f_{circle}(x, y) = x^2 + y^2 - r^2$ which is same as $r^2 = x^2 + y^2$

Which is derived from equation of circle,

$(x - x_c)^2 + (y - y_c)^2 = r^2$. But here we are considering x_c, y_c as (0, 0) $\therefore x^2 + y^2 = r^2$.

- Any point (x, y) on the boundary of the circle with radius r satisfies the equation $f_{circle}(x, y) = 0$. If point is inside the circle, then circle function is negative and if point is outside circle then circle function is positive.

i.e. if

$f_{circle}(x, y) < 0$, then x, y is inside the circle boundary

$= 0$, then x, y is on circle boundary

> 0, then x, y is outside the circle boundary.

- Thus circle function is the decision parameter in this algorithm. Assuming that we have just plotted x_k, y_k point.
- Now we have to determine whether next point $(x_k + 1, y_k)$ is closer or $(x_k + 1, y_k - 1)$ is closer to actual circle. See Fig. 2.5.6.

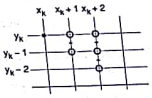


Fig. 2.5.6

- Our decision parameter (P_k) is circle function.

$\therefore P_k = f_{circle}(x_k + 1, y_k)$ or $P_k = f_{circle}(x_k + 1, y_k - 1)$

- Therefore we will take midpoint of these two points as,

$P_k = f_{circle}(x_k + 1, y_k - 1/2)$

$= (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$ from circle equation

- If $P_k < 0$, then it means this point $(x_k + 1, y_k - 1/2)$ is inside the circle. So we have to plot pixel $(x_k + 1, y_k)$.

After this we have to increase x by 1.

i.e. $P_{k\,new} = f_{circle}(x_k + 2, y_k - 1/2)$

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

if we solve this we will get

$P_{k\,new} = P_k + (2\,x_k + 1)$

i.e. when, earlier we have not changed row.

- If $P_k \geq 0$ then we will select $y_k - 1$ for displaying. Like this we will select the point to be displayed.

$P_{knew} = f_{circle}(x_k + 2, y_k - 3/2)$

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

After solving above equation we will get,

$P_{knew} = P_k + (2\,x_k - 2\,y_k + 1)$

i.e. when, earlier we have changed row.

- Similarly the initial decision parameter is obtained by evaluating the circle function at start position $(x_0, y_0) = (0, r)$. See Fig. 2.5.7.

Now calculate the decision parameter.

$P_0 = f_{circle}(x_k + 1, y_k - 1/2)$

But $x_k = 0$ and $y_k = r$

$\therefore \quad x_k + 1 = 1 \quad$ and $\quad y_k - 1/2 = r - \frac{1}{2}$

$$P_0 = f_{circle}\left(1, r - \frac{1}{2}\right) = 1^2 + \left(r - \frac{1}{2}\right)^2 - r^2$$

$$= 1 + r^2 + \frac{1}{4} - 2\frac{1}{2}r - r^2 = 1 + \frac{1}{4} - r$$

$$P_0 = \frac{5}{4} - r$$

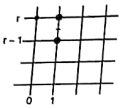


Fig. 2.5.7

If radius r is integer we will round it as $P_0 = 1 - r$.

Steps for midpoint circle generation algorithm

1) Accept radius 'r' and center of circle (x_c, y_c) from user and plot 1st point on circumference of circle.
$$(x_0, y_0) = (0, r)$$
2) Calculate the initial value of the decision parameter.
$$P_0 = 1 - r$$
3) If we are using octant symmetry property to plot the pixels, then until (x < y) we have to perform following steps.

If P_k is less than 0

modify P_k as $P_k = P_k + 2x + 1$ and

then increase x by 1

otherwise

modify P_k as $P_k = P_k + 2(x - y) + 1$ and

increase x by 1

and decrease y by 1

4) Determine the symmetry points in other octants also
5) Since we have derived all formulas by considering center point as origin. Move each calculated pixel position (x, y) on to the circular path centered on (x_c, y_c) and plot the co-ordinate values as
$$x = x + x_c \text{ and}$$
$$y = y + y_c$$

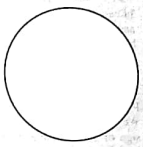
But actually the center point may be anything, so we have to add that center co-ordinates to midpoint x and y to plot the point.

Midpoint Circle generation program

```
/*********************************************
Program :
A C program to draw a circle by using midpoint circle
generation algorithm
*********************************************/
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
#include<dos.h>
void circ_midpt(int x,int y,int rad);
void display(int,int,int,int);
/*============================
Function Name : Main
Purpose : To initialize graphics mode and
call mid point circle function
============================*/
void main()
{
int gd = DETECT, gm, x,y,r;
initgraph(&gd,&gm,"c:\\tcplus\\bgi");
cleardevice();
cout<<" Midpoint circle generation algorithm ";
cout<<"\n enter the center co-ordinates for the circle";
cin>>x>>y;
cout<<"\n enter the radius of the circle";
cin>>r;
circ_midpt(x,y,r);
getch();
closegraph();
}
/*============================
Function Name : circ_midpt
Purpose : To draw a circle by mid point circle
generation algorithm
============================*/
void circ_midpt(int x,int y,int rad)
{
int x1,y1;
float dp; //initialising the descision parameter.
x1 = 0; //initialising the X,Y cordinates.
y1 = rad;
dp = 1 - rad;
while(x1<=y1)
{
if(dp<=0)
{
x1++;
dp += (2 * x1) + 1;
}
else
{
x1++;
y1--;
dp += 2*(x1-y1)+1;
}
display(x1,y1,x,y);
}
}
/*============================
Function : Name: display
Purpose : To plot the pixels on circle by symmetry
property.
============================*/
void display (int x1,int y1,int x,int y)
{
putpixel(x1+x,y1+y,WHITE);
//plotting the pixels by using symmetry.
putpixel(x1+x,y-y1,WHITE);
putpixel(x-x1,y1+y,WHITE);
putpixel(x-x1,y-y1,WHITE);
putpixel(x+y1,y+x1,WHITE);
putpixel(x+y1,y-x1,WHITE);
putpixel(x-y1,y+x1,WHITE);
putpixel(x-y1,y-x1,WHITE);
}
```

Output

```
Midpoint circle generation algorithm
enter the center co-ordinates for the circle
320
240
enter the radius of the circle
150
```



Example 2.5.1

Plot a circle whose radius is 3 and whose center co-ordinates are (0, 0).

Solution :

Given : $r = 3$, $x_c = 0$, $y_c = 0$

Plot 1st point as (0, r) i.e. y = r

∴ Plot (0, 3)

Like that find out other points from (0, 3) by using symmetry property

So plot (3, 0)

plot (0, – 3)

plot (– 3, 0)

find $P = 1 - r$ ∴ $P = 1 - 3 = -2$

Till (x < y) we have to perform following

If (P < 0)

In this case P = – 2

So we have to increase x by 1 and

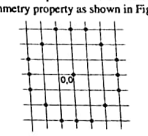
modify P as $P = P + 2x + 1$

i.e. $P = -2 + 2(0) + 1 = -2 + 1 = -1$

Here we are not increasing y.

∴ Plot (x + 1, y) i.e. (1, 3)

Again continuing this process we will get following table. Then plot the co-ordinates of this table by using symmetry property as shown in Fig. P. 2.5.1.



x	y
0	3
1	3
2	2
3	1

Fig. P. 2.5.1

2.5.2 Bresenham Circle Generation

→ (May 2017)

Q. Explain Bresenham's Circle drawing algorithm in detail. MU - May 2017. 10 Marks

- This algorithm does only integer arithmetic which makes it faster than floating point. We are creating only one octant of the circle i.e. from 90° to 45°.
- See Fig. 2.5.9. We derive other seven octants of the circle by symmetry property. The circle is generated by considering center point as origin with radius 'r'. The algorithm calculates one new pixel per step. From any point (x_n, y_n) on the circle, the next point $(x_n + 1, y_n + 1)$ must be either one on right or one to the right and down.

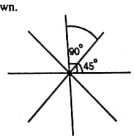


Fig. 2.5.9

- If P_n is a current point with co-ordinate (x_n, y_n) then from Fig. 2.5.10 we will come to know that next point could be either A or B.
- Point A will be having co-ordinates $(x_n + 1, y_n)$ and B will be $(x_n + 1, y_n - 1)$ so for selecting one of these two candidate pixels we have to perform some test.
- To do this, we know that a function of a circle is $f_{circle}(x, y) = x^2 + y^2 - r^2$.

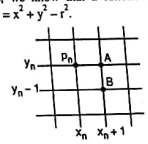


Fig. 2.5.10

- This function determines a circle with radius 'r' around the origin in the following way :
  If f (x, y) = 0 then (x, y) point is on circle.
  If f (x, y) < 0 then (x, y) point lie inside the circle.
  If f (x, y) > 0 then (x, y) point lie outside the circle.
  i.e. for point A the equation of circle will be
  $f(A) = f(x_n + 1, y_n) = (x_n + 1)^2 + y_n^2 - r^2$
  and for point B
  $f(B) = f(x_n + 1, y_n - 1) = (x_n + 1)^2 + (y_n - 1)^2 - r^2$
- Now if f (A) = 0 then point A lies on circle so we have to plot that point else we have to check, if (f (A) < 0) i.e. Is point A lies inside the circle. Or if (f (A) > 0), i.e. Is point A lies outside the circle and accordingly we have to select a particular point.
- To select one point from A and B we are having five different cases, Fig. 2.5.11 shows sketches of all the five cases with two candidate pixels A and B.

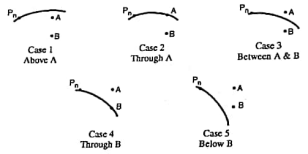


Fig. 2.5.11

- To determine which point to select we are using equation of circle. To be more specific, we are dealing with only the sign of that function, if it is negative or positive.
- Let us summarize this result in a tabular form for all the 5 cases.

	Case 1	2	3	4	5
f (A)	–	0	+	+	+
f (B)	–	–	–	0	+
Sum	–	–	–, 0, +	+	+

- Observe the case 2 of Fig. 2.5.11. Here curve is passing through point Ai.e. f (A) must be equal to 0; And as point B is lying inside, f (B) < 0. So we are selecting point A.
- Similarly observe case 4 of Fig. 2.5.11. Here point B is lying on curve and point A is lying outside. So, f (A) must be > 0 and f (B) must be equal to 0.
- These two cases are the simplest cases. But if the situation is like case 1 orcase 5. Then in both cases the curve lies on one side of both points.
- Let's concentrate on case 1. Here curve is above A and B point. It means both points A and B are lying inside the curve then both f (A) and f (B) must be < 0. So which point to select? Since both points are inside only, we are introducing new term "sum" which will be summation of both f (A) and f (B) i.e. sum = f (A) + f (B). Here we are going to plot a point A because it is more near to actual curve than the point B.
- Similarly in case 5, the curve is below to both points A and B i.e. both points are outside means the f (A) and f (B) must be > 0. Here also we are selecting a point which is more nearer to the curve. In this case point B is more nearer so we are selecting that point.
- Now to decide when to select point A or point B, we are using sum variable. Here we are interested in only the sign of sum variable.
- In case 1 both f (A) and f (B) are negative so the sum will be obviously negative and in case 5 both f (A) and f (B) are positive so the sum will be positive only. From this we can make a statement as, if the sign of sum variable is negative then select point A and if the sign of sum variable is positive then select point B.
- In case 3 of Fig. 2.5.11 point A lies outside the curve and B lies inside. So f (A) must be > 0 and f (B) must be < 0. Then the sum variable will contain either negative or 0 or positive value, depending on the value of f (A) and f (B). If the sum is negative or 0, then pixel A is closer to circle and we have to select A point. If it is positive then, we have to select point B.
- We are going to choose the point A or B depending on sign of sum variable. It is not necessary to calculate both formulas and the sum for every step.
- The starting pixel P_0 is taken as $(x_0, y_0) = (0, r)$. Refer Fig. 2.5.12.

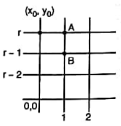


Fig. 2.5.12

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

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

$\therefore$ Sum $= f(A) + f(B)$
$= 1 + 2 - 2r$
$\therefore$ Sum $= 3 - 2r$

So depending on whether sum is <, > or = 0, we will select A or B as next point. And we can find next sum going from P_1 to P_2 by considering old sum (S) value. Consider Fig. 2.5.13.

If we want to make it generalized then
Sum $= f(A) + f(B)$
$= f(x_n + 1, y_n) + f(x_n + 1, y_n - 1)$
$\therefore S = (x_n + 1)^2 + y_n^2 - r^2 + (x_n + 1)^2 + (y_n - 1)^2 - r^2$

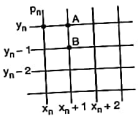


Fig. 2.5.13

Step 1 : If S < 0 then we will print next point as A i.e. $(x_n + 1, y_n)$. We will call this point as $(P_n + 1)$. Now in order to go for next pixel i.e. $(P_n + 2)$ we must compute sum again. We will call this as S_{new} and express it using point P_n. See Fig. 2.5.14.

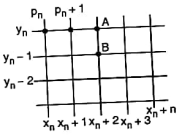


Fig. 2.5.14

$S_{new} = f(A) + f(B)$
$= f(x_n + 2, y_n) + f(x_n + 2, y_n - 1)$
$\therefore S_{new} = (x_n + 2)^2 + y_n^2 - r^2 + (x_n + 2)^2 + (y_n - 1)^2 - r^2$
Solving this we will get,
$S_{new} - S = 4x_n + 6$
$\therefore S_{new} = S + 4x_n + 6$

$\therefore$ For the next pixel $(P_n + 2)$ we have to select point A if $S_{new} <= 0$ else select point B.

Step 2 : But if at initial stage, S > 0 then we will print $(P_n + 1)$ as $(x_n + 1, y_n - 1)$. See Fig. 2.5.15.
Now in this case to find $P_n + 2$
$S_{new} = f(A) + f(B)$
$= f(x_n + 2, y_n - 1) + f(x_n + 2, y_n - 2)$

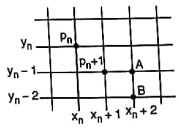


Fig. 2.5.15

$\therefore S_{new} = (x_n + 2)^2 + (y_n - 1)^2 - r^2 + (x_n + 2)^2 + (y_n - 2)^2 - r^2$
Solving this we will get,
$S_{new} - S = 4(x_n - y_n) + 10$
$\therefore S_{new} = S + 4(x_n - y_n) + 10$

This shows that calculating S_{new} from old 'S' is very simple. It uses very few operations like addition, subtraction and multiplication. Here, we are making use of just sign of this 'S' variable to determine which pixel to draw.

Steps for Bresenham's circle generation algorithm

1) Accept radius and center co-ordinates from user and plot first point on circumference of circle.
   (x, y) = (0, r)
2) Calculate the initial value of decision parameter.
   S = 3 – 2r
3) If we are using octant symmetry property to plot the pixels then until (x < y) we have to perform following steps :
   If (S <= 0)
   Update S by S = S + 4x + 6 and increase x by 1.
   else
   Update S by S = S + 4 (x – y) + 10
   and increase x by 1 and decrease y by 1.
4) Determine the symmetry points in other octants also.
5) Move each calculated pixel position (x, y) on to the circular path centered on(x_c, y_c) and plot the co-ordinate values as,
   x = x + x_c and y = y + y_c

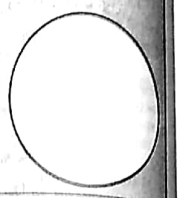
Bresenham's Circle Generation program

```
/*********************************************
Program :
A C program to draw a circle by using Bresenham's circle
generation algorithm
*********************************************/
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
#include<dos.h>
void circ_bre(int x,int y,int rad);
void display(int,int,int,int);
void main()
{
    int gd = DETECT, gm, x,y,r;
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
    cleardevice();
```

```
cout<<"\nBresenhams circle generation algorithm ";
cout<<"\n\n enter the center co-ordinates for circle ";
    cin>>x>>y;
    cout<<"\n enter the radius of the circle";
    cin>>r;
    circ_bre(x,y,r);
    getch();
    closegraph();
}
/*-----------------------------------------
Function Name   : circ_midpt
Purpose         : To draw a circle by mid point circle
                  generation algorithm
-----------------------------------------*/
void circ_bre(int x,int y,int rad)
{
float dp;   //initialising the decision parameter.
    int x1,y1;
x1 = 0;     //initialising the X,Y coordinates.
    y1 = rad;
    dp = 3 - 2*rad;
    while(x1 <= y1)
    {
        display(x1=0)
            dp = (3 * x1) + 6;
        else
        {
            dp = 4*(x1-y1) + 10;
            y1--;
        }
        x1 + + ;
        display(x1,y1,x,y);
    }
}
/*-----------------------------------------
Function Name   : display
Purpose         : To plot the pixels on circle by symmetry
                  property
-----------------------------------------*/
void display (int x1,int y1,int x,int y)
{
    putpixel(x1+x,y1+y,WHITE);    //plotting the pixels.
    putpixel(x1+x,-y1+y,WHITE);
    putpixel(-x+x1,y1+y,WHITE);
    putpixel(-x+x1,-y+y1,WHITE);
    putpixel(y+y1,x+x1,WHITE);
    putpixel(y+y1,-x+x1,WHITE);
    putpixel(-y+y1,x+x1,WHITE);
    putpixel(-y+y1,-x+x1,WHITE);
}
```

Output

```
Bresenhams circle generation algorithm
Enter the center co-ordinates for circle
320
240
enter the radius of the circle
150
```



Example 2.6.2

Plot a circle by Bresenham's algorithm whose radius is 3 and center co-ordinates are (0, 0).

Solution :

Given :

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

Plot 1st point as (0, r) i.e. (0, 3)

here $x = 0$ and $y = r = 3$

Find out rest three points from this by using symmetry property andplot (0, – 3), (– 3, 0), (3, 0).

Then find initial value of decision parameter.

$$S = 3 - 2r$$

$$\therefore S = 3 - 2(3) = -3$$

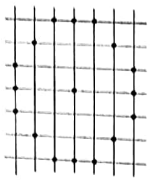
As S is negative i.e. $S < 0$,

$$x = x + 1 \text{ i.e. } x = (0 + 1) = 1$$

$$y = 3 \text{ and}$$

$$S = S + dx + 6 = -3 + 4(0) + 6 = +3$$

Now plot new x and y i.e. plot (1, 3) similarly continue this process till ($x < y$) then we will get following table. Then after plotting these points we will get the circle as shown in Fig. P. 2.5.2.



x	y
0	3
1	3
2	2
3	1

Fig. P. 2.5.2

This pixel positions are similar to the pixel positions indicated by midpoint circle generation algorithm.

Syllabus Topic : Midpoint Algorithm for Ellipse Drawing

2.6 Ellipse Generation

→ **(May 2013)**

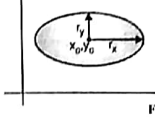
Q. Derive the midpoint algorithm for ellipse generation. **MU - May 2013, 10 Marks**

Q. Explain Midpoint ellipse algorithm with all required mathematical representation. **(12 Marks)**

Q. Derive decision parameters for the midpoint ellipse algorithm (Region 1), assuming the starting position is $(0, r_y)$ and points are to be generated along the curve path in clockwise order. **(10 Marks)**

Ellipse generating algorithms

- Ellipse is nothing but elongated circle. Therefore it can be generated by modifying circle drawing procedures.
- Ellipse has major axis and minor axis. Refer Fig. 2.6.1.



r_x Represents semimajor axis

r_y Represents semiminor axis

Fig. 2.6.1

- Ellipse is symmetric between quadrants, but not symmetric between the two octants of a quadrant.
- Thus we have to calculate pixel positions along the elliptical arc throughout one quadrant, then we obtain positions in the remaining 3 quadrants by using symmetry property. See Fig. 2.6.2.

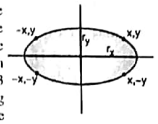


Fig. 2.6.2

Midpoint ellipse generation method

- The *midpoint ellipse* method is applied throughout the first quadrant in two parts i.e. region-1 and region-2. We are forming this regions by considering the slope of the curve. If the slope of the curve is less than –1 then we are in region-1 and when the slope becomes greater than – 1 then in region-2. See Fig. 2.6.3.

i.e. At the boundary between region-1 and 2

$$\frac{dy}{dx} = -1.$$

The slope of the ellipse is calculated as

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

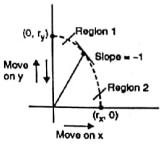


Fig. 2.6.3

At the boundary between region-1 & 2,

$$\frac{dy}{dx} = -1 \quad \text{and} \quad 2r_y^2 x = 2r_x^2 y$$

So, we move out of region-1 when

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

- For region-1, we can start at $(0, r_y)$ and step clockwise along the elliptical path in the 1st quadrant shifting from unit steps x to unit steps in y, till the slope becomes less than – 1 **OR** we can start at $(r_x, 0)$ and select points in anticlockwise shifting from unit steps in y to unit steps in x, till the slope becomes greater than – 1.

Equation of ellipse is,

$$f_{ellipse}(x, y) = r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$$

- Here we are making use of this function to decide whether the point is inside or outside the elliptical curve. Same as in case of circle.

  If $f_{ellipse}(x, y) < 0$ then (x, y) is inside the ellipse boundary.

  $= 0$ then (x, y) is on boundary.

  > 0 then (x, y) is outside the ellipse boundary

- Thus the $f_{ellipse}(x, y)$ acts as a decision parameter. So, we are going to select the next pixel along the path of ellipse, according to the sign of the function.
- We are starting at position $(0, r_y)$ and take unit steps in the x-direction until we reach the boundary between region-1 and 2. Once we reach to the boundary, we switch to unit steps in the y-direction for the remainder of the curve.

Here we have to calculate the slope at each step.

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

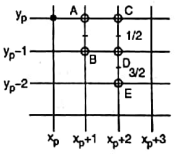
We move to region-2 from region-1 whenever

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

In region-1 if the current pixel is located at (x_p, y_p). Then next candidate pixels are A & B, So the decision variable for region-1 will be $(x_p + 1, y_p - 1/2)$, as we are using midpoint algorithm. See Fig. 2.6.4.

$$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 1, y_p - 1/2)$$

$$dp\text{-}1\text{-}old = r_y^2 (x_p + 1)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 r_y^2$$



Changing x uniformly and deciding which row is nearer

Fig. 2.6.4

1) If $f_{ellipse}(x_p + 1, y_p - 1/2) < = 0$ then print 'A'.

And for next point our candidate pixels will be C and D. So our next midpoint will be $(x_p + 2, y_p - 1/2)$.

$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 2, y_p - 1/2)$

dp-1-new $= r_y^2 (x_p + 2)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 r_y^2$

To make it generalized.

We know that,

dp-1-old $= r_y^2 (x_p + 1)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 \cdot r_y^2$

and dp-1-new $= r_y^2 (x_p + 2)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 \cdot r_y^2$

Now replace $r_x^2 (y_p - 1/2)^2 - r_x^2 \cdot r_y^2$ as dp-1-old $- r_y^2 (x_p + 1)^2$

$= r_y^2 (x_p + 2)^2 + \text{dp-1-old} - r_y^2 (x_p + 1)^2$

$= \text{dp-1-old} + r_y^2 (x_p^2 + 4x_p + 4) - r_y^2 (x_p^2 + 2x_p + 1)$

$= \text{dp-1-old} + r_y^2 \cdot x_p^2 + 4r_y^2 \cdot x_p + 4r_y^2 - r_y^2 x_p^2 - 2r_y^2 x_p - r_y^2$

$= \text{dp-1-old} + 2r_y^2 \cdot x_p + 3r_y^2$

$= \text{dp-1-old} + r_y^2 (2x_p + 3)$

$\therefore$ dp-1-new $= \text{dp-1-old} + r_y^2 (2x_p + 3)$

(i.e. if y is not changed)

2) But if $f_{ellipse}(x_p + 1, y_p - 1/2) > 0$ then print point 'B' and for next point our candidate pixels will be D and E. So our next midpoint will be $(x_p + 2, y_p - 3/2)$.

$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 2, y_p - 3/2)$

$\therefore$ dp-1-new $= r_y^2 (x_p + 2)^2 + r_x^2 (y_p - 3/2)^2 - r_x^2 \cdot r_y^2$

If we represent this in terms of dp-1-old i.e. with respect to (x_p, y_p).

Then dp-1-new will be as follows :

As we know that,

dp-1-old $= r_y^2 (x_p + 1)^2 + r_x^2 (y_p - 1/2)^2 - r_x^2 \cdot r_y^2$

and dp-1-new $= r_y^2 (x_p + 2)^2 + r_x^2 (y_p - 3/2)^2 - r_x^2 \cdot r_y^2$

Replacing $r_y^2 \cdot r_x^2$ as dp-1-old $- r_y^2 (x_p + 1)^2 - r_x^2 (y_p - 1/2)^2$ we get

dp-1-new $= r_y^2 (x_p + 2)^2 + r_x^2 (y_p - 3/2)^2 + \text{dp-1-old} - r_y^2 (x_p + 1)^2 - r_x^2 (y_p - 1/2)^2$

$= \text{dp-1-old} + r_y^2 [x_p^2 + 4x_p + 4] + r_x^2 \left[y_p^2 - 3y_p + \frac{9}{4}\right] - r_y^2 [x_p^2 + 2x_p + 1] - r_x^2 \left[y_p^2 - y_p + \frac{1}{4}\right]$

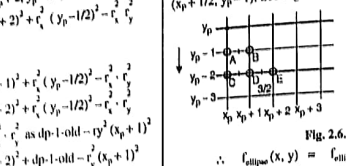
$= \text{dp-1-old} + r_y^2 x_p^2 + 4r_y^2 x_p + 4r_y^2 + r_x^2 y_p^2 - 3r_x^2 y_p + \frac{9}{4} r_x^2 - r_y^2 x_p^2 - 2r_y^2 x_p - r_y^2 - r_x^2 y_p^2 + r_x^2 y_p - \frac{1}{4} r_x^2$

$= \text{dp-1-old} + 3r_y^2 + 2r_y^2 \cdot x_p - 2r_x^2 y_p + 2r_x^2$

$= \text{dp-1-old} + r_y^2 (3 + 2x_p) + r_x^2 (2 - 2y_p)$

dp-1-new $= \text{dp-1-old} + r_y^2 (2x_p + 3) + r_x^2 (-2y_p + 2)$ (i.e. if y is changed)

Similarly, in region-2, if current pixel is at (x_p, y_p) then candidate pixels are A and B, so midpoint will be $(x_p + 1/2, y_p - 1)$. See Fig. 2.6.5.



Changing y uniformly and deciding which column is closer

Fig. 2.6.5

$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 1/2, y_p - 1)$

$\therefore$ dp-2-old $= f_{ellipse}(x_p + 1/2, y_p - 1)$

$\therefore$ dp-2-old $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$

1) If dp-2-old > 0 then A is selected i.e. $(x_p, y_p - 1)$ and next candidate pixels will be C and D. So our next midpoint will be $(x_p + 1/2, y_p - 2)$.

$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 1/2, y_p - 2)$

dp-2-new $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 2)^2 - r_x^2 \cdot r_y^2$

To make it generalized.

We know that,

dp-2-old $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$

dp-2-new $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 2)^2 - r_x^2 \cdot r_y^2$

Replacing $r_x^2 (y_p - 2)^2 - r_x^2 \cdot r_y^2$ as dp-2-old $- r_y^2 (x_p + 1)^2$, we get

dp-2-new $= r_x^2 (y_p - 2)^2 + \text{dp-2-old} - r_x^2 (y_p - 1)^2$

$= \text{dp-2-old} + r_x^2 (y_p^2 - 4y_p + 4) - r_x^2 (y_p^2 - 2y_p + 1)$

$= \text{dp-2-old} + r_x^2 \cdot y_p^2 - 4r_x^2 \cdot y_p + 4r_x^2 - r_x^2 \cdot y_p^2 + 2r_x^2 \cdot y_p - r_x^2$

$= \text{dp-2-old} + 3r_x^2 - 2r_x^2 \cdot y_p$

$= \text{dp-2-old} + r_x^2 (-2y_p + 3)$

$\therefore$ dp-2-new $= \text{dp-2-old} + r_x^2 (-2y_p + 3)$

...(i.e. if x is not changed)

2) But If dp-2-old $< = 0$ then point 'H' is selected i.e. $(x_p + 1, y_p - 1)$ and next candidate pixels will be D and E. So our midpoint will be $(x_p + 3/2, y_p - 2)$

$\therefore f_{ellipse}(x, y) = f_{ellipse}(x_p + 3/2, y_p - 2)$

$\therefore$ dp-2-new $= r_y^2 (x_p + 3/2)^2 + r_x^2 (y_p - 2)^2 - r_x^2 \cdot r_y^2$

To make it generalized.

We know that,

dp-2-old $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$

and dp-2-new $= r_y^2 (x_p + 3/2)^2 + r_x^2 (y_p - 2)^2 - r_x^2 \cdot r_y^2$

Replacing $r_x^2 \cdot r_y^2$ as dp-2-old $- r_y^2 (x_p + 1/2)^2 - r_x^2 (y_p - 1)^2$ we get,

dp-2-new $= r_y^2 (x_p + 3/2)^2 + r_x^2 (y_p - 2)^2 + \text{dp-2-old} - r_y^2 (x_p + 1/2)^2 - r_x^2 (y_p - 1)^2$

$= \text{dp-2-old} + r_y^2 \left[x_p^2 + 3x_p + \frac{9}{4}\right] + r_x^2 [y_p^2 - 4y_p + 4] - r_y^2 \left[x_p^2 + x_p + \frac{1}{4}\right] - r_x^2 [y_p^2 - 2y_p + 1]$

$= \text{dp-2-old} + r_y^2 x_p^2 + 3r_y^2 x_p + \frac{9}{4} r_y^2 + r_x^2 y_p^2 - 4r_x^2 y_p - r_y^2 x_p^2 - r_y^2 x_p - \frac{1}{4} r_y^2 - r_x^2 y_p^2 + 2r_x^2 y_p - r_x^2$

$= \text{dp-2-old} + 2r_y^2 x_p + 2r_y^2 - 2r_x^2 y_p + 3r_x^2$

$= \text{dp-2-old} + r_y^2 (2x_p + 2) + r_x^2 (3 - 2y_p)$

$\therefore$dp-2-new=dp-2-old $+ r_y^2 (2x_p + 2) + r_x^2 (3 - 2y_p)$

...(i.e. if x is changed)

- We must also compute the initial condition. There will be two initial conditions, one for region-1 and another for region-2. Let us first see for region-1.
- Assuming ellipse starts at $(0, r_y)$. So candidate pixels are A and B. See Fig. 2.6.6.
- $\therefore$ Midpoint will be $(1, r_y - 1/2)$ and the initial value of decision parameter will be $f_{ellipse}(1, r_y - 1/2)$. The variable dp-1-initial represents initial decision parameter for region-1.

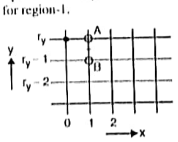


Fig. 2.6.6

dp-1-initial $= r_y^2 + (r_y - 1/2)^2 \cdot r_x^2 - r_x^2 \cdot r_y^2$

$= r_y^2 + \left[r_y^2 - r_y + \frac{1}{4}\right] r_x^2 - r_x^2 \cdot r_y^2$

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

$= r_y^2 + r_x^2 \left(-r_y + \frac{1}{4}\right)$

- At each iteration in region-1, we must not only test the decision variable but also see whether we should switch regions. When the midpoint crosses over into region-2 we change to travel along y-axis. That is, if the last pixel chosen in region-1 is located at (x_p, y_p) then the decision variable is initialized at $(x_p + 1/2, y_p - 1)$. We stop drawing pixels in region-2 when the y-value of the pixel is equal to zero. See Fig. 2.6.7.

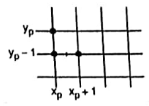


Fig. 2.6.7

The last point of region-1 will be acting as starting point of region-2.

Initial value of region-2 will be $f_{ellipse}(x_p + 1/2, y_p - 1)$.

dp-2-initial $= r_y^2 (x_p + 1/2)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$

$= r_y^2 \left[x_p^2 + x_p + \frac{1}{4}\right] + r_x^2 [y_p^2 - 2y_p + 1] - r_x^2 \cdot r_y^2$

$= r_y^2 x_p^2 + r_y^2 \cdot x_p + \frac{1}{4} r_y^2 + r_x^2 \cdot y_p^2 - 2y_p \cdot r_x^2 + r_x^2 - r_x^2 \cdot r_y^2$

dp-2-initial $= r_y^2 \left[x_p^2 + x_p + \frac{1}{4}\right] + r_x^2 (1 - 2y_p)$

Steps for midpoint ellipse generation

1) Accept r_x, r_y and ellipse center co-ordinates (x_c, y_c) and plot the first point on an ellipse centered on the origin as,

$(x, y) = (0, r_y)$

2) Calculate the initial value of decision parameter for region-1 as,

dp-1 $= r_y^2 + r_x^2 \left(-r_y + \frac{1}{4}\right)$

3) We have to check whether the slope of the curve is < -1, if yes, then we are in region-1 and perform following steps for region-1.

(i) If dp-1 is less than 0 then step along x-axis and modify decision parameter as,

dp-1 $=$ dp-1 $+ r_y^2 (2x_p + 3)$

Otherwise

Step along x-axis and decrease y by 1 and modify decision parameter as,

dp-1 $=$ dp-1 $+ r_y^2 (2x_p + 3) + r_x^2 (-2y_p + 2)$

(ii) Draw the pixel (x, y) by using four point symmetry.

4) If the slope of the curve is > -1 then we have to change the region from 1 to 2. Now the last point of region-1 will be considered as first point of region-2.

5) Calculate the initial value of decision parameter for region-2 as,

dp-2 $= r_y^2 \left(x_p + \frac{1}{2}\right)^2 + r_x^2 (y_p - 1)^2 - r_x^2 \cdot r_y^2$

6) Perform following steps till $(y > 0)$

i) If dp-2 > 0 then

decrease y by 1 and update dp-2.

$dp\text{-}2 = dp\text{-}2 + r_x^2 (-2 y_p + 3)$

otherwise

decrease y by 1 and increase x by 1 and update dp-2 as,

$dp\text{-}2 = dp\text{-}2 + r_y^2 (2 x_p + 2) + r_x^2 (3 - 2 y_p)$

ii) Draw the pixel (x, y) by using four point symmetry.

Program for mid-point ellipse generation

```
/*****************************************
Program :
A C program to draw an ellipse by suing mid-point ellipse
generation algorithm.
*****************************************/
#include<iostream.h>
  #include<dos.h>
  #include<conio.h>
  #include<math.h>
  #include<graphics.h>
  void display (int xs1, int ys1, int x, int y);
/*============================
Function Name   :  ellips1
Purpose         :  To draw a ellipse by mid point
                   ellipse generation algorithm
============================*/
  void ellips1(int xs1,int ys1,int rx, int ry)
  {
     int x,y;
     float d1,d2,dx,dy;
x = 0;            // take start position as (0,ry)
y = ry;           // finding decision parameter d1
d1 = pow(ry,2) - (pow(rx,2) * ry) + (0.25 * pow(rx,2));
     dx = 2 * pow(ry,2) * x;
  dy = 2 * pow(rx,2) * y;
  do                // region one
  {
     display(xs1,ys1,x,y);
     if(d1<0)
     {
        x++;
        dx = dx + (2 * (pow(ry,2)));
        d1 = d1 + dx +(pow(ry,2));
     }
     else
     {
        x++;
     y--;
     dx = dx + (2 * (pow(ry,2)));
     dy = dy - (2 * (pow(rx,2)));
     d1 = d1 + dx - dy + (pow(ry,2));
  }
  }while(dx<dy);    // change over condition for region-2
  do                // region two
    {
       display(xs1,ys1,x,y);
       if(d2>0)
       {  x = x;
          y--;
          dy = dy - (2 * (pow(rx,2)));
          d2 = d2 - dy + pow(rx,2);
       }
       else
       {
          x++;
          y--;
          dy = dy - (2 * (pow(rx,2)));
          dx = dx + (2 * (pow(ry,2)));
       d2 = d2 +dx - dy + pow(rx,2);
    }
  }while(y>0);
}
/*============================
Function Name   :  display
Purpose         :  To plot the pixels on ellipse by
                   symmetry property
============================*/
void display(int xs,int ys,int x,int y)
{
    putpixel(xs+x,ys+y,WHITE);
                  // plot points by using 4 point symmetry
    putpixel(xs-x,ys-y,WHITE);
    putpixel(xs+x,ys-y,WHITE);
    putpixel(xs-x,ys+y,WHITE);
}

int main(void)
{
    int xs1,ys1;
    float rx1,ry1;
    int gd = DETECT,gm;   // Initialise the graphics system
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
    cout<<"\t\tMidpoint Ellipe Drawing Algorithm\n";
    cout<<"Enter the Center Co-ordinates\n";
    cout<<"xc = \t";
    cin>>xs1;
    cout<<"yc = \t";
    cin>>ys1;
    cout<<"Enter the X Radius\t";
    cin>>rx1;
    cout<<"Enter the Y Radius\t";
    cin>>ry1;
    ellips1(xs1,ys1,rx1,ry1);
    getch();
    closegraph();
}
```

Output

```
Midpoint Ellipse Drawing Algorithm
Enter the Center Co-ordinates
xc = 250
yc = 200
Enter the X Radius 100
Enter the Y Radius  50
```

Example 2.6.1

Plot an ellipse having $r_y = 4$, $r_x = 3$ and whose center co-ordinates are $(x_c, y_c) = (0, 0)$.

Solution :

Given : $r_y = 4$, $r_x = 3$

Plot 1st point as (x, y) where x = 0 and y = ry i.e. (x, y) = (0, 4)

Then initial decision parameter is calculated by,

$$dp\text{-}1 = r_y^2 - r_x^2\left(-r_y + \frac{1}{4}\right) = (4)^2 - (3)^2\left(-(4) + \frac{1}{4}\right)$$

$$dp\text{-}1 = -\frac{71}{4}$$

Then we have to check whether we have crossed region-1 or not.

$$\therefore \quad 2 r_y^2 x < 2 r_x^2 y$$

$$2 \cdot (4)^2 \cdot (0) < 2 \cdot (3)^2 \cdot (4)$$

$$0 < 72$$

As $(2 r_y^2 x)$ is smaller than $(2 r_x^2 y)$ it means we are in region-1.

$\therefore$ check for decision parameter, if it is negative then increase x only and update decision parameter.

Here $dp\text{-}1 = -\frac{71}{4}$.

$\therefore$ Increase x by 1, But don't change y and updating dp-1 gives.

$$dp\text{-}1 = dp\text{-}1 + r_y^2 (2 x_p + 3) = \frac{249}{4}$$

So plot new (x, y) as (x = 1, y = 4)

Every time we have to check whether we have crossed the region or not. If yes, then find out the new initial value of region-2 and proceed along y-axis till y becomes zero.

At the end, i.e. when y become 0 we will get following table of x and yco-ordinates.

At last replicate these points to rest of the quadrants to get full ellipse as shown in Fig.P.2.6.1.

x	y
0	4
1	4
2	3
3	2
3	1
3	0

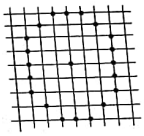


Fig. P. 2.6.1

2.7 Arcs and Sectors

Arc

- An arc may be generated by using either the polynomial or trigonometric method. When *polynomial method* is used, the value of x is varied from x_1 to x_2, and the values of y are found by evaluating the expression $y = \sqrt{r^2 - x^2}$.
- When *trigonometric method* is used, we have to provide starting angle θ_1 and ending angle θ_2. The rest of the steps are similar to circle generation except symmetry. As we want to draw an arc we should not replicate the point in other quadrants. See Fig. 2.7.1.
- Arcs are nothing but a portion of circles however problem occurs only in Bresenham's circle algorithm, because we must provide starting and ending points also for that. When you are calculating next point every time you have to compare that new point with end point and if end point is in between step, then your program goes in infinite loop.

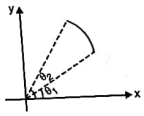


Fig. 2.7.1

Sectors

- Sector is scan converted by using any method of scan-converting an arc and then scan converting two lines from the center of the arc to the endpoints of the arc. Refer Fig. 2.7.2. e.g. say center is (h, k) and we want to scan convert an arc from θ_1 to θ_2.
- Then a line would be scan-converted from (h, k) to $(r \cos(\theta_1) + h, r \sin(\theta_1) + k)$. A second line would be scan-converted from :

  (h, k) to $(r \cos(\theta_2) + h, r \sin(\theta_2) + k)$.

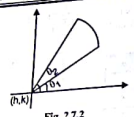


Fig. 2.7.2

Syllabus Topic : Aliasing, Anti-aliasing Techniques like Pre and Post Filtering, Super Sampling and Pixel Phasing

2.8 Aliasing and Anti-aliasing

→ (May 2013, May 2015, Dec. 2016)

Q. What is aliasing ? Explain some antialiasing techniques. **MU - May 2013. 5 Marks**

Q. What are aliasing and antialiasing ? Explain any one antialiasing method. **MU - May 2015. 5 Marks**

Q. What is aliasing ? Explain any two antialiasing techniques. **MU - Dec. 2016. 5 Marks**

Q. Explain Antialiasing Techniques. **(5 Marks)**

- The various forms of distortion that results from the scan conversion operations are collectively called as **aliasing**.

Staircase

A common example of aliasing effects is the staircase or jagged appearance. We see this effect when scan converting a primitive such as line or circle. We also see the stair steps or jaggies along the border of filled region. Specifically when the resolution of the display is poor or low and if the display allows only two pixel states ON or OFF in that case this effect can be noticed very clearly. See Fig. 2.8.1.

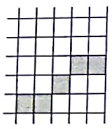


Fig. 2.8.1

Unequal brightness

In this case a slanted line appears dimmer than a horizontal or vertical line, although all are presented at the same intensity level. The reason for this problem is the pixels on the horizontal line are placed one unit apart where as those on diagonal line are approximately 1.414 units apart.

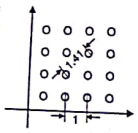


Fig. 2.8.2

This difference in distance produces the difference in brightness. Refer Fig. 2.8.2.

Picket fence problem

- Picket fence problem occurs when an object is not aligned with the pixel grid properly. See Fig. 2.8.3.
- Here the distance between two adjacent pickets is not a multiple of unit distance between pixels.

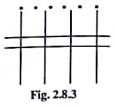


Fig. 2.8.3

- When we scan convert it into image it will result in uneven distance between pickets; since the end point of picket must be matched with pixel co-ordinates. See Fig. 2.8.4. This is sometimes called *global aliasing*, as the overall length of the picket is approximately correct.

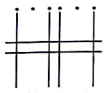


Fig. 2.8.4

- And to maintain the proper/equal length spacing sometimes it will distort the overall length of fence. This is sometimes called as *local aliasing*. See Fig. 2.8.5.

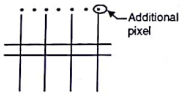


Fig. 2.8.5

Antialiasing

- Most aliasing effects, occurs in static images at a moderate resolution, are often tolerable or negligible. But when the image is moving or in animation this drawback becomes bold. We can come out of this drawback by straight way increasing the image resolution.
- But for that we have to pay extra money and still the results are not always satisfactorily. There are techniques that can greatly reduce this effect and improve the images. Such techniques are collectively known as antialiasing.
- When we have displays that allow *more than two colors*, we can use the following techniques to soften the jaggies.

1. Prefiltering

- This is a technique that determines pixel intensity based on the amount of that particular pixels coverage by the object in the scene i.e. It computes pixel colors depending on an objects coverage. It means how much part or fraction of that pixel is covered by the object and depending on that, it sets the value of that pixel. It requires large number of calculations and approximations. Prefiltering generates more accurate antialiasing effect. But due to its high complexity of calculations it is not used.

2. Supersampling

- Supersampling tries to reduce the aliasing effect by sampling or taking more than one sample per pixel. It subdivides each pixel into 9 sub pixels of equal size which is often called as sampling points. See Fig. 2.8.6. Some of these 9 sub pixels may get color of background i.e. a line may not pass through them and rest of sub pixels may get color of object.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Fig. 2.8.6

- Suppose 3 samples or sub pixels get background color and 6 gets foreground color then the color of the pixel will be the sum of $\left(\frac{1}{3}\right)$ of background color and $\left(\frac{2}{3}\right)$ of object color. So ultimately the pixel value becomes the average of several samples.

3. Postfiltering

1/8	1/8	1/8
1/8	50%	1/8
1/8	1/8	1/8

Fig. 2.8.7

- Postfiltering and supersampling are almost same. If we consider 9 sampling points as in supersampling, we may give a lot more weightage to center point.We could give the center sample as weightage of 50% or 1/2; and remaining 50% is distributed among the rest of 8 samples. See Fig. 2.8.7.
- So we can treat supersampling as the special case of postfiltering in which each sampling point has an equal weightage $\left(\frac{1}{9}\right)$.

4. Pixel phasing

- Pixel phasing is a hardware based anti-aliasing technique. The graphics system in this case is capable of shifting individual pixels from their normal positions in the pixel grid by a fraction (typically $\frac{1}{4}$ or $\frac{1}{2}$) of unit distance between pixels.
- By moving pixels closer to the true line, this technique reduces the aliasing effect.

5. Gray level

- Many displays allow only two pixel states, ON and OFF. In that case we observe stair step aliasing effect. To reduce this effect we can make use of gray level technique.
- In this technique display allow setting pixels to gray levels between black and white to reduce the aliasing effect.
- See Fig. 2.8.8. It uses the gray levels to gradually turn off the pixels in one row as it gradually turns on the pixels in the next.

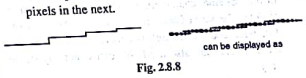


Fig. 2.8.8

2.9 Character Generation

Q. Explain Bit-Map character generation method. **(5 Marks)**

Q. Explain character Generation methods. **(5 Marks)**

- Usually characters are generated by hardware. But we can generate/prepare characters by software also. There are three primary methods for character generation. First is called as *stroke method* or *vector character* generation and the second is called as dot-matrix or *bitmap* method and third is called as *starbust method.*

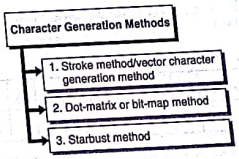


Fig. C2.2 : Character Generation

➜ **(1) Stroke method/vector character generation method**

- This method creates characters by using a set of line segments.
- We could build our own stroke method by vector generation algorithm or by using any line generation method.
- To produce a character we will give a sequence of commands that defines the start point and end points of the straight lines.
- By using this we can change the scale of the characters. We can make a character twice as large as its original size. Similarly we can get characters slanted also. By using this method we can change the style of characters also. Different character styles are shown in Fig. 2.9.1.

Fig. 2.9.1

➜ **(2) Dot-matrix or bit-map method**

- In this scheme, characters are represented by an array of dots. The size of this array may vary. An array of 5 dots wide and 7 dots high is generally used, but 7 × 9 and 9 × 13/14 are also used. We can select any size of array.

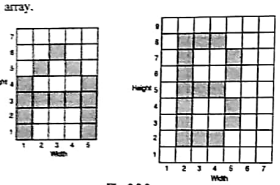


Fig. 2.9.2

- This array is like a small buffer, just big enough to hold a single character.
- The dots are the pixels of the small array.
- Placing the character on the screen then becomes a matter of copying pixel values from small character array into some portion of the screens frame buffer. See Fig. 2.9.2.
- A bitmap font uses a rectangular pattern of pixels to define each character.
- The entire font can be loaded into an area of memory called font cachedisplaying character means then copying characters image from font cache into the frame buffer at the desired position.
- Bitmap fonts require more space, because each variation (size or format) must be stored.

➜ **(3) Starburst method**

- In this method a fix pattern of line segments are used to generate characters. There are 24 line segments and out of these 24 line segments, segments required to display for particular character are highlighted.
- This method of character generation is called Starbust method because of its characteristic appearance.
- The pattern for particular character is stored in the form of 24 bit code. Each bit representing one line segment. The bit is set to one to highlight the line segment. Otherwise it is set to zero.

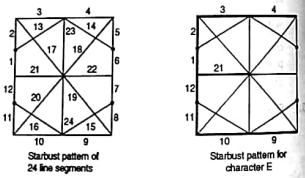


Fig. 2.9.3

24 bit code for character E is,

Bit number

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

But its main disadvantage is, it requires more memory and requires additional code conversion programs to display characters from the 24 bit code.

☞ **Program for bitmap character generation**

A 'C' Program to generate the characters using Bit Map method

```
#include<stdio.h>
#include< conio.h>
#include<graphics.h>
Void main()
{
   int gd, gm, a[7][5] = {0}, i, j;
   gd= DETECT;
   initgraph(&gd, &gm, "C:/TC/BGI");
   clrscr();
   printf("Enter the array : ");
   for(i=0;i<7;i++)
   {   for(j=0;j<5;j++)
       Scanf("%d", &a[i][j]);
   }
printf("The entered character is :\n ");
for(i=0;i<7;i++)
   {   for(j=0;j<5;j++)
       If(a[i][j]==1)
       printf(" %d ", a[i][j]);
   else
       printf(" 0 ");
       printf(" \n ");
   }
```

Output

Enter the array

```
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
```

The entered character is :

```
    1
  1   1
1       1
1 1 1 1 1
1       1
1       1
1       1
```

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words, and for that we are providing guidelines to the that answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 Explain DDA line generation algorithm.

Ans. :

The answer for this question should include following points :

- Gentle slope and steep slope.
- How to find next pixel from the current point.
- Ceiling and flooring functions.
- Steps for DDA line generation algorithm with diagram of all cases such as gentle slope and steep slope lines.
- State advantages and disadvantages of algorithm.
- Take a suitable example and explain it by steps of algorithm.
- Draw the necessary diagrams whenever possible.

Q. 2 Explain Bresnham line generation algorithm.

Ans. :

The answer for this question should include following points :

- Gentle and steep slope
- If possible, derivation of decision parameter.
  i.e. $G_{new} = G_{old} + 2Dy$ and
  $G_{new} = G_{old} + 2Dy - 2Dx$
- Steps for Bresenham line drawing algorithm.
- Draw the diagrams wherever possible.
- Take an example and explain it by algorithm.
- State its advantages and disadvantages.

Q. 3 Explain Bresenham or midpoint circle generation algorithm.

Ans. : To answer this question we should include following points :

- Definition and equation of circle.
- Symmetry property of circle with diagram.
- Calculation of decision parameter for circle with diagrams.
- Derivation of initial decision parameter with diagram.
- Steps for circle generation algorithm.
- Explain the algorithm with suitable example and diagram.

Q. 4 Explain midpoint ellipse generation algorithm.

Ans. :

The answer for this question should include following points :

- Definition and equation of ellipse.
- Explanation of region1 and region2 and symmetry property.
- Calculation of decision parameters with diagram.
- Derivation of initial decision parameter for region1 and region2.
- Steps for ellipse generation algorithm.
- Explain the algorithm with suitable example and diagram.

Q. 5 Explain aliasing and antialiasing techniques.

Ans. :

For this we have to include following points :

- Definition of aliasing and antialiasing.
- List aliasing and antialiasing techniques and explain them in 3-4 lines with diagrams.

Q. 6 Compare DDA line algorithm with Bresenham's line algorithm.

Ans. :

Comparison between DDA and Bresenham line drawing algorithms

Sr. No.	DDA	Bresenham
1.	Based on increment method.	Based on increment method.
2.	Use floating point arithmetic.	Use only integers.
3.	Slower than Bresenham	Faster than DDA
4.	Use of multiplication and division operations.	Use of only Addition and Subtraction operations.
5.	To display pixel we need to use either floor or ceil function.	No need of floor or ceil function for display.
6.	Because of floor and ceil function error component is introduced.	No error component is introduced.
7.	The co-ordinate location is same as that of Bresenham.	The co-ordinate location is same as that of DDA.

Q. 7 Write a short note on any one character generation method.

Ans. : There are three methods for character generation. We have to explain any one of them. They are :

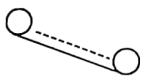
Stroke method Bit-map method Starbust method

To explain any one method of above we should draw proper diagram.

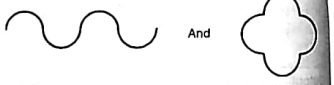
Important Laboratory Assignments

This chapter is important from practical examination point of view. We have already explained and given, enough number of programs at the time of explaining algorithms. In addition to these programs, in final practical examination, few modifications are also expected. The list of such programs is given below :

1. Write a program to draw a temple by using DDA line generation algorithm.
2. Write a program to draw house by using bresenhams line generation algorithm.
3. Write a program to draw a car by using bresenhams circle and line algorithm.
4. Write a program to slide the circle on slanted ramp. Accept the circle and the line for ramp from user.



5. Write a program to generate following shapes by using any circle generation algorithm.



6. Write a program to generate a spiral circle.

Above mentioned programs are very simple to implement only few modifications are required to normal algorithms. Students are advised to make practice of these assignments.

Exercise

Q. 1 Write the DDA line drawing algorithm. Calculate the pixel co-ordinates of line AB using DDA algorithm. Where A = (0,0) and B = (4,6). **(10 Marks)**

Ans. :

Digital differential analyzer (DDA) is based on incremental method. The slope intercept equation for a straight line is,

$$y = m \cdot x + b$$

Where m is slope and b is y-intercept. We can determine the value of m as,

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

Steps for DDA algorithm

(1) Accept two end points : (x_a, y_a) and (x_b, y_b)

(2) Find out horizontal and vertical difference between end points.

$Dx = x_b - x_a$ and $Dy = y_b - y_a$

(3) Difference between greater magnitude determines the value of parameter steps.

(4) Determine the offset needed at each step i.e. to generate the next pixel loop through this step times.

(5) If $|Dx| > |Dy|$ and $x_a < x_b$,

Then the values of the increments in the x and y-directions are 1 and m respectively. See Fig. 1.

$x_{increment} = 1$

$y_{increment} = m$

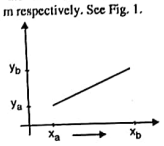


Fig. 1

(6) If $|Dx| > |Dy|$ and $x_a > x_b$,

Then the decrements –1 and –m $x_{increment} = -1$

are used to generate new point. $y_{increment} = -m$

See Fig. 1(a).

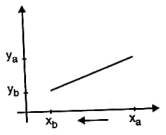
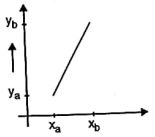


Fig. 1(a)

(7) Similarly if $|Dx| < |Dy|$ and $y_a < y_b$

Then the values of increments in x and y-direction are 1/m and 1 respectively.

Refer Fig. 1(b).



$x_{increment} = \frac{1}{m}$

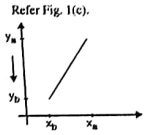
$y_{increment} = 1$

Fig. 1(b)

(8) Similarly if $|Dx| < |Dy|$ and $y_a > y_b$

Then the values of increments in x and y-direction are $-\frac{1}{m}$ and – 1.

Refer Fig. 1(c).



$x_{increment} = -\frac{1}{m}$

$y_{increment} = -1$

Fig. 1(c)

Example

Pixel calculation for a Line AB.

Suppose if we want to draw a line from (0,0) to (4,6).

So, $x_1 = 0$, $y_1 = 0$, $x_2 = 4$ and $y_2 = 6$.

$\therefore Dx = (x_2 - x_1) = (4 - 0) = 4$

$Dy = (y_2 - y_1) = (6 - 0) = 6$

As $|Dx| < |Dy|$ the line is of Steep slope category.

$\therefore$ Steps = abs (Dy) = 6

$$x_{increment} = \frac{Dx}{steps} = 4/6 = 0.6$$

$$y_{increment} = \frac{Dy}{steps} = 6/6 = 1$$

First point we know i.e. x_1, y_1 so plot it.

$\therefore x_{new} = x_{old} + x_{increment} = 0 + 0.6 = 0.6$

$y_{new} = y_{old} + y_{increment} = 0 + 1 = 1$

But over here we have to round off 0.6 as 1 for displaying that point.

For next iteration

$x_{new} = x_{old} + x_{increment} = 0.6 + 0.6 = 1.2$

$y_{new} = y_{old} + y_{increment}$

$= 1 + 1 = 2$

Now loop steps are tabulated as follows :

i	x	y	Plot
1	0	0	0, 0
2	0.6≈1	1	1, 1
3	1.2≈1	2	1, 2
4	1.8≈2	3	2, 3
5	2.4≈2	4	2, 4
6	3	5	3, 5
7	3.6≈4	6	4, 6

For calculation purpose we are using original value but for display only, we are rounding the values. If we do this till x_{new} and y_{new} becomes same as end point then we will get the desired line.

Q. 2 Write Bresenham's line drawing algorithm. Calculate the pixel co-ordinates of line PQ using Bresenham's Algorithm, Where P1(20,20) and Q = (10,12) **(10 Marks)**

Ans. :

Now, let us calculate the pixel co-ordinates of line PQ.

Given : $P(x_1, y_1) = (20, 20)$

$Q(x_2, y_2) = (10, 12)$

$x_1 = 20;$

$y_1 = 20$

$\therefore \quad x_2 = 10;$

$y_2 = 12$

$\therefore \quad Dx = x_2 - x_1 = 10 - 20 = -10$

$Dy = y_2 - y_1 = 12 - 20 = -8$

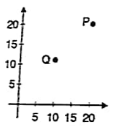


Fig. 2

Since, Dx and Dy are negative we will take their absolute values.

$\therefore \quad Dx = 10$

and $\quad Dy = 8$

Now, let's find decision factor G,

$G = 2Dy - Dx = 2(8) - 10 = 06$

$\therefore \quad G = 6$

Here, since Dx > Dy the line is of gentle slope category so we need to move along x axis and find corresponding y value for each x till x becomes equal to 20.

First plot point (10, 12),

Now increase x by 1

$\therefore \quad x = 11$

Here G = 6 i.e. positive so increase y by 1 and update G

As, $G = G + 2(Dy - Dx)$

$= 6 + 2(8 - 10) = 6 - 4 = 2$

Now plot x = 11, y = 13,

Now, $G = 2$

Again G is psitive so again increase y by 1 and update G as,

$G = G + 2(Dy - Dx) = 2 + 2(8 - 10)$

$= 2 - 4 = -2$

Now plot x = 12, y = 14

Now since G = – 2 i.e. Negative.

Next time don't increase y and update G by,

$G = G + 2(Dy) = -2 + 2(8)$

$= -2 + 16 = 14$

Now, plot (x = 13, y = 14)

Like this we will get following points.

X	Y
10	12
11	13
12	14
13	14
14	15
15	16
16	17
17	18
18	18
19	19
20	20

Q. 3 Specify midpoint circle algorithm. Using the same, plot the circle whose radius is 10units. **(May 2014, Dec.2015, May 2016, 10 Marks)**

Ans. :

Given : Since center of the circle is not specified we will assume it as origin.

So the given things will be $r = 10, x_c = 0, y_c = 0$

Plot 1st point as (0, r) i.e. y = r

$\therefore$ Plot (0, 10)

Like that find out other points from (0, 10) by using symmetry property

So plot (10, 0)

plot (0, – 10)

plot (– 10, 0)

find $P = 1 - r \quad \therefore P = 1 - 10 = -9$

Till (x < y) we have to perform following

If (P < 0)

In this case P = – 9

So we have to increase x by 1

and

modify P as $P = P + 2x + 1$

i.e. $P = -9 + 2(0) + 1 = -9 + 1 = -8$

Here we are not updating y.

$\therefore$ Plot (x + 1, y) i.e. (1, 10)

Continue this process till (x = y) i.e. till angle 45, where x and y becomes equal. Then plot the co-ordinates of this set by using either octant symmetry property. The following table shows co-ordinates of the first octant.

x	y
0	10
1	10
2	10
3	9
4	9
5	8
6	8
7	7

2.10 Exam Pack (University and Review Questions)

Syllabus Topic : Scan Conversions of Point, Line, Circle and Ellipse – DDA Algorithm and Bresenham Algorithm for Line Drawing

Q. What are the disadvantages of DDA algorithm ? *(Refer Section 2.3.1)* **(May 2015, May 2017, 5 Marks)**

Q. Specify the disadvantages of DDA algorithm. *(Refer Section 2.3.1)* **(May 2016, 5 Marks)**

Q. Derive Bressenham's line drawing algorithm for lines with slope. *(Refer Section 2.3.2)* **(Dec. 2014, 10 Marks)**

Q. Derive Bresenham's line drawing algorithm for lines with slop < 1. *(Refer Section 2.3.2)* **(Dec. 2015, 5 Marks)**

Q. Write Bresenham's line drawing algorithm. Also write mathematical derivations for the same. *(Refer Section 2.3.2)* **(10 Marks)**

Q. Explain the method to draw a thick line using Bresenham's algorithm. *(Refer Section 2.4)* **(May 2013, 5 Marks)**

Syllabus Topic : Midpoint Algorithm for Circle

Q. Specify midpoint circle algorithm. *(Refer Section 2.5.1)* **(May 2014, 5 Marks)**

Q. Explain midpoint circle algorithm. Use the same to plot the circle, whose radius is 10 units. *(Refer Section 2.5.1)* **(Dec. 2014, Dec. 2015, May 2016, Dec. 2016, 10 Marks)**

Q. Explain the midpoint circle generating algorithm. *(Refer Section 2.5.1)* **(May 2015, 8 Marks)**

Q. Explain mid-point circle algorithm. In order to support your explanation, show mathematical derivation. *(Refer Section 2.5.1)* **(8 Marks)**

Q. Derive Mid point circle algorithm. *(Refer Section 2.5.1)* **(10 Marks)**

Q. Explain Bresenham's Circle drawing algorithm in detail. *(Refer Section 2.5.2)* **(May 2017, 10 Marks)**

Syllabus Topic : Midpoint Algorithm for Ellipse Drawing

Q. Derive the midpoint algorithm for ellipse generation. *(Refer Section 2.6)* **(May 2013, 10 Marks)**

Q. Explain Midpoint ellipse algorithm with all required mathematical representation. *(Refer Section 2.6)* **(12 Marks)**

Q. Derive decision parameters for the midpoint ellipse algorithm (Region 1), assuming the starting position is $(0, r_y)$ and points are to be generated along the curve path in clockwise order. *(Refer Section 2.6)* **(10 Marks)**

Syllabus Topic : Aliasing, Anti-aliasing Techniques like Pre and Post Filtering, Super Sampling and Pixel Phasing

Q. What is aliasing ? Explain some antialiasing techniques. *(Refer Section 2.8)* **(May 2013, 5 Marks)**

Q. What are aliasing and antialiasing ? Explain any one antialiasing method. *(Refer Section 2.8)* **(May 2015, 5 Marks)**

Q. What is aliasing ? Explain any two antialiasing techniques. *(Refer Section 2.8)* **(Dec. 2016, 5 Marks)**

Q. Explain Antialiasing Techniques. *(Refer Section 2.8)* **(5 Marks)**

Q. Explain Bit-Map character generation method. *(Refer Section 2.9)* **(5 Marks)**

Q. Explain character Generation methods. *(Refer Section 2.9)* **(5 Marks)**

□□□

Module 2

CHAPTER 3

Filled Area Primitives

Section Nos.	Name of the Topic
3.1	Introduction
3.2	Inside/Outside Test of Polygons
3.3	Polygon Filling
3.4	Scan Line Filling

Types of polygons, interior test of a point and filling of polygons by various ways, forms the subject matter of chapter 3. There are four major sections :

3.1 **Introduction** explains different types of polygons and how to represent the polygon.

3.2 **Inside/Outside test of polygon** explains even-odd test and winding number test to determine whether the given point is lying inside the polygon or not.

3.3 **Polygon filling** normally occurs once the polygon is defined. It is filled with different techniques which is referred as pixel level polygon filling algorithms. We explain these algorithm in this section.

3.4 **Scan line filling** gives an idea about geometric level filling algorithm. Here we explain scan line fill algorithm in details.

3.1 Introduction

- Till now we have discussed about the line, circle and ellipse, but many times we need to display solid objects also. These solid objects are generally referred as a new graphic primitive, the *polygon*.
- We will discuss different types of polygons and representation of them. We will also learn how to determine if a point is inside a polygon and finally we will see various methods for filling a polygon.

3.1.1 Polygon

- Polygon is a figure having many sides. It may be represented as a number of line segments connected end to end to form a closed figure. The line segments which form the boundary of polygon are called as edges or sides of polygon. The end points of the sides are called the polygon vertices.

Fig. 3.1.1

- Triangle is the most simple form of polygon having three sides and three vertices. The polygon may be of any shape. (See Fig. 3.1.1). To form a polygon we need minimum three vertices and three edges and that too must form a close figure.
- In Fig. 3.1.2 there are two sides and three vertices but as it is not forming close figure, we cannot call this figure as a polygon. Polygon must be a close figure.

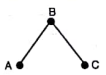


Fig. 3.1.2

- **Polygon** is defined as a figure having number of vertices and edges in such a way that they are forming a close loop. Circle is forming a close loop. But the number of vertices and edges joining these vertices are also important. In circle we are having pixels on peripheral of circle. There are no edges in circle. Therefore we can say that circle is not a polygon.

Types of Polygon

We can divide the polygons into three types :

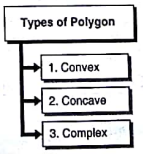


Fig. C3.1 : Types of Polygon

1. Convex

- Convex polygon is a polygon in which if we take any two points which are surely inside the polygon and if we draw a line joining these two points and if all the points on that line lies inside the polygon, then such a polygon is called as ***convex polygon***.
- Examples of convex polygons are as shown in Fig. 3.1.3.

Fig. 3.1.3

2. Concave

- Concave polygon is a polygon in which if we take any two points which are surely inside the polygon and if we draw a line joining these two points and if all the points on that line are not lying inside the polygon, then such a polygon is called as ***concave polygon***. Generally the polygons which are not convex are referred as concave polygons.
- Concave polygons are as shown in Fig. 3.1.4.

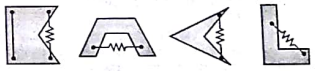


Fig. 3.1.4

3. Complex

- In computer graphics, a polygon which is neither convex nor concave is referred as complex polygons. The complex polygon includes any polygon which intersects itself or the polygons which overlaps itself. The complex polygon has a boundary. Complex polygons are as shown in Fig. 3.1.5.

Fig. 3.1.5

- The complex polygons may not be always interpreted as a normal polygon. In this type of polygons the vertices are counted only at the End of the edges and not at intersection points of the overlapping edges.

3.1.2 Polygon Representation

- When we are going to use the polygons in our graphics system, first we have to decide how to represent the polygons. Some graphics devices give permission to directly image polygon shapes. In that case, a polygon acts as a whole unit. Some graphics devices may provide trapezoid primitives.
- Trapezoid primitive means polygons are imaged or drawn in the form of trapezoids. Trapezoids are formed by using two scan lines and two line segments. Polygon shown in Fig. 3.1.6 is imaged as a set of trapezoids as shown in Fig. 3.1.7.

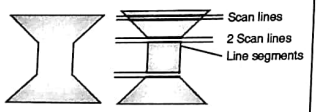


Fig. 3.1.6 Fig. 3.1.7

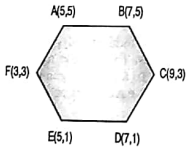
- Some graphics devices may not provide support to polygons at all. In that case we have to break up the polygon into lines and points which can be imaged or drawn to form polygon. So we should store full polygon in our display file and then try to investigate or find the methods for imaging or displaying them using lines or points.
- For this we have to store the polygon in display file. Display file contains the information necessary to construct the picture. The information will be in the form of commands, such as *move* or *line*. The structure of display file is 3 arrays, one for opcode. i.e. command and other two to store x and y values.
- If we place line commands only in opcode array of display file for representing the edges of polygon then there may be two difficulties. First, we will not get exact idea of how many sides the polygon is having or which set of commands we have to group.

OP	x	y
2	–	–
2	–	–
2	–	–
2	–	–
2	–	–
2	–	–

Fig. 3.1.8

Suppose the structure of display file is as shown in Fig. 3.1.8. Where, each command represent edge, but if our polygon is created only of first 5 commands or having 5 edges, then in that case there is no breaking system to come out of loop. As opcode 2 means draw line we will draw all lines.

- Second difficulty is what should be the starting point or from where we have to start drawing polygon. i.e. starting position on display.
- We can overcome this problem by prefacing the commands by new commands. This new command will tell us how many sides are there for the polygon and from that we will come to know number of sides the polygon is having. We can interpret this new command as starting point code to draw first side also.
- Until now we have used only 2 opcodes for display file and those are opcode '1' for *move* command and '2' for *line* command. So we can use a number greater than 2 to indicate the sides of polygon, because minimum sides of polygon are three, so when there is a number which is other than 1 or 2 in opcode field it is treated as count of polygon edge and we have to draw that many lines or we have to execute that many further commands for that polygon. The x and y operands of this command i.e. the command whose opcode field is other than 1 or 2, will be considered as starting points co-ordinates.



OP	x	y
6	5	5
2	7	5
2	9	3
2	7	1
2	5	1
2	3	3
2	5	5

Fig. 3.1.9

- In Fig. 3.1.9 the polygon is formed by 6 vertices and 6 edges. So we will put 6 in opcode field to indicate that we have to execute next 6 commands only, from the display file. The x and y field of command whose opcode is 6 will represent the starting point for the polygon. So the cursor will be placed to the position (5, 5). Then the next command will be executed which is (2,7,5) i.e. 2 means draw the line from current position to specified location (7,5). Like that we are executing six instructions, means up to instruction (2,5,5).

Syllabus Topic : Inside Outside Tests

3.2 Inside/Outside Test of Polygons

➔ (Dec. 2016)

Q. What is a purpose of inside outside test? Explain with example. MU - Dec. 2016, 5 Marks

Q. Illustrate inside-outside tests with example. (5 Marks)

- Can we determine whether a point is inside the polygon or outside ? Yes, to determine this we are generally using two methods :

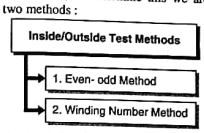


Fig. C3.2 : Inside/Outside test of Polygons

➔ 3.2.1 Even - odd Method

Q. Explain even-odd test to determine whether the point is inside or outside of polygon. (5 Marks)

- One method of doing this is to construct a line segment between a point in question i.e. point to test whether inside or outside, and a point which is surely outside the polygon. But how we are going to find out a point which is surely outside the polygon? It is very easy to find out that point.
- For example, pick a point with an x co-ordinate smaller than the smallest x co-ordinate of the polygons vertices and the y co-ordinate will be any y value, or for simplicity we will take y value same as the y value of point in question. Refer Fig. 3.2.1.
- In this case point A is one, which we want to check i.e. whether point A is inside polygon or not.
- As we are using arrays to store vertices of polygon we can easily come to know the vertex which is having lowest value of x and that is x_1. So we have to select a point smaller than x_1 and generally we select same value of y as that of point A, in order to draw straight line. But even if you select any y value for outside point, it will not make any difference.

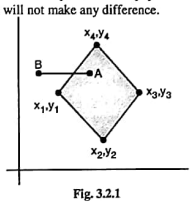


Fig. 3.2.1

- Then count how many intersections are occurring with polygon boundary by this line till the point in question i.e. 'A', is reached. If there are an odd number of intersections then the point in question is inside. If even number of intersections then the point is outside the polygon.
- This is called *Even-odd method* to determine interior points of polygon.
- But this even-odd test fails for one case i.e. when the intersection point is a vertex. See Fig. 3.2.2. To handle this case we have to make few modifications. We must look at other end points of the two segments of a polygon which meet at this vertex.
- If these points lie on the same side of the constructed line AB, then the intersection point counts as an even number of intersection i.e. (2, 4, ... etc.). See Fig. 3.2.2.

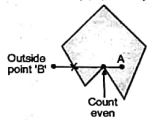


Fig. 3.2.2

- But if they lie on opposite sides of the constructed line AB, as shown in Fig. 3.2.3, then the intersection point is counted as a single intersection.
- Let's take an example which is shown in Fig. 3.2.4(a)

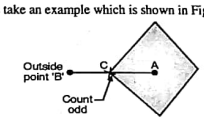


Fig. 3.2.3

- In this case the constructed line AB intersects polygon in a vertex 'C'. Here the other end points of both the line segments which meets at vertex 'C' are lieing in opposite sides of this constructed line AB. So the intersection at vertex 'C' is counted as odd count i.e. 1. The total number of intersections made by this new line with polygon is 1 i.e. odd and as it is odd therefore the point 'A' is inside.
- Similarly for case II refer Fig. 3.2.4(b).
- Here we want to check point A. So we are taking one point B which is surely outside and drawing a line between A to B. Here this new line intersects in two points i.e. C and D. Point C is not a vertex point so it is counted as normal intersection count. But point D is a vertex. So we have to check the other end points of both the line segments which meets at vertex D. These end points are on one side of constructed line AB. So we have to count this intersection point as even count, say as 2.

(a) (b)

Fig. 3.2.4

- Here as the constructed line intersects the polygon in two points C and D. So we have to make sum of this i.e. for C we are having count as 1 and for D we are having even count i.e. 2.
- So the sum of this count will be 1 + 2 i.e. 3, which is odd. So the point A is inside. If this sum is even then the point will be outside.

→ 3.2.2 Winding Number Method

- The other alternative method for defining a polygons interior point is called the *winding number method*. Consider a piece of elastic between point of question (A) and a point on polygon boundary. Treat that, elastic is tied to point of question firmly and the other end of elastic is sliding along the boundary of the polygon. Until it has made one complete circuit. (see Fig. 3.2.5.)

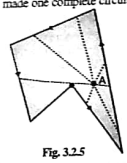


Fig. 3.2.5

- Then we check how many times the elastic has been wound around the point of question (A). If it is wound at least once then the point is inside. If no net winding then point is outside.
- To explain in more simple words, we begin with our even-odd method. We draw a line between point of question and outside point. Then consider the edges or sides of polygons where this line crosses. In even-odd method we just counts the number of intersections. But in winding number method we give direction numbers to each boundary line which is crossed by this line and we sum these direction numbers. Direction number indicates the direction in which polygon edges are drawn. (See Fig. 3.2.6)
- The side which starts at above the drawn line and crosses line and then ends below the line. In that case we assign that side, direction number as 1.

  If the edge of polygon starts below the drawn line, crosses line and then ends above the line, we give –1 to direction number.

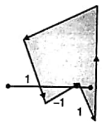


Fig. 3.2.6

- Then find the sum of these numbers. If it is nonzero then the point is inside. If sum is zero then point is outside. Let us take an example.

Case (I) : Refer Fig. 3.2.7.

- Here point in question (A) and the point which is surely outside, i.e. B, is connected by a new line and we got the intersection point as C.
- Now the edge of polygon which is getting intersected by this line AB is starting from above this constructed line and ends below the constructed line.

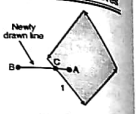


Fig. 3.2.7

- So we have to give count for this edge as 1. As constructed line intersects only in one point there is no need to find the sum of intersection points. The total is only 1. i.e. nonzero. As the result is nonzero, so the point in question (A) is inside.

Case (II) : Refer Fig. 3.2.8.

- Here there are two intersections C and D. We want to find whether A is inside or not. For intersection point C the count will be + 1 and for D it will be – 1. So the sum will be + 1 – 1 = 0. As the result is zero, so the point A is outside the polygon.

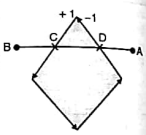


Fig. 3.2.8

Special cases

- But the intersection point could be one of the vertex of the polygon as shown in following three cases of Fig. 3.2.9.

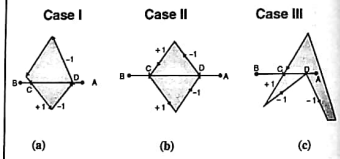


Fig. 3.2.9

- In case 1 and case 2 of Fig. 3.2.9 intersection point is a vertex, so in this situation we have to check the other end points of the line segments which meets at vertex.
- If the other end points are lieing in opposite direction of the newly constructed line then we will take count as any one edge's count. In *case I*, for point C the count is (1) and for D it will be (– 1). Therefore sum will be + 1 – 1 = 0, so the point (A) lies outside.
- Similarly in *case II*, for point C we will take count as (+ 1) and for point D it will be (– 1). So the sum will be equal to zero. Therefore the point (A) lies outside.
- But if the situation is like *case III*, then in that case as the other end points of the line segments which meets at vertex are lieing in same side, so we have to count that vertex intersection as zero. So the count for C will

be + 1 and for D will be zero. Therefore sum will be + 1 + 0 = + 1 i.e. as the result is nonzero, the point A lies inside the polygon.

- Use of even-odd or winding number method will depend on application. But the even-odd and winding number method gives different results for the overlapping polygons. (See Fig. 3.2.10.)

Even odd Winding number method

Fig. 3.2.10

Syllabus Topic : Filled Area Primitive - Scan Line Polygon Fill Algorithm, Boundary Fill and Flood Fill Algorithm

3.3 Polygon Filling

Q. Explain 4-connected and 8-connected methods. **(5 Marks)**

- `Filling is the process of "coloring in " a fixed area or region. Regions may be defined at pixel level or geometric levels. When the regions are defined at pixel level, we are having different algorithms like :

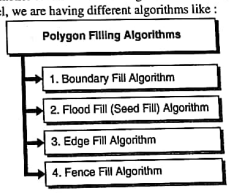


Fig. C3.3 : Polygon filling algorithms

- In case of geometric level we are having scan line fill algorithm.
- We will first see pixel level polygon filling. But before going to actual algorithm, we will see 4-connected and 8-connected pixel concept. These two techniques are the ways in which pixels are considered as connected to each other.
- In 4-connected method the pixels may have upto 4 neighbouring pixels. (See Fig. 3.3.1.). Let's assume that the current pixel is (x, y). From this pixel we can find four neighboring pixels as right, above, left and below of the current pixel.

Fig. 3.3.1

- Similarly in 8-connected method the pixels may have upto 8 neighbouring pixels. (See Fig. 3.3.2.) Here from one pixel location we are finding the neighbouring 8 pixels position.

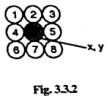


Fig. 3.3.2

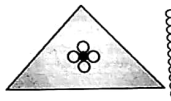


Fig. 3.3.3

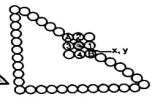


Fig. 3.3.4

- By using any one of these technique we can fill the interior of the polygon. (See Fig. 3.3.3.) But there are some cases where the use of 4-connected method is not efficient.
- Suppose we want to fill a triangle, so we are using 4-connected method. It will work fine but at boundary this method is not efficient. (See Fig. 3.3.4.). Suppose we have filled point (x, y). Now from that point we have to select either point A or B which is not possible by using 4-connected method because in 4-connected method we can go from (x, y) to either 1, 2, 3 or 4 and not A and B. So in this case we have to use 8-connected method where we can choose any one of the neighbouring pixel.

→ 3.3.1 Boundary Fill Algorithm

→ (May 2017)

Q. Write a boundary fill procedure to fill 8-connected region. **MU - May 2017. 5 Marks**

Q. Illustrate one example of polygon filling with diagram where 4-connected approach fails, while 8 connected approaches succeeds. **(7 Marks)**

Q. Write pseudo codes for boundary fill procedure. **(5 Marks)**

Q. Explain boundary fill algorithms. **(5 Marks)**

Q. Write a pseudo-code to implement boundary fill and flood fill algorithm using 4 connected method. **(10 Marks)**

- This algorithm is very simple. It needs one point which is surely inside the polygon. This point is called as "Seed point" which is nothing but a point from which we are starting the filling process. This is a recursive method. The algorithm checks to see if the seed pixel has a boundary pixels color or not.

- If the answer is no, then fill that pixel with color of boundary and make recursive call to itself using each of its neighbouring pixels as new seed. If the pixel color is same as boundary color then return to its caller. (See Fig. 3.3.5.)

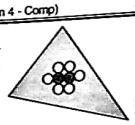


Fig. 3.3.5

- This algorithm works for any shaped polygon and fills that polygon with boundary color. But as it uses high number of recursive calls it takes more time and memory.
- The following procedure illustrates a recursive method for boundary fill by using 4-conneced method.

```
bfill (x, y, newcolor)
{
    current = getpixel (x, y) ;
    if (current != newcolor) && (current != boundary
                                 color)
    {
        putpixel (x, y, newcolor);
        bfill (x + 1, y, newcolor);
        bfill (x – 1, y, newcolor);
        bfill(x, y + 1, newcolor);
        bfill(x, y – 1, newcolor);
    }
}
```

- As we are using recursive function there is a chance that system stack may become full. So we have to develop our own logic to solve the stack problem. For this there are many solutions. One solution is instead of using system stack we can use our own stack and write our own PUSH and POP functions for that stack.
- Second solution could be use of link list i.e. dynamic memory allocation, for implementation of stack. Another solution could be develop some logic while pushing the pixels on stack i.e. before pushing the pixel on stack check whether that pixel is already visited or not.
- If it is already visited there is no point to store that pixel again. Another solution could be instead of using stack we can make use of queue also.
- But the boundary fill algorithm is having limitation. It fills the polygon having unique boundary color. If the polygon is having boundaries with different colours then this algorithm fails.

→ 3.3.2 Flood Fill (Seed Fill) Algorithm

→ (Dec. 2014, May 2015, Dec. 2016)

Q. Explain 4 connected flood fill algorithm in detail. MU - Dec. 2014. 5 Marks

Q. Write short note on : Flood fill algorithm. MU - May 2015, Dec. 2016. 5 Marks

Q. Write pseudo codes for flood fill procedure. (5 Marks)

Q. Explain flood fill algorithms. (5 Marks)

- The limitations of boundary fill algorithms are overcome in flood fill algorithm. Like boundary fill algorithm this algorithm also begins with seed point which must be surely inside the polygon.
- Now instead of checking the boundary color this algorithm checks whether the pixel is having the polygon's original color i.e. previous or old color.
- If yes, then fill that pixel with new color and uses each of the pixels neighbouring pixel as a new seed in a recursive call. If the answer is no i.e. the color of pixel is already changed then return to its caller.
- Sometimes we want to fill an area that is not defined within a single color boundary. (See Fig. 3.3.6.). Here edge AB, BC, CD and DA are having red, blue, green and pink color, respectively.

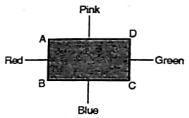


Fig. 3.3.6

- This Fig. 3.3.6 shows an area bordered by several color regions. We can paint such areas by replacing a specified interior color instead of searching for a boundary color value. Here we are setting empty pixel with new color till we get any colored pixel.
- Flood fill and boundary fill algorithms are somewhat similar. A flood fill algorithm is particularly useful when the region or polygon has no uniformed colored boundaries.
- The flood fill algorithm is sometimes also called as **seed fill algorithm** or Forest fire fill algorithm. Because it spreads from a single point i.e. seed point in all the direction.
- The following procedure illustrates a recursive method for flood fill by using 4-connected method.

```
f-fill (x, y, newcolor)
{   current = getpixel (x, y);
    if (current != newcolor)
    {   putpixel (x, y, newcolor);
        f-fill (x – 1, y, newcolor);
        f-fill ( x + 1, y, newcolor);
        f-fill (x, y – 1, newcolor);
        f-fill (x, y + 1, newcolor);
    }
}
```

- The efficiency of this algorithm can be increased in the same way as discussed in boundary fill algorithm.

→ 3.3.3 Edge Fill Algorithm

- In edge fill algorithm the polygon is filled by selecting each edge of the polygon. This algorithm is very simple. The statement of this algorithm is, "for each scan line intersecting a polygon edge at (x_1, y_1) complement all pixels whose mid points lie to the right of (x_1, y_1)". Here the order in which the polygon edges are considered is unimportant.
- Let us try to understand this algorithm by considering an example. Refer Fig. 3.3.7. Suppose we want to fill a polygon ABCD which is shown in Fig. 3.3.7(a). As the statement of algorithm says we have to consider one edge at a time. So let us select edge AB. Now we have to complement all the pixels whose midpoints are lying on right hand side of the edge AB. After performing that task we will get the polygon as shown in Fig. 3.3.7(b). Now we have to select another edge say AD. Here the order of selection of edges is not important. Again complement all the pixels whose midpoints are lying on right hand side of selected edge. We will get the polygon as shown in Fig. 3.3.7(c). Complement means if earlier the pixel is having background color, now we have to make color of that pixel as fill color and if earlier it is fill color, then now we have to make it as background color. Like that we are selecting each and every edge of the polygon and complementing all the pixels which are on right hand side of selected edge till end of the screen.

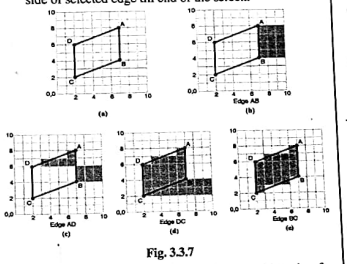


Fig. 3.3.7

- But the main drawback of this algorithm is, for complex pictures each individual pixel is addressed many times. So the algorithm requires more time. One more drawback of this algorithm is even though the polygon is placed in upper left hand corner of the screen, every time we are complementing all the pixels which are on right hand side of the selected edge till the end of the screen. So unnecessarily we are visiting pixels which we are not considering at all.

→ 3.3.4 Fence Fill Algorithm

- In order to reduce the problems of edge fill algorithm, we are using Fence fill algorithm. The word fence means border. The algorithm is somewhat similar to edge fill algorithm. The fence fill algorithm says that for each scan line intersecting a polygon edge,
- If the intersection is to the left of the fence, complement all pixels having a midpoint to the right of the intersection of the scan line and the edge and to the left of the fence.
- If the intersection is to the right of the fence, complement all pixels having a midpoint to the left of the intersection of the scan line and the edge and to the right of the fence.
- Here in this algorithm we are using half scan line; either from edge to fence or from fence to edge only. The location of the fence can be selected anywhere in the polygon. Usually fence location is one of the polygon vertices. Let us consider an example. Refer Fig. 3.3.8.

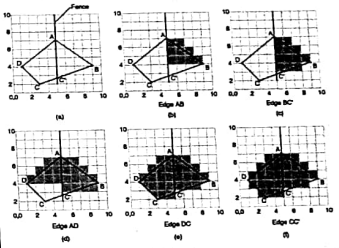


Fig. 3.3.8

- Suppose we have to fill polygon ABCD, so we will take a fence which will pass through any vertex, let us say the fence is passing from A and it cuts edge CB at C′. So our fence will be AC′. Now apply the algorithm to each edge. Let us consider edge AB. Edge AB is on right side of fence AC′ so complement all the pixels which are to the right of fence and to the left of edge AB. See Fig. 3.3.8(b).
- Then select edge BC′. Again this edge BC′ is at right side of fence so complement the pixels which are to the right side of fence and to the left of edge BC′. See Fig. 3.3.8(c). Now select edge AD. This edge is to the left of fence. So we have to complement all the pixels which are to the right of edge AD and to the left of fence. See Fig. 3.3.8(d). Like that we have to do for all the edges of the polygon. At last we will get a filled polygon which is shown in Fig.3.3.8(f).

- This algorithm is faster than the edge fill. But still the disadvantage of both edge fill and fence fill algorithms is, the number of pixels are addressed more than once.

3.4 Scan Line Filling

➔ **(May 2013, May 2014, May 2016)**

Q. Explain with example Scan line fill algorithm. **MU - May 2013, May 2014, 10 Marks**

Q. Compare Boundary fill and flood fill algorithm. **MU - May 2016, 10 Marks**

Q. Explain scan line fill area conversion algorithm with suitable example. **(8 Marks)**

- In contrast to boundary fill and flood fill algorithm at pixel level, this algorithm is defined at geometric level i.e. co-ordinates, edges, vertices etc. This algorithm starts with first scan line and proceeds line by line toward the last scan line and checks whether every pixel on that scan line satisfies our inside point test or not. i.e. it checks which points on that scan line are inside the polygon. This method avoids the need for seed point.
- Let us explain this algorithm by considering an example of convex polygon. (See Fig. 3.4.1.)
- Here the algorithm begins with the first scan line that the polygon occupies i.e. y_{max} and proceeds line by line towards the last scan line i.e. y_{min}.

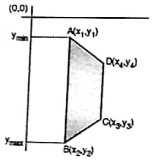


Fig. 3.4.1

- Here we are considering first and last scan line of polygon not individual edges. For each edge of the polygon we are storing five attributes. As we know that to draw any edge or line we need two points. So we are going to store x_{max}, x_{min}, y_{max}, and y_{min} of the edges of the polygon along with their slopes. For edge AB we are going to store

$$x_{max} = x_2$$
$$x_{min} = x_1$$
$$y_{max} = y_2$$
$$y_{min} = y_1$$

- Here x_{max} and x_{min} has as such no meaning, whereas y_{max} is the maximum y value for a particular edge and y_{min} is the minimum y value for that edge. x_{max} and x_{min} are the corresponding x component of y_{max} and y_{min}. Along with this we are going to store the slope of the edge also

Let us tabularize this

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
AD	y_4	x_4	y_1	x_1	m_2
CD	y_3	x_3	y_4	x_4	m_3
BC	y_2	x_2	y_3	x_3	m_4

- For edge AB which is formed by (x_2,y_2) and (x_1,y_1), y_2 is greater than y_1. Therefore $y_{max} = y_2$ and $y_{min} = y_1$ and so $x_{max} = x_2$ and $x_{min} = x_1$.
- As we are filling the polygon line by line, at any particular time we are not considering all the edges. So there is no point to find whether a particular scan line intersects with each edge or not. In fact we have to select only those edges which are getting intersected by the scan line. To decide which edges are getting intersected by scan line we are making use of y_{max} of particular edge. One thing is sure that we are storing all the attributes of all edges. Out of those attributes we are selecting y_{max} and then we are sorting this y_{max} array. If some swapping is required in this y_{max} then we are going to swap whole edge. After sorting y_{max} array we will get following attribute table.

Edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
AB	y_2	x_2	y_1	x_1	m_1
BC	y_2	x_2	y_3	x_3	m_4
CD	y_3	x_3	y_4	x_4	m_3
AD	y_4	x_4	y_1	x_1	m_2

- Now the y_{max} array is sorted. So we can find out the intersection of scan line with first two edges from the sorted attribute table i.e. AB and BC. Every time we are decreasing scan line by 1, from y_{max} to y_{min} of the polygon. In this procedure it may happen that the edge which we have selected to find intersection point may get finished i.e. scan line goes below the y_{min} of selected edge. In that case we have to discard that edge and select next edge from the sorted table and continue till scan line becomes equal to y_{min} of polygon.
- Here important point is how to find out the intersection point of scan line with a particular edge. While at the time of storing attributes of edges we have stored the slope of that edge. We can make use of this slope to find intersection point of scan line with edge. (See Fig. 3.4.2.)

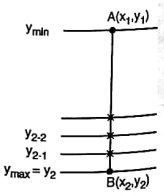


Fig. 3.4.2

- When we are moving from y_{max} to y_{min} every time we are decreasing y by 1, as the distance between two scan line is 1. So we know new points y value, which will be

$y_{max}-1$. Now we have to find, what will be its corresponding x value for the intersection point.

We know that

$$\text{Slope (m)} = \frac{\Delta y}{\Delta x} = \frac{\text{change in y}}{\text{change in x}} = \frac{y_{new} - y_{old}}{x_{new} - x_{old}}$$

$$m = \frac{1}{x_{new} - x_{old}}$$

$$x_{new} - x_{old} = \frac{1}{m}$$

$$\therefore x_{new} = x_{old} + \frac{1}{m}$$ Where m, we know from attribute table.

- x_{old} will be x-coordinate of previous lines intersection point.
- Thus by knowing the slope of every edges we can find the intersection point by decrementing 1 every time from y_{max} to y_{min} and calculating corresponding x by

$$x_{old} + \frac{1}{m}$$

- Like this we are finding intersection of scan line with every edge of polygon. Once we have found the intersection points with both edges i.e. selected edges like AB and BC, then join these two intersection points by solid line and continue. See Fig. 3.4.3.
- Again decrease y by 1 and find new intersection points for those edges till one of the edge gets over, i.e. the scan line goes below y_{min} of one edge. In this case edge BC gets over earlier, so discard edge BC and select next edge from sorted table, which is CD and continue the process. Now we have to find intersection points of scan line with edge AB and edge CD. And then join them. This process continues till scan line reaches to y_{min} of the polygon.

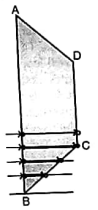


Fig. 3.4.3

- This will work very fine for convex polygons. But in some cases of concave polygons it will introduce errors. See the example given in Fig. 3.4.4.

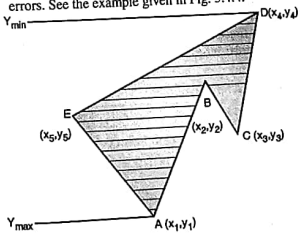


Fig. 3.4.4

- So to handle this situation we have to slightly modify the algorithm.
- Here also we are storing edges of polygon with their attributes. Then we are sorting the y_{max} array also. After sorting y_{max} array for polygon shown in Fig. 3.4.4, we will get the sorted table as follows :

edge	y_{max}	x_{max}	y_{min}	x_{min}	Slope
→ AB	y_1	x_1	y_2	x_2	m_1
→ AE	y_1	x_1	y_5	x_5	m_2
→ BC	y_3	x_3	y_2	x_2	m_3
→ CD	y_3	x_3	y_4	x_4	m_4
DE	y_5	x_5	y_4	x_4	m_5

- Now we are selecting 1st two edges i.e. AB and AE. We are starting our scan lines from y_{max} to y_{min}. If scan line is in between y_{max} and y_{min} of edge AB. Find out intersection point with that edge. Similarly for edge AE also. After finding intersection points join them by solid line. If scan line goes below y_{min} of any one of these two edges then discard that edge and select next edge from the sorted table and continue the process. Upto this point everything is similar to convex polygons. In concave polygons, in addition to checking the scan line is in between y_{max} and y_{min} of the selected lines we have to check also whether scan line goes below to y_{max} of other remaining edges or not, by comparing scan line with y_{max} of remaining lines.
- Now in our example we are selecting first two edges AB and AE. Every time we are checking that scan line is in between the y_{max} and y_{min} of these edges and we are drawing solid lines. See Fig. 3.4.5.
- But at one stage when scan line becomes y_p; at that time as y_p is in between y_{max} and y_{min} of both AB and AE. So we are finding two intersection points. Lets call them as P_1 and P_2. But at the same time the scan line y_p is below the y_{max} of next two edges of the sorted table, i.e. y_p becomes smaller than y_{max} of edge BC and CD.
- So we have to activate these two edges also and find out the intersection of scan line with these edges also. Lets assume that these points are P_3 and P_5. So join P_3 and P_4 and continue. Here we have to draw solid lines from P_1 to P_2 and P_3 to P_4. But not from P_2 to P_3.

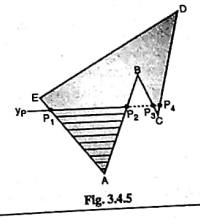


Fig. 3.4.5

- Similarly when the situation is as shown in Fig. 3.4.6, at that time we are going to using the vertex point twice i.e. we are drawing line P_1 to P_2 and P_2 to P_3.

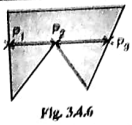


Fig. 3.4.6

- Table 3.4.1 represents the comparison of different polygon filling algorithms.

Table 3.4.1

Flood fill	Boundary fill	Edge fill	Fence fill	Scan line fill
Needs Seed point to start	Needs Seed point to start	Based on complimenting the pixels	Based on complimenting the pixels	Based on line drawing, polygon is filled
Current pixels color is compared with new pixel color	Current pixels color is compared with new pixel color and boundary color	Pixels which are on right side of an edge are getting complemented	Pixels which are on right side of an edge and to the left of fence as well as left side of edge and right side of fence are getting complemented	Intersection of polygon edges are found with the scan lines and then the solid lines are drawn between two such intersection points.
Useful for polygons having single color boundary	Useful for polygons having multi color boundary	More number of pixels are unnecessarily accessed	Less number of pixels are accessed as compared to Edge fill	Need separate attention to handle concave polygons

3.4.1 Pattern Fill Algorithm

- In this section we consider, filling with pattern, which we do by adding extra control to the part of the scan conversion algorithm that actually writes each pixel. The main issue for filling with pattern is the relation of the area of the pattern to that of the primitive. In other words, we need to decide where the pattern is anchored so that we know which pixel in the pattern corresponds to the current pixel of the primitive.
- One technique is to anchor the pattern at a vertex of a polygon by placing the leftmost pixel in the patterns first row. This choice allows the pattern to move when the primitive is moved, a visual effect that would be expected for patterns with a strong geometric organization, such as the cross hatches often used in drafting applications. But there is no distinguished point on a polygon that is obviously right for such a relative anchor, and no distinguished points at all on smoothly varying primitives such as circles and ellipses. Therefore, the programmer must specify the anchor point as appoint on or within the primitive. In some systems, the anchor point may even be applied to a group of primitives.
- Another technique is to consider the entire screen as being tiled with the pattern and to think of the primitive as consisting of an outline or filled area of transparent bits that let the pattern show through. To apply the pattern to the primitive, we index it with the current pixels (x,y) coordinates. Since patterns are defined as small P by Q bitmaps, we use modular arithmetic to make the pattern repeat. The pattern[0,0] pixel is considered coincident with the screen origin, and we can write, a bitmap pattern in transparent mode with the statement

```
If(pattern[x%P][y%Q])
 Writepixel (x,y,value);
```

- If we are filling an entire span in replace write mode, we can copy a whole of the pattern at once, assuming a low-level version of a copy pixel facility is available to write multiple pixels.
- Another technique is to scan convert a primitive first into a rectangular work area, and then to write each pixel from that bitmap to the appropriate place in the canvas. This rectangle write to the canvas is simply a nested loop in which a 1 writes the current color and 0 writes nothing or background color. This two step process is twice as much work as filling during scan conversion and therefore is not worthwhile for primitives that are encountered and scan converted only once. The advantage of a pre-scan converted bitmap lies in the fact that it is clearly faster to write each pixel in a rectangular region, without having to do any clipping or span arithmetic, than to scan convert the primitive each time from scratch while doing such clipping.
- For bi-level displays, writing current color 1, copy pixel works fine : For transparent mode, we use or write mode; for opaque mode, we use replace write mode. For multilevel displays, we cannot write the bitmap directly with a single bit per pixel, but must convert each bit to a full n-bit color value that is then written.

Program to fill polygon

```
/*************************************Program :
A C++ program to implement different polygon filling
algorithms such as
                    * flood fill
                    * Edge fill
                    * Scan line fill
```



```
by making use of mouse to accept the polygon vertices
**************************************/
#include<bstream.h>
#include<conio.h>
#include<dos.h>
#include<process.h>
#include<graphics.h>
#include<math.h>
#include<malloc.h>

struct node
{
    int x,y;
    node * next;
};

union REGS i,o;
struct node * home = NULL;
/*=====================Class Definition.
============================*/
class polyfill
{
    int sx,sy,xx[10],yy[10];
    public:
        polyfill() { }
        initmouse();
        void showmouse();
        void push(int,int);
        void getmousepos(int*,int*,int*);
        void flood(int,int,int,int);
        void scanline(int[],int[],int,int);
        void edge(int[],int[],int,int);
};
/*============================
Member Name     :   initmouse().
Purpose         :   To initialize the mouse.
============================*/
polyfill :: initmouse()
{
    i.x.ax=0;
    int86(0x33,&i,&o);
    return(o.x.ax);
}
/*============================
Member Name     :   showmouse().
Purpose         :   To show the mouse.
============================*/
void polyfill :: showmouse()
{
    i.x.ax=1;
    int86(0x33,&i,&o);
}
/*============================
Member Name     :   getmousepos().
Purpose         :   To get the pixel clicked by the mouse.
============================*/
void polyfill :: getmousepos(int *button,int *x,int *y)
{
    i.x.ax=3;
    int86(0x33,&i,&o);
    *button=o.x.bx;
    *x=o.x.cx;
    *y=o.x.dx;
}
/*============================
Member Name  :  push().
Purpose      :  To maintain the stack.
============================*/
void polyfill :: push(int x,int y)
{
    struct node * newn;
    if(home==NULL)
    {
        newn=(struct node *)malloc(sizeof(struct node));
        newn->x=x;
        newn->y=y;
        newn->next=NULL;
        home=newn;
    }
    else
    {
        newn=(struct node *)malloc(sizeof(struct node));
        newn->x=x;
        newn->y=y;
        home->next=newn;
        home=newn;
    }
}
/*============================
    point, background color and new color
    Member Name :  flood().
    Purpose     :  To implement the flood fill algorithm.
    parameters are :  seed point, background color and
                      new color
============================*/
void polyfill :: flood(int sx,int sy,int b,int col)
{
    int x=sx,y=sy;
    struct node * temp;
    push(x,y);
    temp=home;
    while(temp)
    {
```

```
        x=temp->x;
        y=temp->y;
        putpixel(x,y,col);
if(getpixel(x,y+1)!=15 && getpixel(x,y+1)!=col)
        {
            putpixel(x,y+1,col);
            push(x,y+1);
        }
if(getpixel(x,y-1)!=15 && getpixel(x,y-1)!=col)
        {
            putpixel(x,y-1,col);
            push(x,y-1);
        }
if(getpixel(x+1,y)!=15 && getpixel(x+1,y)!=col)
        {
            putpixel(x+1,y,col);
            push(x+1,y);
        }
if(getpixel(x-1,y)!=15 && getpixel(x-1,y)!=col)
        {
            putpixel(x-1,y,col);
            push(x-1,y);
        }
        struct node * temp1=temp;
        temp=temp->next;
        free(temp1);
        delay(1);
    }
    getch();
}
/* Like flood fill algorithm we can implement Boundary fill
algorithm also. Only difference is we have to check every
pixels color with boundary color, instead of background color.
Students are advised to implement boundary function by their
own */
/*============================
    Member Name    :   edge().
    Purpose        :   To implement the edge fill
                       algorithm.
    Parameters are :   vertices array of polygon, new
                       color, no. of vertices
============================*/
void polyfill :: edge(int xx[10],int yy[10],int col,int n)
{
    float x1,x2,y1,y2,m,x;
    int i=0;
    while(i<n)
    {
        if(yy[i]<yy[i+1])
        {
            y1 = yy[i];
            x1 = xx[i];
            y2 = yy[i+1];
            x2 = xx[i+1];
        }
        else
        {
            y1=yy[i+1];
            x1=xx[i+1];
            y2=yy[i];
            x2=xx[i];
        }
        m=(float)(x2-x1)/(y2-y1);
        while(y1<y2)
        {
            y1++;
            x1=x1+m;
            x=x1;
            while(x<=640)
            {
                if(getpixel(x,y1)==0)
                    putpixel(x,y1,col);
                else
                    putpixel(x,y1,0);
                x++;
            }
            delay(10);
        }
        i++;
    }
    getch();
}
/* Like Edge fill algorithm we can implement fence fill
algorithm also. Students are advised to implement fence
function by their own. */
/*============================
Member Name    :   scanline().
Purpose        :   To implement the scanline algorithm.
Parameters are :   vertices array of polygon, new color,
                   no. of vertices
============================*/
void
 polyfill :: scanline(int xx[10],int yy[10],int col,int n)
{
    int i,k,inter_x[50],temp,y,ymax=0,ymin=480;
    float m[50],dx,dy;
    for(i=0;i<n;i++)
    {
        if(yy[i]>=ymax) ymax=yy[i];
        if(yy[i]<=ymin) ymin=yy[i];

        dx=xx[i+1]-xx[i];
        dy=yy[i+1]-yy[i];
        if(dx==0) m[i]=0;
        if(dy==0) m[i]=1;

        if(dx!=0 && dy!=0)
        m[i]=(float)dx/dy;
    }
    int cnt;
    setcolor(col);

    for(y=ymax;y>=ymin;y--)
    {
        cnt=0;
        for(i=0;i<n;i++)
        {
if((yy[i]>y&&yy[i+1]<=y) || (yy[i]<=y&&yy[i+1]>y))
            {
                inter_x[cnt]=(xx[i]+(m[i]*(y-yy[i])));
                cnt++;
            }
        }
        for(k=0;k<cnt-1;k++)
        {
            for(i=0;i<cnt-1;i++)
            {
                if(inter_x[i]>inter_x[i+1])
                {
                    temp=inter_x[i];
                    inter_x[i]=inter_x[i+1];
                    inter_x[i+1]=temp;
                }
            }
        }
        for(i=0;i<cnt-1;i+=2)
        {
            line(inter_x[i],y,inter_x[i+1]+1,y);
            delay(10);
        }
    }
    getch();
}

/*============================
    .Main Function Definition.
============================*/
void main()
{
    polyfill p;
    int
col,n,xx[10],yy[10],i=0,x,y,button,gm,gd=DETECT,sy,sx,ch;
    cout << "Enter the algorith to follow : \n"
        << "\t1.Flood fill.\n\t2.Edge fill.\n\t3.Scan line.\n"
        << "Entered choice is : ";
    cin >> ch;
    cout << "\nEnter the color number : ";
    cin >> col;
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
    p.initmouse();
p.showmouse();
    do              // accept polygons vertices
    {               // till keyboard gets hit
        p.getmousepos(&button,&x,&y);
        if((button&1)==1)
        {
            delay(500);
            xx[i] = x;
            yy[i] = y;
            i++;
        }
    } while(!kbhit());
    xx[i]=xx[0];
    yy[i]=yy[0];
    n=i;                //n will be no. of vertices
    getch();
    for(int j=0;j<i;j++)
    {
line(xx[j],yy[j],xx[j+1],yy[j+1]);  // draw a polygon
    }
    do
    {
        p.getmousepos(&button,&x,&y);
        if((button&1)==1)
{           // accepting seed point for flood fill
            delay(500);
            sx = x;
            sy = y;
            break;
        }
    }while(1);
    int b = getbkcolor();
    if(ch==1)
        p.flood(sx,sy,b,col);
    else if(ch==2)
        p.edge(xx,yy,col,n);
    else if(ch==3)
        p.scanline(xx,yy,col,n);
    getch();
    closegraph();
}
/*============================*/
```

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words, and for that we are providing guidelines to the that answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 What are the different types of polygon ? How to find whether given point is lieing inside the polygon or not ?

Ans. :

The answer for this question should include Definition of polygon

- Types of polygons i.e. convex and concave
- Give examples of these types by drawing images of polygons.
- For inside point test we have to explain even-odd method and winding number method in detail.
- Explain these methods by giving suitable example and diagrams.
- Explain what will happen if the intersection point is vertex.

Q. 2 What is a 4-connected and 8-connected method ? State its advantages and disadvantages.

Ans. :

Here we have to explain that both 4-connected and 8-connected methods are used for polygon filling. And also include the following points such as.

- How to move from one pixel to another in 4-connected and in 8-connected method.
- Draw both the diagrams.
- Explain where they can be used.

Q. 3 Explain any pixel level polygon filling algorithm.

Ans. :

Here we have to explain any algorithm among the four algorithm such as flood fill, boundary fill, Edge fill and Fence fill, but for all these algorithms, while explaining we have to consider following points.

- Explain the statement of algorithm.
- Draw the diagrams.
- Take an example and show how the algorithm can be applied to fill that polygon.
- Give the advantage and disadvantages of that algorithm.

Q. 4 Explain the geometric level polygon filling algorithm.

Ans. :

The geometric level polygon filling algorithm means we have to explain scan line algorithm for polygon filling. In this algorithm we have to explain

- How we are storing each edge of the polygon with its slope.
- How we are deciding that which edges we have to consider for finding the intersection point
- How to find the intersection point.
- If the polygon is of concave type then how to fill the polygon.
- Explain this algorithm by taking suitable example and diagrams.
- State its advantages and disadvantages.

Important Assignments

This chapter is important from practical examination point of view. We have already explained and given, enough number of programs at the time of explaining algorithms. In addition to these programs, in final practical examination, few modifications are also expected. The list of such programs is given below :

Q. 1 Write a program to implement polygon filling algorithms such as :

- Boundary fill
- Flood fill
- Edge fill
- Fence fill
- Scan line fill
- Accept the co-ordinates of polygon by using mouse.

Q. 2 Write a program to fill half polygon with red and other half with blue color, by using fence fill algorithm.

Q. 3 Write a program to fill the rectangle with three colors as shown below, by using appropriate algorithm.

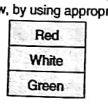


Fig. 1

Q. 4 Write a program to fill one corner of the rectangle with red color and opposite corner with blue color by using appropriate algorithm. See the following polygon.

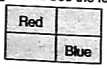


Fig. 2

Q. 5 Write a program to simulate sand clock by using scan line fill a algorithm as shown in Fig. 3.

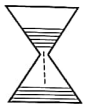


Fig. 3

3.5 Exam Pack (University and Review Questions)

Syllabus Topic : Inside Outside Tests

Q. What is a purpose of inside outside test? Explain with example. *(Refer Section 3.2)* **(Dec. 2016, 5 Marks)**

Q. Illustrate inside-outside tests with example. *(Refer Section 3.2)* **(5 Marks)**

Q. Explain even-odd test to determine whether the point is inside or outside of polygon. *(Refer Section 3.2.1)* **(5 Marks)**

Syllabus Topic : Filled Area Primitive : Scan Line Polygon Fill Algorithm, Boundary Fill and Flood Fill Algorithm

Q. Explain 4-connected and 8-connected methods. *(Refer Section 3.3)* **(5 Marks)**

Q. Write a boundary fill procedure to fill 8-connected region. *(Refer Section 3.3.1)* **(May 2017, 5 Marks)**

Q. Illustrate one example of polygon filling with diagram where 4-connected approach fails, while 8 connected approach succeeds. *(Refer Section 3.3.1)* **(7 Marks)**

Q. Write pseudo codes for boundary fill procedure. *(Refer Section 3.3.1)* **(5 Marks)**

Q. Explain boundary fill algorithms. *(Refer Section 3.3.1)* **(5 Marks)**

Q. Write a pseudo-code to implement boundary fill and flood fill algorithm using 4 connected method. *(Refer Section 3.3.1)* **(10 Marks)**

Q. Explain 4 connected flood fill algorithm in detail. *(Refer Section 3.3.2)* **(Dec. 2014, 5 Marks)**

Q. Write short note on : Flood fill algorithm. *(Refer Section 3.3.2)* **(May 2015, Dec. 2016, 5 Marks)**

Q. Write pseudo codes for flood fill procedure. *(Refer Section 3.3.2)* **(5 Marks)**

Q. Explain flood fill algorithms. *(Refer Section 3.3.2)* **(5 Marks)**

Q. Explain with example Scan line fill algorithm. *(Refer Section 3.4)* **(May 2013, May 2014, 10 Marks)**

Q. Compare Boundary fill and flood fill algorithm. *(Refer Section 3.4)* **(May 2016, 10 Marks)**

Q. Explain scan line fill area conversion algorithm with suitable example. *(Refer Section 3.4)* **(8 Marks)**

□□□

Module 3

CHAPTER 4

2D Geometric Transformations

Chapter 4 describes various two dimensional transformations. There are eight sections under this chapter.

4.1 **Introduction** explains the basic need of two-dimensional transformations.

4.2 **Matrix Representation** sets the background for the 2-D transformations.

4.3 **Basic Transformations** explains the basic scaling, rotation and translation operations with the derivation of their matrices.

4.4 **Homogeneous Co-ordinate** system explains the different way of representing images with the help of matrix.

4.5 **Rotation about arbitrary point** explains various steps to perform rotation when the reference point is other than origin.

4.6 **Reflection Transformations** are the transformations which are other than the basic transformations.

4.7 **Inverse Transformations** are necessary when we want to find the original image from the transformed image.

4.8 **Solved Problems** are given in last section to get an idea of how to solve the 2D problems.

4.1 Introduction

- A graphics system must allow the user to change the way object appears. In the real world, we frequently rearrange objects or look at them from different angles. The computer should do the same with images on the screen. We can change the size of an object, its position on the screen, or its orientation.
- It is much more sensible to make changes later to the objects original basic description. Implementing such a change is called a **transformation**. Basically transformation means making changes or allowing modifications to the picture.

Syllabus Topic : Matrix Representation

4.2 Matrix Representation

- Our computer graphics images are generated from a series of line segments which are represented by the co-ordinates of their end points. Certain changes in an image can be easily made by performing mathematical operations on these co-ordinates. Before we go in details of various transformations let us review some of the mathematical tools we shall need, namely, matrix multiplication. We will consider a matrix to be a two-dimensional array of numbers.

☞ **Example**

$$\begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} \qquad \begin{vmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{vmatrix} \qquad |1\ 1\ 2\ 1| \text{ etc.}$$

- We are defining a matrix as set of rows and columns. Suppose our matrix A is

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

- Then we are saying that matrix A is of dimension (3 × 3) i.e. having 3 rows and 3 columns. We can access a particular element in this matrix by specifying its position in terms of row and column. In above (3 × 3) matrix, we can access element 3 as 3rd row and 1st column i.e. A (3,1) = 3.
- We can perform many operations on matrix like matrix addition, subtraction, multiplication, Transpose etc. out of these operations we are most dealing with matrix multiplication. But multiplication of two matrices is not a simple product of two numbers. We can multiply two matrices A and B only if the number of columns of the 1st matrix (A) is the same as the number of rows of 2nd matrix (B).

☞ **Example**

$$\text{If } A = \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \text{ and } B = \begin{vmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{vmatrix}$$

- In this case matrix A is of dimension (3 × 3) and B is of (3 × 2). Here can we multiply these two matrices? Yes, we can, because the columns of the 1st matrix are 3 and the rows of 2nd matrix are also 3. Therefore we can perform matrix multiplication. Unlike multiplication of numbers, the multiplication of matrices is not commutative. i.e. while we can multiply A time B, we cannot multiply B times A; because B has only 2 columns and A is having 3 rows. So for this combination we cannot perform multiplication.
- When we perform multiplication of two matrices we will get the result as a matrix only. This resultant matrix will be having dimension as the same number of rows as that of first matrix and the same number of columns as the second matrix. Multiplying the (3 × 3) matrix with the (3 × 2) matrix gives a (3 × 2) matrix as the resultant matrix.
- The elements of the product matrix are given in terms of the elements of matrices A and B.

☞ **Example**

$$A = \begin{vmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{vmatrix} \qquad B = \begin{vmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{vmatrix}$$

(3 × 3) (3 × 2)

Column's of A = Row's of B

- The dimension of resultant matrix C will be (3 × 2) and the elements in resultant matrix will be calculated as,

$$\begin{aligned} C(1,1) &= A(1,1)\cdot B(1,1) + A(1,2)\cdot B(2,1) + A(1,3)\cdot B(3,1) \\ &= (1)\cdot(1) + (4)\cdot(3) + (7)\cdot(5) = 1 + 12 + 35 \\ &= 48 \end{aligned}$$

Similarly,

$$\begin{aligned} C(1,2) &= A(1,1)\cdot B(1,2) + A(1,2)\cdot B(2,2) + A(1,3)\cdot B(3,2) \\ &= (1)\cdot(2) + (4)\cdot(4) + (7)\cdot(6) \\ &= 2 + 16 + 42 \\ &= 60 \end{aligned}$$

- To form the resultant matrix we are multiplying 1st row of 1st matrix with 1st column of 2nd matrix and that result will be taken for C(1, 1). Similarly to find C(1, 2) we are multiplying 1st rows of 1st matrix with 2nd column of 2nd matrix and so on. Like this we are finding multiplication of two matrices.
- There is a special type of matrix. If we multiply any matrix A with this special matrix then the result will be the as original matrix A. Such a special matrix is called as **Identity matrix**. This identity matrix is a square matrix i.e. having number of rows and columns are same with all the elements 0 except the main diagonal elements are all 1.

Example

$$I = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} \text{ or } \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

We can call this identity matrix as I matrix.

$\therefore A * I = A$

Syllabus Topic : Basic Transformations – Translation, Scaling, Rotation, Raster Method for Transformation

4.3 Basic Transformations

- Basically transformation means change in image. We can modify the image by performing some basic transformations, such as :

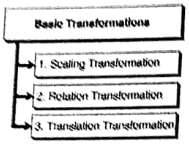


Fig. C4.1 : Basic Transformations

4.3.1 Scaling Transformation

Q. Draw matrix for representing following operation : Scaling. (5 Marks)

- Scaling is changing the size of the object or image. Suppose we are having point P_1 whose co-ordinates are (x_1, y_1), then we can represent this point P_1 in matrix form as

$$P_1 = (x_1, y_1) = [x_1, y_1]$$

- The dimension for this matrix will be (1 x 2) i.e. one row and two columns.

$$\therefore P_1 = [x_1, y_1]$$

- Now suppose if we want to multiply this point i.e. P_1's matrix, by some another matrix, say matrix T of dimension (2 x 2), then we will get resultant point P_2 with dimension as (1 x 2).

i.e. $P_1 * T = P_2$

$[x_1, y_1] * T = [x_2, y_2]$

- Thus we can say, the matrix T gives the relation between original point P and the new point P_2. But if the matrix T is identity matrix then point P_2 will be the same as P.

i.e. $P * T = P_2$

$$P * \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = P_2$$

$$[x_1, y_1] * \begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix} = [x_1, y_1]$$

$\therefore P_1 = P_1$

But if matrix T is not identity matrix, but it is

$$T = \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix}$$ Then P_2 will become

$$P_1 * T = P_2$$

$$[x_1, y_1] * \begin{vmatrix} 2 & 0 \\ 0 & 1 \end{vmatrix} = [2x_1, y_1]$$

$$\therefore P_2 = [2x_1, y_1]$$

- i.e. all x co-ordinates will get double as compared to original one. But the y co-ordinates remains same as original. See Fig. 4.3.1. It means the width of the object becomes double but height is not getting changed.

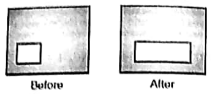


Fig. 4.3.1

If matrix T is $T = \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix}$ Then

$$P_1 * T = P_2$$

$$[x_1, y_1] * \begin{vmatrix} 0.5 & 0 \\ 0 & 1 \end{vmatrix} = [0.5x_1, y_1]$$

$$\therefore P_2 = [0.5x_1, y_1]$$

- i.e. all x co-ordinates becomes half of the original and y co-ordinates remains same as original. See Fig. 4.3.2.

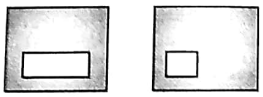


Fig. 4.3.2

- Similarly we can change the height of image also by keeping width constant.

In that case our transformation matrix will be

$$T = \begin{vmatrix} 1 & 0 \\ 0 & 2 \end{vmatrix}$$

- Like this we can get both width and height double also by using matrix T as

$$T = \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix}$$

- So in general term, we can say matrix S is a scaling matrix and s_x is a scale factor for x co-ordinate and s_y will be the scale factor for y co-ordinate.

$$S = \begin{vmatrix} s_x & 0 \\ 0 & s_y \end{vmatrix}$$

- When we scale the image, all points will get changed except origin. A scale in x by a factor greater than 1 will causes the image to elongate i.e., wider. A scale in x by a factor less than 1 causes the image to shrink. Similarly a scale in y will change its height.

4.3.2 Rotation Transformation

Q. Draw matrix for representing following operation : Rotation (5 Marks)

- For rotation we need some trigonometry. See Fig. 4.3.3. Suppose we have a point $P_1 = (x_1, y_1)$ and we rotate it about the origin by an angle θ to get a new position $P_2 = (x_2, y_2)$. We wish to find the transformation which will change (x_1, y_1) into (x_2, y_2) i.e.

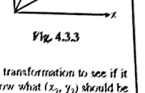


Fig. 4.3.3

$$P_1 * T = P_2$$

- But before we can check any transformation to see if it is correct, we have to first know what (x_2, y_2) should be in terms of (x_1, y_1) and angle θ. To determine this we need trigonometric functions like sine and cosine.
- Let us say we have drawn a line from origin to point (x, y) of length 'L' at an angle θ anticlockwise from x-axis, as shown in Fig. 4.3.4. The line segment L will have (0, 0) and (x, y) as its two end points.

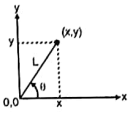


Fig. 4.3.4

$\therefore$ We can say $L = (x^2 + y^2)^{1/2}$

Now $\sin\theta = \dfrac{\text{oppsoite side}}{\text{hypotenuse}} = \dfrac{y}{L} = \dfrac{y}{(x^2+y^2)^{1/2}}$

Similarly $\cos\theta = \dfrac{\text{adjcent side}}{\text{hypotenuse}} = \dfrac{x}{L} = \dfrac{x}{(x^2+y^2)^{1/2}}$

If we consider length as exactly 1 then

$$\sin\theta = \frac{y}{1} = y$$

and $\cos\theta = \dfrac{x}{1} = x$ ...(4.3.1)

- Now we will determine the transformation matrix for rotation.
- Consider a point P_1 as (1, 0). (Refer Fig. 4.3.5). After rotating this point P_1 in anticlockwise by an angle θ we will get point P_2. Now co-ordinates of P_2 will be

Fig. 4.3.5

$x_2 = \cos\theta$

$y_2 = \sin\theta$ ... From Equations (4.3.1)

- Similarly if current point P_1 is (0, 1). (Refer Fig. 4.3.6) and if we rotate P_1 in anticlockwise by θ angle then we will get P_2 as (– sin θ, sin θ).

Here distance of P_1 from origin is 1 unit.

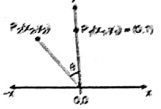


Fig. 4.3.6

$\therefore L = 1$

as $\sin\theta = \dfrac{\text{opposite side}}{\text{hypotenuse}} = \dfrac{-x}{L}$ as $L = 1$

$= -x$

Similarly $\cos\theta = \dfrac{\text{adjcent side}}{\text{hypotenuse}} = \dfrac{y}{L}$ as $L = 1$

$= y$

$\therefore x = -\sin\theta$ and $y = \cos\theta$

Coming back to original concept

$$P_1 * T = P_2$$

Where P_1 is original point

P_2 is new point

T is transformation matrix

For case 1 i.e. when $P_1 = (1, 0)$

$$P_1 * T = P_2$$

$$|1, 0| * |T| = |\cos\theta, \sin\theta|$$

- Now we know original point i.e. P_1, we also know what will be new point i.e. P_2. Here we have to find what will be the transformation matrix, which will change P_1 to P_2. For simplicity let us assume that transformation matrix (T) as

$$T = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

Now, $|1, 0| * \begin{vmatrix} a & b \\ c & d \end{vmatrix} = |a\ b|$

$\therefore \cos\theta = a$ and $\sin\theta = b$

Similarly for case 2 i.e. when $P_1 = (0, 1)$

$$P_1 * T = P_2$$

$$|0, 1| * |T| = |-\sin\theta, \cos\theta|$$

$$\therefore |0, 1| * \begin{vmatrix} a & b \\ c & d \end{vmatrix} = |c, d|$$

$\therefore c = -\sin\theta$ and $d = \cos\theta$

- Now, place the values of a, b, c, d in transformation matrix.

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

- $\therefore$ For rotation in anticlockwise direction, the matrix will be

$$R = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

- The sign of angle determines the direction of rotation. When angle is positive, i.e. when we are following the direction exactly reverse as that of the direction of rotating hands of a normal clock, then it is **anticlockwise** or sometimes it is called as Counter clockwise. See Fig. 4.3.7. And when angle is negative i.e. when we are following the same path as that of hands of clock then it is called **clockwise**, as shown in Fig. 4.3.8.

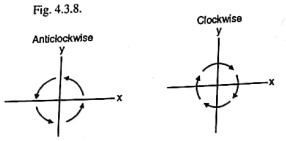


Fig. 4.3.7

Fig. 4.3.8

- So to rotate a point in clockwise direction our matrix will become

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

- Here as we are rotating in clockwise direction, so we are replacing θ by $-\theta$. But $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin\theta$

The transformation matrix for rotation in clockwise direction will be

$$R = \begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix} \quad \text{and}$$

For Anticlockwise rotation

$$R = \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

➜ 4.3.3 Translation Transformation

Q. Draw matrix for representing operation : Translation. (5 Marks)

- Moving the whole image is called translation. We can easily shift the image by adding same value to all the points of image by which we want to shift the image.

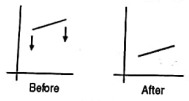


Fig. 4.3.9

- If we want to shift the image down by 5 units then we have to add – 5 to all the y values of that image. (See Fig. 4.3.9).
- Similarly if we want to shift image up by 5 units, as shown in Fig. 4.3.10, then add 5 to all y values of image.
- Same thing for x values also.

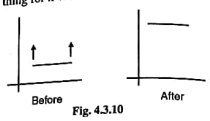


Fig. 4.3.10

- In general to translate the image we have to add some value to old (x, y) values, so that we will get new (x, y) values.

$$\therefore \quad x_2 = x_1 + t_x \quad \text{and} \quad y_2 = y_1 + t_y$$

where t_x is a translation factor for x-axis

and t_y is a translation factor for y-axis.

- Depending on values of t_x and t_y we are shifting the image to new position. But we cannot represent translation by using matrix.

Syllabus Topic : Homogeneous Co-ordinates

4.4 Homogeneous Co-ordinates

- We have seen and developed matrices for rotation about origin in clockwise and anticlockwise direction. But suppose we want to rotate a point (x, y) with respect to a point other than the origin i.e. Q as shown in Fig. 4.4.1, then in that case we cannot use our rotation matrices, because they are considering reference point as origin.

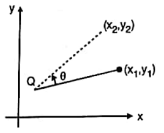


Fig. 4.4.1

- So to handle such situations we have to use series of transformations. i.e. we have to first translate that image until the center of rotation becomes origin. Now we can perform rotation with respect to origin by angle θ in clockwise or in anticlockwise direction. But it doesn't finish our job, we have to again translate this new image back to its position by shifting the image with the same amount which we have used when we shifted image to origin.
- So here we have to use, three operations or transformations namely translation, rotation and again translation. So instead of three transformations can we accomplish this effect in single transformation for the sake of simplicity? Yes, we can combine different transformations by using **Homogeneous co-ordinates.**

- In homogeneous co-ordinates we use 3 × 3 matrix instead of 2 × 2 and introduce additional dummy variable w. Generally, points are represented as (x, y); but here every point is specified by a set of 3 values (xw, yw, w). The first value of homogeneous co-ordinate will be product of x and w, the 2nd will be y and w and the third will be just w. But how we are going to represent this homogeneous point on screen because screen deals with only x and y. so we have to recover x and y from (xw, yw, w). The x and y co-ordinates can be easily recovered by dividing the 1st and 2nd number by the third. We are not really using the 3rd variable w. For simplicity we are keeping the value of w as 1.

In normal co-ordinates the scaling matrix is

$$s = \begin{vmatrix} s_x & 0 \\ 0 & s_y \end{vmatrix}$$

But in homogeneous co-ordinate it becomes

$$s = \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- In homogeneous co-ordinates as every point is represented as (xw, yw, w)

$$P_1 * T = P_2$$

$$|xw, yw, w| * \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = |\, xw \cdot s_x, yw \cdot s_y, w \,|$$

- Dividing first and second number by w, we will get,

$|\, x \cdot s_x, y \cdot s_y \,|$ which will be correctly scaled point.

Similarly the anticlockwise rotation matrix can be represented in homogeneous co-ordinates as

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix} \text{ becomes } \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- Let us say we are applying anticlockwise rotation to point P_1 then

$$P_1 * T = P_2$$

$$|\, xw, yw, w \,| * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$= |\, (xw \cdot \cos\theta - yw \cdot \sin\theta), (xw \cdot \sin\theta + yw \cdot \cos\theta), w \,|$

After dividing by w we will get,

$$P_2 = (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

- We know that we cannot represent translation by using matrix. For translation we have to say

$$x_2 = x_1 + t_x \quad \text{and} \quad y_2 = y_1 + t_y$$

- But by using homogeneous co-ordinates we can represent translation in the form of matrix. It will be

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

Let us see whether it works or not .

$$P_1 * T = P_2$$

$$|\, xw, yw, w \,| * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} = |\, (xw + t_x w), (yw + t_y w), w \,|$$

After dividing by w we will get translated point P_2 as

$$|\, x + t_x, y + t_y \,|$$

- So, by using homogeneous co-ordinates we can represent translation in the form of matrix and also we can combine different transformations to form a single transformation matrix.
- Normalized device co-ordinate is a good example of transformation. In normalized device co-ordinates screen is treated as 1 × 1 size. See Fig. 4.4.2. We can convert screen co-ordinates to normalized co-ordinates by using

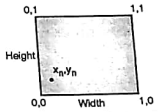


Fig. 4.4.2

$$x_n = \frac{x_s - \text{widthstart}}{\text{width}}$$

$$y_n = \frac{y_s - \text{heightstart}}{\text{height}}$$

- where x_n, y_n represents normalized co-ordinates and x_s, y_s represents screen co-ordinates. Similarly we can convert normalized co-ordinates to screen co-ordinates by using

$$x_s = \text{width} * x_n + \text{widthstart}$$

$$y_s = \text{height} * y_n + \text{heightstart}$$

- By using transformation we can form a single matrix which will convert normalized co-ordinates to screen co-ordinates.
- Here to convert normalized co-ordinate to screen co-ordinate i.e. x co-ordinate of screen, we are multiplying x_n with width and then adding widthstart. Similarly for y co-ordinate of screen we are multiplying y_n with height and then adding heightstart. That means in terms of transformation we have to say, first scale x_n or y_n by width or height and then translate it by width-start or height-start.

Let us create a full transformation matrix for this.

$$P_1 * T = P_2$$

- But T is itself a combination of Scaling and Translation.

$$\therefore T = \begin{vmatrix} width & 0 & 0 \\ 0 & height & 0 \\ 0 & 0 & 1 \end{vmatrix} \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ widthstart & heightstart & 1 \end{vmatrix}$$

(Scaling) (Translation)

$$\therefore T = \begin{vmatrix} width & 0 & 0 \\ 0 & height & 0 \\ widthstart & heightstart & 1 \end{vmatrix}$$

4.5 Rotation about Arbitrary Point

➔ **(May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, May 2017)**

Q. Derive matrices for Rotation about an arbitrary point. **MU - May 2014, Dec. 2014, May 2017. 10 Marks**

Q. Explain the steps used in rotation of 2-D object an arbitrary axis and derive the matrices for same. **MU - May 2015, Dec. 2015, May 2016. 10 Marks**

Q. Write short note on : Composite transformation. **MU - May 2015. 5 Marks**

Q. Derive 2D transformation matrix, for performing rotation of given point P(X, Y) by angle θ (theta) in anticlockwise direction about origin. **(5 Marks)**

- We have derived rotation matrices with respect to origin. But suppose the reference point of rotation is other than origin, then in that case we have to follow series of transformation. Such transformation is also called as **composite transformation.** Consider Fig. 4.5.1,

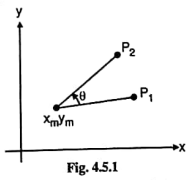


Fig. 4.5.1

- Assume that we have to rotate a point P_1 with respect to (x_m, y_m) then we have to perform three steps. First we have to translate the (x_m, y_m) to origin as shown in Fig. 4.5.2. So our translation matrix (T_1) will become

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} \Rightarrow$$

Here $t_x = -x_m$ and $t_y = -y_m$

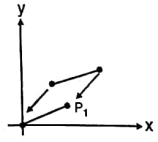


Fig. 4.5.2

- Then we have to follow second step i.e. rotate it in clockwise or anticlockwise. See Fig. 4.5.3. Let's assume as anticlockwise rotation by angle θ.

So our rotation matrix will be

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

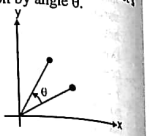


Fig. 4.5.3

- Now follow 3rd step as shown in Fig. 4.5.4 i.e. translate back to original position. So the translation matrix (T_2) will become

$$T_2 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix} \Rightarrow$$

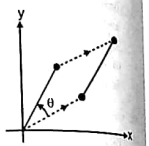


Fig. 4.5.4

Now let us form a combined matrix.

= Translation * Rotation * Translation

$$= T_1 * R * T_2$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_m & -y_m & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ x_m & y_m & 1 \end{vmatrix}$$

$$= \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ -x_m * \cos\theta + y_m * \sin\theta + x_m & -x_m * \sin\theta - y_m \cos\theta + y_m & 1 \end{vmatrix}$$

This transformation matrix is the overall transformation matrix for rotation about arbitrary point (x_m, y_m) by an angle θ in anticlockwise direction.

Syllabus Topic : Other Transformations -Reflection and Shear

4.6 Reflection Transformations

- Until now we have seen some basic transformations such as scaling, rotation and translation. There are some other transformations also, which are applied on axes. It is also called as co-ordinate transformation.

4.6.1 Reflection at y-axis

- Basically reflection at y-axis is similar to placing a mirror at y-axis and taking the mirror image of an object. If we have a point (x, y) then the reflection at y-axis will become (– x, y) which is shown in Fig. 4.6.1.
- Here the magnitude of both x and y remains same. Only the sign of x co-ordinate gets changed. Now let us derive what will be the transformation matrix which gives us the desired result.

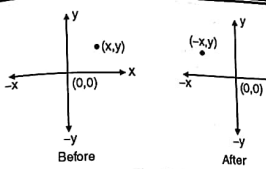


Fig. 4.6.1

Our basic transformation rule will be

$$P_1 * T = P_2$$

Now, $P_1 = [x, y]$ and we need P_2 as $[-x, y]$

$$\therefore (x, y) * T = (-x, y)$$

- Now what should be the transformation matrix 'T', to get the desired result.

So the T will be $\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$. Let us check it out

$$P_1 * T = P_2$$

$$[x, y] * \begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix} = [-x, y]$$

- After multiplying the transformation matrix with the point P_1 we will get the desired point P_2.

4.6.2 Reflection at x-axis

- Reflection at x-axis will be very similar to reflection at y-axis. Here we have to change the rolls of x and y only. It means now we are assuming that we are keeping mirror at x-axis. So, if the point is (x, y) then its reflection at x-axis will become as (x, – y). See Fig. 4.6.2.

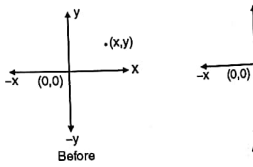


Fig. 4.6.2

- Here, after reflection the values of x co-ordinates remains same, but the y co-ordinate values get changed by sign only. So let us find the transformation matrix for reflection at x-axis.

$$P_1 * T = P_2$$

$$\therefore (x, y) * T = (x, -y)$$

- As the values of y becomes negative, so the transformation matrix will be

$$T = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$$

Let us check it out

$$P_1 * T = P_2$$

$$[x, y] * \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} = [x, -y]$$

- Here we can make a statement as, for reflection at a particular axis we have to keep all the values of that particular axis as it is and change the sign of remaining axis only.

4.6.3 Reflection in the Origin

- We have seen reflection at x-axis and reflection at y-axis. But we have seen them separately. Suppose if we want both these transformations simultaneously then, we have to apply reflection in the origin.

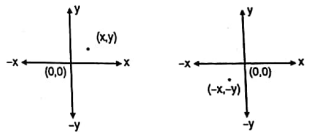


Fig. 4.6.3

- Reflection at origin is a combination of both reflection at x-axis and reflection at y-axis see Fig. 4.6.3. So, its transformation matrix will be

$$\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} * \begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$$

$$T = \begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$$

$$P_1 * T = P_2$$

$$(x, y) * \begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix} = (-x, -y)$$

4.6.4 Reflection in the Line y = x

- Here we are not going to take reflection at any standard axis or in the origin, but it is at line y = x. Here we are drawing a line whose x and y values are same i.e. the line is exactly at 45° and passes through the origin. See Fig. 4.6.4.

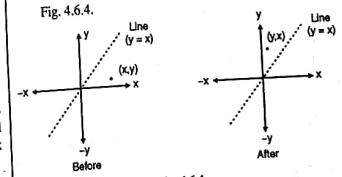


Fig. 4.6.4

- The dotted line shows a line y = x. Now if we want to take reflection of a point (x, y) with respect to this line, then the point (x, y) will become point (y, x). It means as if we are placing mirror at line y = x. In this case the values of x and y gets interchanged.

$$P_1 * T = P_2$$

$\therefore (x, y) * T = (y, x)$

$\therefore$ The transformation matrix T will be

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

$$\therefore (x, y) * \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = (y, x)$$

4.6.5 Shear Transformations in 2D

➔ (May 2015, May 2016)

Q. Define shearing and give example. MU - May 2015. 5 Marks

Q. Write short note on : Shearing transformation. MU - May 2016. 10 Marks

- The shear transformation means slanting the image. This shear transformation is of two types :

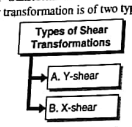


Fig. C4.2 : Types of shear Transformations

➔ 4.6.5(A) Y-shear

- In y-shear we are keeping x co-ordinate values as it is and shifting the y co-ordinate values only, which results in tilting of horizontal lines. See Fig. 4.6.5.

$$P_1 * T = P_2$$

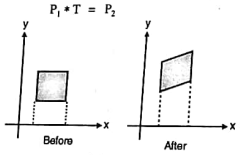


Fig. 4.6.5

- If P_1 is (x, y), then x co-ordinate of P_2 must be same as x co-ordinate of P_1. But y co-ordinate of P_2 should change.
- $\therefore$ Transformation matrix for y-shear will be $\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$ where a is constant factor which will decide, how much to tilt.

Now, $P_1 * T = P_2$

$$|x, y| * \begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix} = |x, x \cdot a + y|$$

- So x co-ordinate of P_2 are not changed but y co-ordinate of P_2 will be the combination of constant value 'a' and 'x' and y values of P_1.

$$P_2 = (x, xa + y)$$

here we can say value of y will depend on value of x.

➔ 4.6.5(B) X-shear

- x-shear preserves the y co-ordinate values and shifts x co-ordinate values causing vertical lines to tilt. See Fig. 4.6.6.
- If point P_1 is (x, y) then y co-ordinate of P_2 must remain unchanged but x co-ordinate of P_2 should change.

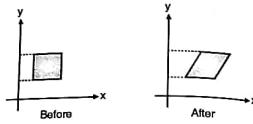


Fig. 4.6.6

$\therefore$ Transformation matrix for x-shear will be

$\begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix}$ where b is constant factor, which will decide, how much to tilt.

Now, $P_1 * T = P_2$

$$|x, y| * \begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix} = |x + yb, y|$$

Here we have to say, value of x will depend on value of y.

Example 4.6.1

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).

Solution :

Given :

A square ABCD having (1, 0), (0, 0), (0, 1), (1, 1)

Let's represent a square ABCD in matrix form with homogeneous coordinates as

$$\begin{vmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

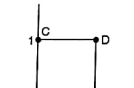


Fig. P. 4.6.1(a)

We need to rotate this square in anticlockwise direction by 45° about a point (1, 0) i.e. point 'A'.

$\therefore$ First we need to translate point A i.e. (1, 0) to origin so that we can apply rotation.

$\therefore$ We need translation matrix having tx = – 1 and ty = 0.

$$\therefore \begin{vmatrix} 1 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix}$$

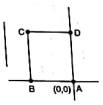


Fig. P. 4.6.1(b)

Now we can perform rotation

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{vmatrix} * \begin{vmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 0 & 1 \\ -0.7 & -0.7 & 1 \\ -1.4 & 0 & 1 \\ -0.7 & 0.7 & 1 \end{vmatrix}$$

Now we need to translate this square back to its original position i.e. this time tx and ty will become as : tx = + 1 and ty = 0.

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ -0.7 & -0.7 & 1 \\ -1.4 & 0 & 1 \\ -0.7 & 0.7 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 \\ 0.3 & -0.7 & 1 \\ -0.4 & 0 & 1 \\ 0.3 & 0.7 & 1 \end{vmatrix}$$

$\therefore$ The final co-ordinates of square ABCD will be A(1, 0), B (0.3, – 0.7), C (– 0.4, 0), D (0.3, 0.7)

Example 4.6.2

Perform a 45° rotation of triangle A (0, 0), B (1, 1), C (5, 2).

(i) about the origin (ii) about P (– 1, – 1).

Solution :

Let's represent triangle by matrix form

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

Case 1 : About the origin

$$\begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 5 & 2 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{vmatrix}$$

$$\therefore \begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 5 & 2 \end{vmatrix} * \begin{vmatrix} 0.7 & 0.7 \\ -0.7 & 0.7 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 0 & 1.4 \\ 2.1 & 4.9 \end{vmatrix}$$

After 45° rotation about origin, the new co-ordinates of triangle ABC will be (0, 0), (0, 1.4) and (2.1, 4.9)

Case 2 : about p (– 1, – 1)

Here we need three steps, which are

- Translation
- Rotation
- Translation

For first translation we have to consider tx = 1 and ty = 1 to shift the point P to origin.

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 6 & 3 & 1 \end{vmatrix}$$

Now perform rotation

$$\begin{vmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 6 & 3 & 1 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 2 & 2 & 1 \\ 6 & 3 & 1 \end{vmatrix} * \begin{vmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 1.4 & 1 \\ 0 & 2.8 & 1 \\ 2.1 & 6.3 & 1 \end{vmatrix}$$

Now again perform translation with tx = – 1, ty = – 1.

$$\therefore \begin{vmatrix} 0 & 1.4 & 1 \\ 0 & 2.8 & 1 \\ 2.1 & 6.3 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{vmatrix} = \begin{vmatrix} -1 & 0.4 & 1 \\ -1 & 1.8 & 1 \\ 1.1 & 5.3 & 1 \end{vmatrix}$$

$\therefore$ The final co-ordinates of a triangle ABC after 45° rotation about a point p (– 1, – 1) will be (– 1, 0.4), (– 1, 1.8) and (1.1, 5.3)

Example 4.6.3

Consider the square A(1, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square by 45° anticlockwise direction followed by reflexion about X-axis.

Solution :

Given : A square ABCD with (1, 0), (0, 0), (0, 1), (1, 1)

Let's first represent the square in matrix form as,

$$\begin{vmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{vmatrix}$$

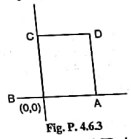


Fig. P. 4.6.3

We need to rotate a square ABCD by 45° is anticlockwise direction

$$\therefore \begin{vmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{vmatrix}$$

$$\therefore \begin{vmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{vmatrix} * \begin{vmatrix} 0.7 & 0.7 \\ -0.7 & 0.7 \end{vmatrix} = \begin{vmatrix} 0.7 & 0.7 \\ 0 & 0 \\ -0.7 & 0.7 \\ 0 & 1.4 \end{vmatrix}$$

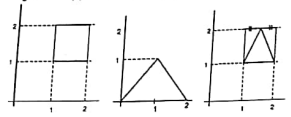
Now perform reflection about X-axis.

$$\therefore \begin{vmatrix} 0.7 & 0.7 \\ 0 & 0 \\ -0.7 & 0.7 \\ 0 & 1.4 \end{vmatrix} * \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} = \begin{vmatrix} 0.7 & -0.7 \\ 0 & 0 \\ 0.7 & -0.7 \\ 0 & -1.4 \end{vmatrix}$$

The final co-ordinates of the square will be (0.7, – 0.7), (0, 0), (0.7, – 0.7), (0, – 1.4).

Example 4.6.4

Fig. P. 4.6.4(a) and (b) show basic 2D blocks. Apply translation and scaling transformations to get the Fig. P. 4.6.4(c). Draw diagrams of all intermediate steps.

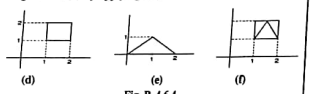


(a) (b) (c)

Fig. P. 4.6.4

Solution :

Fig. P. 4.6.4(a) and (b) are given. We need to get Fig. P.4.6.4(c) by applying different transformations.



(d) (e) (f)

Fig. P. 4.6.4

Here we have to fit Fig. P. 4.6.4(e) inside Fig. P. 4.6.4(d). Here height in Fig. P. 4.6.4(e) is same as that at Fig. P. 4.6.4(f). So we have to apply scaling on Fig. P. 4.6.4(e) with $sx = \frac{1}{2}$

We have to use scaling matrix as $s = \begin{vmatrix} 1/2 & 0 \\ 0 & 1 \end{vmatrix}$

The matrix form of Fig. P. 4.6.4(e) is $\begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 0 \end{vmatrix}$

After multiplying this by scaling matrix we will get

$$\begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 2 & 0 \end{vmatrix} * \begin{vmatrix} 1/2 & 0 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 1/2 & 1 \\ 1 & 0 \end{vmatrix}$$

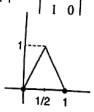


Fig. P. 4.6.4(g)

Now this triangle we have to shift inside Fig. P. 4.6.4(f) so we need translation with tx = 1 and ty = 1.

$$\begin{vmatrix} 0 & 0 & 1 \\ 1/2 & 1 & 1 \\ 1 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 1.5 & 2 & 1 \\ 2 & 1 & 1 \end{vmatrix}$$

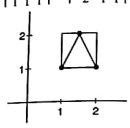


Fig. P. 4.6.4(b)

Example 4.6.5

A 2D rectangular block with 1 unit height and 2 units width has one vertex "A" at origin. The block is shifted by 1 unit in X-direction and scaled by 2 units along Y-direction. Draw initial state of the rectangle and transformed final state of given rectangle. Give complete mathematical formulation.

Solution :

Given :

Rectangle with width = 2 and height = 1 with one vertex at origin.

Let's draw the initial state of rectangle and represent it in matrix form as

$$\begin{vmatrix} 0 & 0 \\ 2 & 2 \\ 10 & 4 \end{vmatrix} \quad \begin{vmatrix} 0 & 0 \\ 0 & 1 \\ 2 & 1 \\ 2 & 0 \end{vmatrix}$$

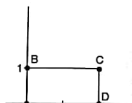


Fig. P. 4.6.5(a)

This rectangle is shifted by 1 unit in x direction.

$$\begin{vmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 2 & 1 & 1 \\ 2 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 0 & 1 \end{vmatrix}$$

After translation the rectangle will be

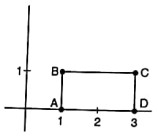


Fig. P. 4.6.5(b)

Now this rectangle we have to scale by 2 units in y-direction.

$$\therefore \begin{vmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 3 & 1 & 1 \\ 3 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1 \\ 1 & 2 & 1 \\ 3 & 2 & 1 \\ 3 & 0 & 1 \end{vmatrix}$$

After scaling the rectangle will look like

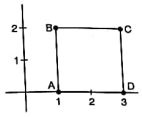


Fig. P. 4.6.5(c)

Example 4.6.6

Magnify the triangle with vertices A (0, 0) B (1, 1), C (5, 2) to twice its size as well as rotate it by 45°. Derive the translation matrices.

Solution :

Magnify the triangle to twice means we need to scale the triangle with Sx = 2 and Sy = 2.

Let's represent the given triangle in matrix form as

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

Now we need scaling matrix with Sx = 2 and Sy = 2.

$$\begin{vmatrix} 0 & 0 \\ 1 & 1 \\ 5 & 2 \end{vmatrix} * \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 2 & 2 \\ 10 & 4 \end{vmatrix}$$

Since the rotation angle is 45° we have to apply anticlockwise rotation.

$$\therefore \begin{vmatrix} 0 & 0 \\ 2 & 2 \\ 10 & 4 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{vmatrix} = * \begin{vmatrix} 0.707 & 0.707 \\ -0.707 & 0.707 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 0 \\ 0 & 2.828 \\ 4.242 & 9.898 \end{vmatrix}$$

∴ The new co-ordinates of triangle after magnifying it to twice and rotating by 45° will be A (0, 0), B (0, 2.828), C (4.242,9.898)

Example 4.6.7 MU - May 2013, 10 Marks

Derive the transformation matrix to Magnify the triangle with vertices A (0, 0) B (1, 2), C (3, 2) to twice its size so that the point C (3,2) remain fixed.

Solution :

Magnify the triangle to twice means we need to scale the triangle with $S_x = 2$ and $S_y = 2$.

Let's represent the given triangle in matrix form as

$$\begin{vmatrix} 0 & 0 \\ 1 & 2 \\ 3 & 2 \end{vmatrix}$$ Now we need scaling matrix with $S_x = 2$ and $S_y = 2$.

$$\begin{vmatrix} 0 & 0 \\ 1 & 2 \\ 3 & 2 \end{vmatrix} * \begin{vmatrix} 2 & 0 \\ 0 & 2 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 2 & 4 \\ 6 & 4 \end{vmatrix}$$

Since the point C(3,2) should remain fixed we have to apply translation with tx = – 3 and ty = – 2. For this we have to use homogeneous coordinates

$$\begin{vmatrix} 0 & 0 & 1 \\ 2 & 4 & 1 \\ 6 & 4 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -2 & 1 \end{vmatrix} = \begin{vmatrix} -3 & -2 & 1 \\ -1 & 2 & 1 \\ 3 & 2 & 1 \end{vmatrix}$$

∴ The new co-ordinates of triangle after magnifying it to twice and keeping point C(3, 2) fixed will be A (– 3, – 2), B (– 1, 2), C (3, 2)

4.7 Inverse Transformation

- The inverse transformation is very similar to matrix inversion. When any normal matrix (A) is multiplied with the inverse of same matrix (A^{-1}) then resultant will be identity matrix.

$$A * A^{-1} = I$$

- Similarly when we multiply any transformation matrix with its inverse, we will get identity matrix.
- Now let us revise how to find the inverse of a matrix. First step for this is to find the determinant of the matrix.

Suppose we want to find inverse of matrix A.

$$\det(A) = \det \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix}$$

$$= A_{11}(A_{22} \cdot A_{33} - A_{23} A_{32}) - A_{12}(A_{21} A_{33} - A_{23} \cdot A_{31}) + A_{13}(A_{21} \cdot A_{32} - A_{22} \cdot A_{31})$$

Then step two will be finding the cofactor of the matrix.

To find cofactor of

$$\begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix}$$

We have to solve

$$B_{11} = \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} \quad B_{12} = \begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} \quad B_{13} = \begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix}$$

$$B_{21} = \begin{vmatrix} A_{12} & A_{13} \\ A_{32} & A_{33} \end{vmatrix} \quad B_{22} = \begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} \quad B_{23} = \begin{vmatrix} A_{11} & A_{12} \\ A_{31} & A_{32} \end{vmatrix}$$

$$B_{31} = \begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \quad B_{32} = \begin{vmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \end{vmatrix} \quad B_{33} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix}$$

Then cofactor (A) will be $\begin{vmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ B_{31} & B_{32} & B_{33} \end{vmatrix}$

- After finding co-factor, next step is to find the adjoint of matrix A. To find this adjoint we have to take transpose of co-factor.

$$\text{i.e. Adj (A)} = \begin{vmatrix} B_{11} & B_{21} & B_{31} \\ B_{12} & B_{22} & B_{32} \\ B_{13} & B_{23} & B_{33} \end{vmatrix}$$

$$\text{Now inverse} = \frac{\text{Adj (A)}}{\text{determinant (A)}}$$

- But in case of transformations we are having direct formula to find the inverse of transformations. For this we have to use homogeneous co-ordinates and we have to arrange the matrix in specific order.

$$\text{Inverse} \begin{vmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{vmatrix} = \frac{1}{(ae-bd)} \begin{vmatrix} e & -d & 0 \\ -b & a & 0 \\ (bf-ce) & (cd-af) & (ae-bd) \end{vmatrix}$$

Let us take an example so that the things will get more clear.

Example 4.7.1

A polygon co-ordinates are A (7,3), B (9, 3), C (9,5) and D (7,5). We have done scaling Sx = Sy = 2 and reflection through origin and translation by 1 in both x of y-direction. Find the original figure.

Solution :

Here polygon co-ordinates are given, but these co-ordinates are of final figure. i.e. We know the last state. In addition to that we know what are the operations made. Now we have to find the original figure.

To solve this problem, we are having two methods :

1. Just undo all the transformations OR
2. Use inverse transformation method.

By 1st method

Here we have to undo all the transformations. So the transformations done on object are, 1st scaling by 2 then reflection through origin and then lastly translation by 1 in both x and y-direction.

So first we have to undo the operation which is done last. The transformed image co-ordinates are

$$\begin{vmatrix} 7 & 3 & 1 \\ 9 & 3 & 1 \\ 9 & 5 & 1 \\ 7 & 5 & 1 \end{vmatrix}$$

we are representing polygon co-ordinates in terms of homogeneous co-ordinates.

$$\text{Now} \begin{vmatrix} 7 & 3 & 1 \\ 9 & 3 & 1 \\ 9 & 5 & 1 \\ 7 & 5 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{vmatrix} = \begin{vmatrix} 6 & 2 & 1 \\ 8 & 2 & 1 \\ 8 & 4 & 1 \\ 6 & 4 & 1 \end{vmatrix}$$

(Transformed image) (Translation)

Here, in translation matrix t_x and t_y will be – 1 as we are undoing the operations.

Then we have performed reflection about origin. So undo that also by multiplying this matrix with reflection matrix.

$$\begin{vmatrix} 6 & 2 & 1 \\ 8 & 2 & 1 \\ 8 & 4 & 1 \\ 6 & 4 & 1 \end{vmatrix} * \begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} -6 & -2 & 1 \\ -8 & -2 & 1 \\ -8 & -4 & 1 \\ -6 & -4 & 1 \end{vmatrix}$$

Then we are scaling this new object by ½ because we have doubled the co-ordinates at the time of scaling. So,

$$\begin{vmatrix} -6 & -2 & 1 \\ -8 & -2 & 1 \\ -8 & -4 & 1 \\ -6 & -4 & 1 \end{vmatrix} * \begin{vmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} -3 & -1 & 1 \\ -4 & -1 & 1 \\ -4 & -2 & 1 \\ -3 & -2 & 1 \end{vmatrix}$$

the original figure must be having co-ordinates as,

A (– 3, – 1), B (– 4, – 1), C (– 4, – 2), D (– 3, – 2)

By 2nd method

Generally we know that

$$P_1 * T = P_2$$

Where P_1 is old object

P_2 is new object

T is transformation matrix.

Generally we know old object and from that we have to find new object by using transformations. But here we know transformations and the new object. From this information we have to find the original object.

Now if we multiply T^{-1} to both sides of

$$P_1 * T = P_2$$

Then $T^{-1} * P_1 * T = P_2 * T^{-1}$

as $T * T^{-1} = I$

$$\therefore P_1 = P_2 * T^{-1}$$

So we have to find T^{-1}. To find this T^{-1} we must form a transformation matrix; which will be

$$T = \text{I Scaling I} * \text{I Reflection I} * \text{I Translation I}$$

$$\therefore T = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix}$$

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

Over here there is no need to take scaling as $\frac{1}{2}$ or translation as – 1. Now we have arranged the transformation matrix in the form of $\begin{vmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{vmatrix}$

According to the inverse transformation formula

$$\text{Inverse} \begin{vmatrix} a & d & 0 \\ b & e & 0 \\ c & f & 1 \end{vmatrix} = \frac{1}{(ae-bd)} \begin{vmatrix} e & -d & 0 \\ -b & a & 0 \\ (bf-ce) & (cd-af) & (ae-bd) \end{vmatrix}$$

$$\therefore \text{Inverse} \begin{vmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 1 & 1 & 1 \end{vmatrix} = \frac{1}{4} \begin{vmatrix} -2 & 0 & 0 \\ 0 & -2 & 0 \\ 2 & 2 & 4 \end{vmatrix}$$

$$= \begin{vmatrix} -2/4 & 0 & 0 \\ 0 & -2/4 & 0 \\ 2/4 & 2/4 & 4/4 \end{vmatrix}$$

$$\therefore T^{-1} = \begin{vmatrix} -1/2 & 0 & 0 \\ 0 & -1/2 & 0 \\ 1/2 & 1/2 & 1 \end{vmatrix}$$

Using $P_1 = P_2 * T^{-1}$

$$= \begin{vmatrix} 7 & 3 & 1 \\ 9 & 3 & 1 \\ 9 & 5 & 1 \\ 7 & 5 & 1 \end{vmatrix} * \begin{vmatrix} -1/2 & 0 & 0 \\ 0 & -1/2 & 0 \\ 1/2 & 1/2 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 7(-1/2)+0+1/2 & 0+(-1/2)*3+1/2 & 0+0+1 \\ 9(-1/2)+0+(+1/2)*1 & 0+(-1/2)*3+1/2 & 0+0+1 \\ 9(-1/2)+0+(1/2)*1 & 0+(1/2)*5+(1/2)*1 & 0+0+1 \\ 7(-1/2)+0+(1/2)*1 & 0+(-1/2)*5+(1/2)*1 & 0+0+1 \end{vmatrix}$$

$$P_1 = \begin{vmatrix} -3 & -1 & 1 \\ -4 & -1 & 1 \\ -4 & -2 & 1 \\ -3 & -2 & 1 \end{vmatrix}$$

So, the original figure is having co-ordinates as

A (– 3, – 1), B (– 4, – 1), C (– 4, – 2), D (– 3, – 2).

The results of method I and method II are same. But the inverse transformation method is much simple than the ordinary method.

Program for 2D Transformations

```
/*****************************************
    Program Title : 2D Transformations.
    A C++ program to implement different 2D
transformations such as
 * Scaling       * Reflection about Y-axis
 * Translation   * Reflection about x-axis
 * Rotation      * Reflection about origin
 * X shear       * Reflection about a line Y = X
 * Y shear       * Reflection about a line Y = -X
*****************************************/
#include <iostream.h>
#include <conio.h>
#include <process.h>
#include <graphics.h>
#include <math.h>
/*============================
Class Definition.
============================*/

class transform
{
  private: // Initialiasing data members
        int input[20][2],scalm[2][2];
        int trans[2][2],clockw;
        int ref[2][2];
        float rot[2][2],theta,resmr[20][2];
        int resm[20][2];
        int shear_mat[2][2];
        int edges,scalx,scaly,tx,ty;
        int shx,shy;
public:  // Declaring all transformation functions
        void xshear();
        void yshear();
        void menu();
        void menu1();
        void accept();
        void scale();
        void translate();
        void rotate();
        void reflect();
void  multiply(int a[20][2],int b[2][2],int c[20][2]);
void  multiplyxx(int a[20][2],int b[2][2],int c[20][2], int
edges);
void  multiply1(int a[20][2],float b[2][2],float c[20][2];
        void plot(int mat[20][2]);
        void plot1(float mat[20][2]);
};

/*============================
Member Name  :  accept().
Purpose      :  To accept the 2D object from the user
                and plot object on the screen in four
                quadrant fashion.
============================*/
void  transform::accept()
{
     int i,j;
     cleardevice();
cout<<"Enter the number of edges in the figure : ";
     cin>>edges;
cout<<"Enter the Co-ordinates in Matrix form : ";
     for(i=1;i<=edges;i++)
     {
        for(j=1;j<=2;j++)
        {
        cout<<"\nA["<<i<<"]["<<j<<"] = ";
```

```
            cin>>input[i][j];
            }
        }
        for(i=1;i<=edges;i++)
        {
            input[i][1] = input[i][1] + 320;
            input[i][2] = 240 - input[i][2];
        }

plot(input);        // Displaying the accepted polygon
        for(i=1;i<=edges;i++)
        {
            input[i][1] = input[i][1] - 320;
            input[i][2] = 240 - input[i][2];
        }
}
void  transform::menu()
{
        int choice;
        cleardevice();
        cout<<"\t\tMenu"
                <<"\n1> Scaling"
                <<"\n2> Translation"
                <<"\n3> Rotate"
                <<"\n4> Shearing"
                <<"\n5> Reflection"
                <<"\nEnter Your Choice\t";

                cin>>choice;
                switch(choice)
                {
         case 1: scale();
                        break;
         case 2: translate();
                        break;
         case 3: rotate();
                        break;
         case 4: menu1();
                        break;
         case 5: reflect();
                        break;
         default:exit(0);
                }
}
/*============================
Member Name  :  reflect().
Purpose      :  To select any type of reflection
transformation
============================*/

id transform::reflect()
{
        int choice,i;
        cleardevice();
        cout<<"\t\tMenu"
<<"\n1> Reflection About Y Axis"
        <<"\n2> Reflection About X Axis"
        <<"\n3> Reflection About Origin"
        <<"\n4> Reflection About y = x Line"
        <<"\n5> Reflection About y = -x Line"
                <<"\nEnter Your Choice\t";
                cin>>choice;
                switch(choice)
                {
case 1:         // Matrix for reflection about Y-axis
                ref[1][2] = ref[2][1] = 0;
                ref[1][1] =-1;
                ref[2][2] = 1;
                break;
case 2:         // Matrix for reflection about X-axis
                ref[1][2] = ref[2][1] = 0;
                ref[1][1] = 1;
                ref[2][2] =-1;
                break;
case 3:         // Matrix for reflection about origin
                ref[1][2] = ref[2][1] = 0;
                ref[1][1] =-1;
                ref[2][2] =-1;
                break;
case 4:       // Matrix for reflection about Y = X line
                ref[1][2] = ref[2][1] = 1;
                ref[1][1] = 0;
                ref[2][2] = 0;
                break;
case 5:       // Matrix for reflection about Y = -X line
                ref[1][1] =0;
                ref[2][2] =0;
                ref[1][2] = ref[2][1] =-1;
                break;
                }
accept();  // Accept the oroginal polygon
        multiply(input,ref,resm);
        for(i=1;i<=edges;i++)
        {
            resm[i][1] = resm[i][1] + 320;
            resm[i][2] = 240 - resm[i][2];
        }
        plot(resm);
}
/*============================
Member Name  :  menu1().
Purpose      :  To select either X or Y shear transformation
```



```
============================*/
void transform::menu1()
{
        int choice;
        cleardevice();
        cout<<"\t\tMenu"
                <<"\n1> X Shear"
                <<"\n2> Y Shear"
                <<"\nEnter Your Choie\t";
        cin>>choice;
        switch(choice)
        {
            case 1: xshear();
                        break;
            case 2: yshear();
                        break;
        }
}
/*============================
        Member Name  :  xshear().
        Purpose      :  To perfrom X shearing
transformation.
============================*/
void  transform::xshear()
{
        int i,sides;
        accept();  // Accepting the polygon

        cout<<"\nEnter the x Shear Factor\t";
cin>>shx;       // Accepting the X shear factor
        shear_mat[1][1] = shear_mat[2][2] = 1;
        shear_mat[1][2] = 0;
        shear_mat[2][1] = shx;
        multiplyxx(input,shear_mat,resm,sides);
        edges=sides;
        for(i=1;i<=edges;i++)
        {
            resm[i][1] = resm[i][1] + 320;
            resm[i][2] = 240 - resm[i][2];
        }
        plot(resm);
}
/*============================
        Member Name  :  yshear().
        Purpose      :  To perfrom Y shearing
transformation.
============================*/
void  transform::yshear()
{
        int i,sides;
        accept();  // Accepting the polygon
        sides = edges;
        cout<<"\nEnter the y Shear Factor\t";
cin>>shy;               // Accepting the Y shear factor
        shear_mat[1][1] = shear_mat[2][2] = 1;
        shear_mat[2][1] = 0;
        shear_mat[1][2] = shy;
        multiplyxx(input,shear_mat,resm,sides);
        edges=sides;
        for(i=1;i<=edges;i++)
        {
            resm[i][1] = resm[i][1] + 320;
            resm[i][2] = 240 - resm[i][2];
        }
        plot(resm);
}
/*============================
        Member Name  :  rotate().
        Purpose      :  To perform clockwise or
                        anticlockwise rotation
                        with specified angle.
============================*/
void transform::rotate()
{
        int i;
        float theta1;
accept();               // Accept the original polygon
        cout<<"\nEnter the angle for rotation\t";
cin>>theta;             // Accept the angle to rotate
        cout<<"Enter 1 for clockwise rotation 0 for
anticlokwise";
cin>>clockw;            // Accept the direction for rotation
        theta1= ((3.14*theta)/180 );
// Conversion of degree to radian
        if(clockw==1)
        {
            rot[1][1] = rot[2][2] = cos(theta1);
            rot[1][2] = -sin(theta1);
            rot[2][1] = sin(theta1);
        }
        Else
        {
            rot[1][1] = rot[2][2] = cos(theta1);
            rot[1][2] = sin(theta1);
            rot[2][1] = -sin(theta1);
        }
        multiply1(input,rot,resmr);
        for(i=1;i<=edges;i++)
        {
            resmr[i][1] = (int)(resmr[i][1] + 320);
            resmr[i][2] = (int)(240 - resmr[i][2]);
        }
        plot1(resmr);
}
```

```
/*=========================================
Member Name  :  scale().
Purpose      :  To perfrom scaling transformation.
=========================================*/
void transform::scale()
{
    int i;
accept();                  // Accept the polygon
    cout<<"\nEnter the Scale X Factor\t";
cin>>scalx;                // Accept the SX factor
    cout<<"\nEnter the Scale Y Factor\t";
cin>>scaly;                // Accept the SY factor
    scalm[1][1] = scalx;
    scalm[1][2] = scalm[2][1] = 0;
    scalm[2][2] = scaly;
    multiply(input,scalm,resm);
    for(i=1;i<=edges;i++)
    {
        resm[i][1] = resm[i][1] +320;
        resm[i][2] = 240 - resm[i][2];
    }
    plot(resm);
}

void transform::multiply(int a[20][2],int b[2][2],int c[20][2])
{   int i;
    for(i=1;i<=edges;i++)
    {
    c[i][1] = (a[i][1] * b[1][1]) + (a[i][2] * b[2][1]);
    c[i][2] = (a[i][1] * b[1][2]) + (a[i][2] * b[2][2]);
    }
}
void transform::multiplyxx(int a[20][2],int b[2][2],int
c[20][2],int edges)
{
    int i;
    for(i=1;i<=edges;i++)
    {
    c[i][1] = (a[i][1] * b[1][1]) + (a[i][2] * b[2][1]);
    c[i][2] = (a[i][1] * b[1][2]) + (a[i][2] * b[2][2]);
    }
}

void transform::multiply1(int a[20][2],float b[2][2],
float c[20][2])
{
    int i;
    for(i=1;i<=edges;i++)
    {
    c[i][1] = (a[i][1] * b[1][1]) + (a[i][2] * b[2][1]);
    c[i][2] = (a[i][1] * b[1][2]) + (a[i][2] * b[2][2]);
    }
}

/*=========================================
Member Name  :  plot().
Purpose      :  To display the polygon on screen.
=========================================*/
void transform::plot(int mat[20][2])
{
    int i;
    setcolor(10);
    line(0,240,639,240);
    line(320,0,320,479);
    outtextxy(295,243,"0,0");
    setcolor(WHITE);
    for(i=1;i<edges;i++)
        line(mat[i][1],mat[i][2],mat[i+1][1],mat[i+1][2]);
    line(mat[1][1],mat[1][2],mat[i][1],mat[i][2]);
    getch();
}

void transform::plot1(float mat[20][2])
{   int i;
    setcolor(10);
    line(0,240,639,240);
    line(320,0,320,479);
    outtextxy(295,243,"0,0");
    setcolor(WHITE);
    for(i=1;i<edges;i++)
    {   line(mat[i][1],mat[i][2],mat[i+1][1],mat[i+1][2]);
        getch();
    }
    line(mat[1][1],mat[1][2],mat[i][1],mat[i][2]);
}

/*=========================================
Member Name  :  translate().
Purpose      :  To perfrom translation operation.
=========================================*/
void transform::translate()
{   int i,j;
accept();          // Accept the polygon
    cout<<"\nEnter tx Factor\t";
cin>>tx;   // Accept the translation factor for X-axis
    cout<<"\nEnter ty Factor\t";
cin>>ty;   // Accept the translation factor for Y-axis
    tx = 320 - tx;
    ty = 240 + ty;
    for(i=1;i<=edges;i++)
    {
```



```
    input[i][1] = input[i][1] + 640 - tx ;
    input[i][2] = 480 - input[i][2] - ty ;
    }
    plot(input);
}
int main(void)
{
    char ch;
    int gd = DETECT,gm,i;
    do
    {
        initgraph(&gd,&gm,"c:\\teplus\\bgi");
        transform t;
        t.menu();
        closegraph();
        cout<<"Do You Wanna Continue(y/n)?\t";
        ch = getche();
    }while(ch=='y'||ch=='Y');
    return 0;
}
/*========== End of Program ==========*/
```

4.8 Solved Problems

Example 4.8.1

Scale a square ABCD with co-ordinates A (0, 0), B (5, 0), C(5, 5) and D (0, 5) by 2 units for x-direction and 3 units for y-direction.

Solution :

Let us store the co-ordinates of square ABCD in the form of homogeneous matrix. As we know that the homogeneous co-ordinates require three values i.e. xw, yw and w. So the homogeneous matrix to represent the square ABCD will be

$$\begin{vmatrix} 0 & 0 & 1 \\ 5 & 0 & 1 \\ 5 & 5 & 1 \\ 0 & 5 & 1 \end{vmatrix}$$

Now we have to scale this square ABCD by 2 units in x-direction and 3 units in y-direction. It means scaling factor for x-direction i.e. s_x should be 2 and scaling factor for y-direction i.e. s_y should be 3.

$\therefore$ Scaling matrix in homogeneous co-ordinates will be

$$s = \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 5 & 0 & 1 \\ 5 & 5 & 1 \\ 0 & 5 & 1 \end{vmatrix} * \begin{vmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ 10 & 0 & 1 \\ 10 & 15 & 1 \\ 0 & 15 & 1 \end{vmatrix}$$

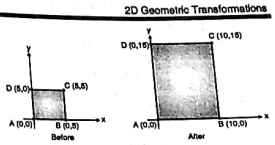


Fig. P. 4.8.1

Above Fig. P. 4.8.1 gives idea about the original square and the square after applying transformation.

Example 4.8.2

Translate the square ABCD whose co-ordinate are A (0, 0), B (3, 0), C (3, 3) and D (0, 3) by 2 units in both directions and then scale it by 1.5 units in x-direction and 0.5 units in y-direction.

Solution :

Here we have to divide the given problem into two parts. First part is translation and second will be scaling.

1st translation

The square ABCD will be represented at matrix in the following way.

$$\begin{vmatrix} 0 & 0 & 1 \\ 3 & 0 & 1 \\ 3 & 3 & 1 \\ 0 & 3 & 1 \end{vmatrix}$$

We have to translate this original square by 2 units in both direction. So here t_x and t_y will equal to 2. So the translation matrix will be

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{vmatrix}$$

After performing translation we will get,

$$\begin{vmatrix} 0 & 0 & 1 \\ 3 & 0 & 1 \\ 3 & 3 & 1 \\ 0 & 3 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 2 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 5 & 1 \\ 2 & 5 & 1 \end{vmatrix}$$

The pictorial representation of this translation step will be as shown in Fig. 4.8.2(a).

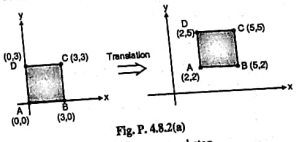


Fig. P. 4.8.2(a)

Now we have to perform second step.

2nd scaling :

Here we have to scale this translated image by 1.5 units in x-direction and by 0.5 units in y-direction.

Scaling matrix will be

$$S = \begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Here $s_x = 1.5$ and $s_y = 0.5$

$\therefore$ The scaling matrix will become

$$S = \begin{vmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$\therefore$ The translated image will become as

$$\begin{vmatrix} 2 & 2 & 1 \\ 5 & 2 & 1 \\ 5 & 5 & 1 \\ 2 & 5 & 1 \end{vmatrix} * \begin{vmatrix} 1.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 3 & 1 & 1 \\ 7.5 & 1 & 1 \\ 7.5 & 2.5 & 1 \\ 3 & 2.5 & 1 \end{vmatrix}$$

The pictorial representation of this step will be as shown in Fig. 4.8.3.

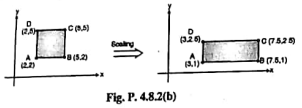


Fig. P. 4.8.2(b)

Example 4.8.3

Rotate a triangle defined by A (0, 0), B (6, 0), and C (3, 3) by 90° about origin in anticlockwise direction.

Solution :

Here we have to first form a matrix for specified triangle. It will be

$$\begin{vmatrix} 0 & 0 & 1 \\ 6 & 0 & 1 \\ 3 & 3 & 1 \end{vmatrix}$$

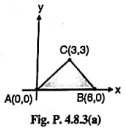


Fig. P. 4.8.3(a)

The pictorial representation of this triangle will be as shown in Fig. 4.8.4.

Now we have to rotate this image by 90° in anticlockwise direction. So the rotation matrix for anticlockwise direction will be

$$\begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

As we want to rotate it by 90°. So the θ will be 90°.

$$\begin{vmatrix} \cos 90 & \sin 90 & 0 \\ -\sin 90 & \cos 90 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Now we have to multiply this rotation matrix with the original image matrix.

$$= \begin{vmatrix} 0 & 0 & 1 \\ 6 & 0 & 1 \\ 3 & 3 & 1 \end{vmatrix} * \begin{vmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 0 & 1 \\ 0 & 6 & 1 \\ -3 & 3 & 1 \end{vmatrix}$$

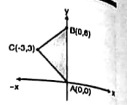


Fig. P. 4.8.3(b)

So the pictorial representation of the final image will be as shown in Fig. 4.8.5.

Example 4.8.4

Show that transformation matrix of reflection about a line y = x is equivalent to reflection relative to x-axis followed by anticlockwise rotation of 90°.

Solution :

In this problem we have to prove that reflection about a line y = x, i.e. its transformation matrix, is equivalent to the combination of reflection at x-axis and anticlockwise rotation of 90°.

The transformation matrix for reflection at line y = x is

$$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$$

Now let us see the right hand side.

For reflection at x-axis, the transformation matrix will be $\begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix}$

And the rotation matrix for anticlockwise direction of 90° will be

$$\begin{vmatrix} \cos 90 & \sin 90 \\ -\sin 90 & \cos 90 \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

Now we have to show left hand side is equivalent to right hand side.

$$\therefore \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} * \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

$$\begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$$

Hence it is proved that reflection at line y = x is equivalent to reflection at x-axis followed by anticlockwise rotation of 90°.

Example 4.8.5

Find the transformation matrix that transforms the square ABCD whose centre is at (2, 2) is reduced to half of its size, with centre still remaining at (2, 2). The co-ordinates of square ABCD are A (0, 0), B (0, 4), C (4, 4) and (4, 0). Find the co-ordinates of new square.

Solution :

Here the object is a square ABCD as shown in Fig. 4.8.6 and we have to perform scaling transformation to reduce its size to half of the original. But the center of this new square must be same as that of original squares center i.e. (2, 2). So we need another transformation i.e. translation. Let's do the transformations one by one.

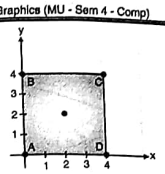


Fig. P. 4.8.5(a)

The matrix for original object is $\begin{vmatrix} 0 & 0 & 1 \\ 0 & 4 & 1 \\ 4 & 4 & 1 \\ 4 & 0 & 1 \end{vmatrix}$

Now we have to scale this to half

$$\therefore s_x = \frac{1}{2} \text{ and } s_y = \frac{1}{2}$$

So, we have to perform

$$\begin{vmatrix} 0 & 0 & 1 \\ 0 & 4 & 1 \\ 4 & 4 & 1 \\ 4 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ 0 & 2 & 1 \\ 2 & 2 & 1 \\ 2 & 0 & 1 \end{vmatrix}$$

The pictorial representation of this scaled square is as shown in Fig. 4.8.7. After scaling the center the square ABCD will become (1, 1). But we want the center of the square as (2, 2), so we have to use translation.

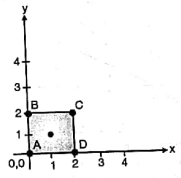


Fig. P. 4.8.5(b)

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 0 & 2 & 1 \\ 2 & 2 & 1 \\ 2 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 3 & 3 & 1 \\ 3 & 1 & 1 \end{vmatrix}$$

After translating the square ABCD by 1 unit in x and 1 unit in y-direction, we get the square ABCD with center (2, 2) as shown in Fig. 4.8.8.

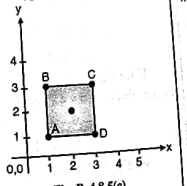


Fig. P. 4.8.5(c)

Example 4.8.6

Perform x-shear and y-shear on a triangle having A (2, 1), B (4, 3), C (2, 3). Consider the constant value a = b = 2.

Solution : We have to represent the co-ordinates of triangle in the form of matrix.

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

Let us perform y-shear.

For this the y-shear matrix will be $\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix}$ where 'a' is constant.

In y-shear we have to shift y co-ordinates by keeping x co-ordinates as it is.

$$\therefore \begin{vmatrix} 2 & 1 \\ 4 & 3 \\ 2 & 3 \end{vmatrix} * \begin{vmatrix} 1 & 2 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 5 \\ 4 & 11 \\ 2 & 7 \end{vmatrix}$$

Pictorial representation will be as shown in Fig. P. 4.8.6(a).

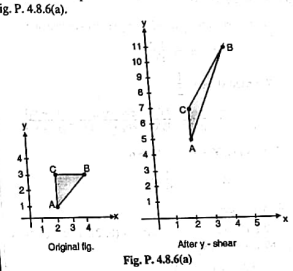


Fig. P. 4.8.6(a)

Now let us perform x-shearing

$$\begin{vmatrix} 2 & 1 \\ 4 & 3 \\ 2 & 3 \end{vmatrix} * \begin{vmatrix} 1 & 0 \\ 2 & 1 \end{vmatrix} = \begin{vmatrix} 4 & 1 \\ 10 & 3 \\ 4 & 3 \end{vmatrix}$$

original x-shear
triangle matrix

In x-shear, x co-ordinate will get shifted by keeping y co-ordinates as it is. The pictorial representation of x-shear will be as shown in Fig. P. 4.8.6(b).

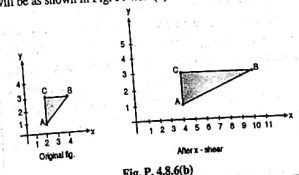


Fig. P. 4.8.6(b)

Example 4.8.7

For the triangle ABC, A (2, 4), B (4, 6) and C (2, 6) obtain reflection through the line $y = \frac{1}{2}(x + 4)$.

Solution :

We know that equation of the line in slope intercept form is

$$y = m \cdot x + b$$

where m = slope and b = y-intercept

Now, here we have to draw a line whose equation is $y = \frac{1}{2}(x + 4)$. If we compare this line equation with normal line equation then it will become

$$y = 0.5 \cdot x + 2$$

i.e. slope should be 0.5 and y-intercept should be 2.

So, we have to draw a line such that its slope is 0.5 and it intersects y-axis at 2.

But slope is the ratio of change in y to change in x.

$$\therefore m = \frac{DY}{DX}$$

We need slope as $\frac{1}{2}$

$$\therefore \frac{1}{2} = \frac{DY}{DX} = \frac{\text{change in y}}{\text{change in x}} = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

Let's say DY = 3 and DX = 6

$$\therefore \frac{1}{2} = \frac{3}{6} = \frac{1}{2}$$

As y-intercept is 2.

∴ We can predict first point as (0, 2). Treat (0, 2) as (x_1, y_1). Now find (x_2, y_2).

$$\therefore \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{DY}{DX}$$

$$\therefore \frac{y_2 - 2}{x_2 - 0} = \frac{3}{6}$$

$$\therefore y_2 = 5 \text{ and } x_2 = 6$$

Now we are having a line (x_1, y_1) to (x_2, y_2) having slope equal to 0.5 and whose y-intercept is 2.

Let us show it by pictorially. See Fig. P. 4.8.7(a).

Now we have to reflect a triangle with respect to this line. We are having different transformation matrices for different reflections like, reflection at y-axis, x-axis, origin, y = x line etc., but not other line. So we cannot use directly any one of the above transformation matrix. So we have to perform different transformations to achieve this reflection.

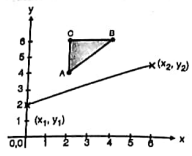


Fig. P. 4.8.7(a)

Step I : We will first translate this line in such a way that one of the end point of this line will become origin, so that we can apply our normal transformations. So we have to translate this line and triangle by 2 units in y-direction only.

$$\therefore \begin{vmatrix} 2 & 4 & 1 \\ 4 & 6 & 1 \\ 2 & 6 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & -2 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 3 & 1 \\ 4 & 4 & 1 \\ 2 & 4 & 1 \end{vmatrix}$$

After translation the triangle becomes as shown in Fig. P. 4.8.7(b).

Similarly reflection line will become (0, 0) to (6, 3).

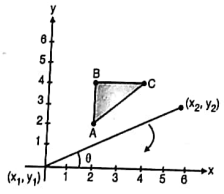


Fig. P. 4.8.7(b)

Step II : Now we can apply **rotation** to this reflection line, to match the line with any of the axis, or to the line y = x i.e. θ = 45°.

To match the reflection line with x-axis we have to rotate this line by θ in clockwise direction. See Fig. 4.8.12. But we know that

$$\cos\theta = \frac{\text{adj. side}}{\text{hypotenuse}} = \frac{6}{\sqrt{6^2 + 3^2}} = \frac{6}{6.7}$$

$$\therefore \cos\theta = 0.895$$

$$\therefore \theta = \cos^{-1}(0.895) = 26.5°$$

If we want to match the reflection line with line y = x i.e. line drawn at θ = 45° then we have to rotate this line by

$$\theta_1 = 45° - 26.5°$$

$$\theta_1 = 18.5°$$

See Fig. P. 4.8.7(c).

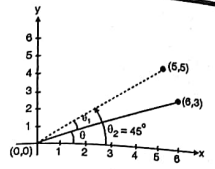


Fig. P. 4.8.7(c)

∴ We have to perform rotation with θ_1 in anticlockwise direction.

$$\begin{vmatrix} 2 & 2 & 1 \\ 4 & 4 & 1 \\ 2 & 4 & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta_1 & \sin\theta_1 & 0 \\ -\sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 2 & 2 & 1 \\ 4 & 4 & 1 \\ 2 & 4 & 1 \end{vmatrix} * \begin{vmatrix} 0.948 & 0.317 & 0 \\ -0.317 & 0.948 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 1.26 & 2.53 & 1 \\ 2.52 & 5.06 & 1 \\ 1.26 & 5.06 & 1 \end{vmatrix}$$

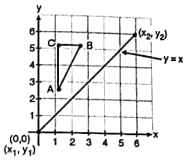


Fig. P. 4.8.7(d)

Now the triangle ABC will become as shown in Fig. P. 4.8.7(d).

Step III : Now we can perform **reflection** of triangle ABC with respect to line y = x

$$\therefore \begin{vmatrix} 1.26 & 2.53 & 1 \\ 2.52 & 5.06 & 1 \\ 1.26 & 5.06 & 1 \end{vmatrix} * \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 2.53 & 1.26 & 1 \\ 5.06 & 2.52 & 1 \\ 5.06 & 1.26 & 1 \end{vmatrix}$$

[Reflection matrix for y = x line.]

Pictorial representation will be as shown in Fig. P. 4.8.7(e).

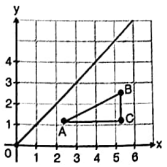


Fig. P. 4.8.7(e)

Step IV : We have got reflection but now we have to perform the **inverse rotation** and inverse translation again to get the actual results.

Earlier in step II we have performed rotation about an angle 18.5° in anticlockwise direction. So now we have to rotate by same angle but in clockwise direction. See the Fig. P. 4.8.7(f).

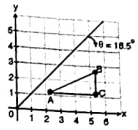


Fig. P. 4.8.7(f)

$$\begin{vmatrix} 2.53 & 1.26 & 1 \\ 5.06 & 2.52 & 1 \\ 5.06 & 1.26 & 1 \end{vmatrix} * \begin{vmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 2.53 & 1.26 & 1 \\ 5.06 & 2.52 & 1 \\ 5.06 & 1.26 & 1 \end{vmatrix} * \begin{vmatrix} 0.948 & -0.317 & 0 \\ 0.317 & 0.948 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 2.79 & 0.392 & 1 \\ 5.58 & 0.78 & 1 \\ 5.58 & 0.41 & 1 \end{vmatrix}$$

The same rotation we have to perform on reflection line also point (0, 0) will remain as it is but point (6, 6) will become

$$16\ 61 * \begin{vmatrix} \cos\theta & \sin\theta \\ +\sin\theta & \cos\theta \end{vmatrix}$$

$$16\ 61 * \begin{vmatrix} 0.948 & -0.317 \\ 0.317 & 0.948 \end{vmatrix}$$

$$= 17.58 \quad 3.781$$

See Fig. P. 4.8.7(g).

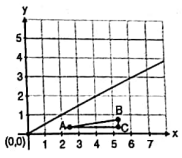


Fig. P. 4.8.7(g)

Step V : Now the last step i.e. we have to translate the reflection line and triangle back to original position i.e. in step 1 if we have translated a line and triangle by – 2 units in y-direction then now we have to translate iny-direction by + 2 units.

So, $$\begin{vmatrix} 2.79 & 0.392 & 1 \\ 5.58 & 0.78 & 1 \\ 5.58 & 0.41 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & +2 & 1 \end{vmatrix} = \begin{vmatrix} 2.79 & 2.392 & 1 \\ 5.58 & 2.78 & 1 \\ 5.58 & 2.41 & 1 \end{vmatrix}$$

and the lines co-ordinates become $(x_1, y_1) = (0, 0)$

$= (0), (0 + 2) = (0,2)$

and $(x_2, y_2) = 7.58, 3.78 = (7.58), (3.75 + 2)$

$= (7.58, 5.75)$

∴ The final pictorial representation will be as shown in Fig. P. 4.8.7(h).

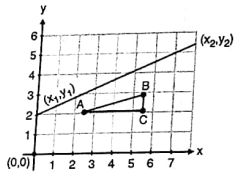


Fig. P. 4.8.7(h)

Example 4.8.8

Consider line A (2, 3), B (8, 10). Obtain the co-ordinates of transformed line using $[T] = \begin{vmatrix} 4 & 0 \\ 0 & 1 \end{vmatrix}$

Solution : Line AB in matrix form can be represented as,

$$\therefore \begin{vmatrix} 2 & 3 \\ 8 & 10 \end{vmatrix}$$

Now apply the given transformation on this line.

$$\therefore \begin{vmatrix} 2 & 3 \\ 8 & 10 \end{vmatrix} * \begin{vmatrix} 4 & 0 \\ 0 & 1 \end{vmatrix} = \begin{vmatrix} 8 & 3 \\ 32 & 10 \end{vmatrix}$$

∴ The transformed line co-ordinates will be A(8, 3) and B(32, 10).

Example 4.8.9

Find out final co-ordinate of a figure bounded by co-ordinates A (2,1), B(2,3), C(4,2), D(4,2) with scale factor Sx = Sy = 3.

Solution : The given data in matrix form it will be ;

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

Now scale this by scaling matrix. For the scaling Sx = Sy = 3 is given.

$$\therefore \begin{vmatrix} 2 & 1 \\ 2 & 3 \\ 4 & 2 \\ 4 & 2 \end{vmatrix} * \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix}$$

$$\therefore \begin{vmatrix} 2 & 1 \\ 2 & 3 \\ 4 & 2 \\ 4 & 2 \end{vmatrix} * \begin{vmatrix} 3 & 0 \\ 0 & 3 \end{vmatrix} = \begin{vmatrix} 6 & 3 \\ 6 & 9 \\ 12 & 6 \\ 12 & 6 \end{vmatrix}$$

∴ The final co-ordinates of the figure will be A(6,3), B(6,9), C(12,6) and D(12,6).

Example 4.8.10

Consider the object ABC with co-ordinate A(1,1), B(10,1), C(5,5). Rotate object with 90° in counter clockwise direction, give the co-ordinates of transformed object.

Solution : First draw the object ABC as shown in Fig. P. 4.8.10(a).

The matrix representation of object ABC will be,

$$\begin{vmatrix} 1 & 1 \\ 10 & 1 \\ 5 & 5 \end{vmatrix}$$

Now the object should be rotated with 90° in counter clockwise direction.

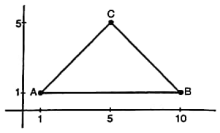


Fig. P. 4.8.10(a)

The matrix for counter clockwise rotation is,

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$ Where θ is now 90°

∴ Multiply the ABC object matrix's with the counter clockwise rotation matrix.

$$\therefore \begin{vmatrix} 1 & 1 \\ 10 & 1 \\ 5 & 5 \end{vmatrix} * \begin{vmatrix} \cos 90 & \sin 90 \\ -\sin 90 & \cos 90 \end{vmatrix}$$

$$= \begin{vmatrix} 1 & 1 \\ 10 & 1 \\ 5 & 5 \end{vmatrix} * \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} = \begin{vmatrix} -1 & 1 \\ -1 & 10 \\ -5 & 5 \end{vmatrix}$$

Now draw the final object ABC after performing the transformation. See Fig. P. 4.8.10(b).

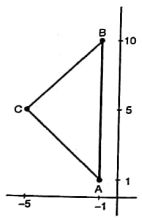


Fig. P. 4.8.10(b)

Example 4.8.11

Covert the unit square to shifted parallelogram using x-direction shear transformation operation where parameter $sh_x = \frac{1}{2}$ and $Y_{ref} = -1$ and unit square dimensions are (0, 0), (1, 0), (0, 1) and (1, 1).

Solution :

For the given problem we have to apply X-shear transformation

Lets draw the square first as shown in Fig. P. 4.8.11(a).

Here shearing factor $Sh_x = 0.5$

$$\begin{vmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 \\ Sh_x & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{vmatrix} * \begin{vmatrix} 1 & 0 \\ 0.5 & 1 \end{vmatrix}$$

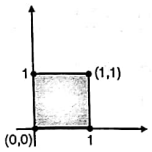


Fig. P. 4.8.11(a)

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

Now let's draw the resultant parallelogram as shown in Fig. P. 4.8.11(b).

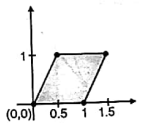


Fig. P. 4.8.11(b)

Example 4.8.12

Prove that a shear transform can be expressed in terms of rotation and scaling operations.

Solution :

We know that a shear matrix in homogeneous co-ordinate system is,

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

When value of a = 0 it behaves like shearing in y-direction and when value of b = 0 then it will be shearing in x-direction.

We know that the rotation matrix is

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

Scaling matrix is

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

We combine this rotation and scaling matrix

$$\therefore \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} S_x\cos\theta & S_x\sin\theta & 0 \\ -S_y\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Now if we compare this combined matrix with shearing matrix, we will get

$$\begin{vmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} S_x\cos\theta & S_x\sin\theta & 0 \\ -S_y\sin\theta & S_y\cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$\therefore \quad 1 = S_x\cos\theta$$

$$a = S_x\sin\theta$$

$$b = -S_y\sin\theta$$

$$1 = S_y\cos\theta$$

From above we can say

$$S_x = \frac{1}{\cos\theta} \quad \text{and} \quad S_y = \frac{1}{\cos\theta}$$

Substituting value of S_x and S_y we get,

$$a = \frac{1}{\cos\theta}\sin\theta = \tan\theta$$

and $$b = \frac{-1}{\cos\theta}\sin\theta = -\tan\theta$$

Now the shearing transformation can be expressed in terms of rotation and scaling as

$$\begin{vmatrix} 1 & \tan\theta & 0 \\ -\tan\theta & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

where θ = angle of rotation and $1 = S_x\cos\theta = S_y\cos\theta$

Example 4.8.13

Consider the square P(0, 0), Q (0, 10), R(10, 10), S(10, 0). Rotate the square about fixed point R (10, 10) by an angle 45° (anticlockwise) followed by scaling by 2 units in X direction and 2 units in Y direction.

Solution :

Let's represent the square PQRS

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 0 & 10 & 1 \\ 10 & 10 & 1 \\ 10 & 0 & 1 \end{vmatrix}$$

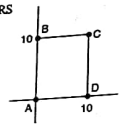


Fig. P. 4.8.13(a)

Now to rotate square about a point (10, 10) we have to perform three steps i.e.

- Translation
- Rotation
- Translation

For first translation tx = – 10, ty = – 10

$$\begin{vmatrix} 0 & 0 & 1 \\ 0 & 10 & 1 \\ 10 & 10 & 1 \\ 10 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -10 & -10 & 1 \end{vmatrix} = \begin{vmatrix} -10 & -10 & 1 \\ -10 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & -10 & 1 \end{vmatrix}$$

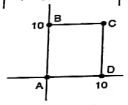


Fig. P. 4.8.13(b)

Now rotate by 45° in anticlockwise direction.

$$\begin{vmatrix} -10 & -10 & 1 \\ -10 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & -10 & 1 \end{vmatrix} * \begin{vmatrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$\therefore \begin{vmatrix} -10 & -10 & 1 \\ -10 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & -10 & 1 \end{vmatrix} * \begin{vmatrix} 0.7 & 0.7 & 0 \\ -0.7 & 0.7 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & -14 & 1 \\ -7 & -7 & 1 \\ 0 & 0 & 1 \\ 7 & -7 & 1 \end{vmatrix}$$

Now again translate by tx = 10, ty = 10

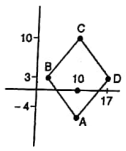


Fig. P. 4.8.13(c)

$$\therefore \begin{vmatrix} 0 & -14 & 1 \\ -7 & -7 & 1 \\ 0 & 0 & 1 \\ 7 & -7 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 10 & 10 & 1 \end{vmatrix} = \begin{vmatrix} 10 & -4 & 1 \\ 3 & 3 & 1 \\ 10 & 10 & 1 \\ 17 & 3 & 1 \end{vmatrix}$$

Now we have to scale this square by sx = 2 and sy = 2

$$\therefore \begin{vmatrix} 10 & -4 & 1 \\ 3 & 3 & 1 \\ 10 & 10 & 1 \\ 17 & 3 & 1 \end{vmatrix} * \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 20 & -8 & 1 \\ 6 & 6 & 1 \\ 20 & 20 & 1 \\ 34 & 6 & 1 \end{vmatrix}$$

∴ The final co-ordinates of square PQRS will be (20, – 8), (6, 6), (20, 20) and (34, 6).

Example 4.8.14

Find out coordinates of figure bounded by (0, 0), (1, 5), (6, 3), (– 3, – 4) column reflected along line whose equation is y = 2x + 4 and sheared by 2 units in x and 2 units in y-direction.

Solution :

Given :

Square (0, 0), (1, 5), (6, 3), (– 3, – 4) needs to reflect along the line y = 2x + 4 and then shared by 2 units in x and y direction.

Let's first represent the square in matrix form.

$$\begin{vmatrix} 0 & 0 & 1 \\ 1 & 5 & 1 \\ 6 & 3 & 1 \\ -3 & -4 & 1 \end{vmatrix}$$

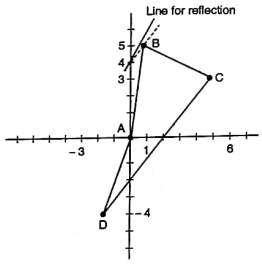


Fig. P. 4.8.14

The line y = 2x + 4

Means slope = 2 and y-intercept is 4.

But slope $= \frac{Dy}{Dx} = \frac{2}{1} = \frac{y_2 - y_1}{x_2 - x_1}$

∴ To draw a line we need two points.

1st point is y-intercept only i.e. x = 0, y = 4, we know the slope of line also. Now we have to decide another point for a line.

$$\therefore \frac{Dy}{Dx} = \frac{y_2 - y_1}{x_2 - x_1} \quad \therefore \frac{y_2 - 4}{x_2 - 0} = \frac{2}{1}$$

$$\therefore y_2 = 6 \text{ and } x_2 = 1$$

Hence a line will be from (0, 4) to (1, 6).

Now with respect to this line we need to take reflection of a square. We can achieve this reflection by performing following steps.

Step I : First we need to translate a line y = 2x + 4 to pass through the origin. This translation is application the square also.

Step II : Once a line passes through the origin then we need to perform rotation so that the line will match the co-ordinate axis.

Step III : Once a line matches to co-ordinate axis now we can perform reflection relative to this axis which will be nothing but reflection relative to the line only.

Step IV : Now we have to undo the rotation transformation which we have carried out for matching the co-ordinate axis.

Step V : Lastly we need to perform translation so that the line will be shifted to its original position.

After this we have to carry out shear transformation where shx = 2 and shy = 2. Here we have to first apply x-shear by

$$\begin{vmatrix} 1 & 0 \\ b & 1 \end{vmatrix} \text{ where } b = 2$$

And then y-shear by,

$$\begin{vmatrix} 1 & a \\ 0 & 1 \end{vmatrix} \text{ where } a = 2$$

Example 4.8.15 MU - May 2016, Dec. 2016, 5 Marks

Show that the composition of two successive rotation are additive i.e. $R(\theta_1).R(\theta_2) = R(\theta_1 + \theta_2)$.

Solution : Let's assume that $\theta_1 = \theta_2 = 90°$

It means that first we will perform rotation of θ_1 i.e. 90° in anticlockwise direction and then on that resultant rotation matrix we will perform again anticlockwise rotation by angle θ_2.

By performing two times of rotation by θ_1 and θ_2 we will get results as if rotation by $(\theta_1 + \theta_2)$.

$$R(\theta_1) = \begin{vmatrix} \cos 90 & \sin 90 \\ -\sin 90 & \cos 90 \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

$$R(\theta_2) = \begin{vmatrix} \cos 90 & \sin 90 \\ -\sin 90 & \cos 90 \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

Now if $R(\theta_1) \cdot R(\theta_2)$

$$\text{Then, } \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} \cdot \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix} = \begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$$

Now, $R(\theta_1 + \theta_2)$ i.e. $R(90° + 90°) = R(180°)$

$$\therefore R(180°) = \begin{vmatrix} \cos 180 & \sin 180 \\ -\sin 180 & \cos 180 \end{vmatrix} = \begin{vmatrix} -1 & 0 \\ 0 & -1 \end{vmatrix}$$

Hence, $R(\theta_1) \cdot R(\theta_2) = R(\theta_1 + \theta_2)$

Ex. 4.8.16

Write transformation matrix for Scaling and Rotation and scale the polygon with co-ordinates A (4, 5), B(8, 10) and C(8, 2) by 2 units in x-direction and 3 units in y-direction. Find the transformed A, B and C points.

Soln. :

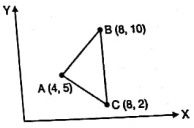
Transformation matrix for scaling is

$$\begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix}$$ where S_x is scaling factor along x axis and S_y is scaling factor along y axis

Transformation matrix for rotation is

$$\begin{vmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{vmatrix}$$ for clockwise rotation

As per problem definition given pollygon's co-ordinates are A(4, 5), B(8, 10) and C(8,2)



Now represent this polygon in matrix form as

$$\begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 8 & 2 \end{bmatrix}$$

Now we need to scale this polygon by 2 units in x-direction i.e. $S_x = 2$ and 3 units in y-direction i.e. $S_y = 3$.

$$\therefore \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 8 & 2 \end{bmatrix} * \begin{vmatrix} S_x & 0 \\ 0 & S_y \end{vmatrix}$$

$$\therefore \begin{bmatrix} 4 & 5 \\ 8 & 10 \\ 8 & 2 \end{bmatrix} \times \begin{vmatrix} 2 & 0 \\ 0 & 3 \end{vmatrix} = \begin{bmatrix} 8 & 15 \\ 16 & 30 \\ 16 & 6 \end{bmatrix}$$

∴ Final co-ordinates of ABC will be A(8, 15), B(16, 30) and (16, 6).

4.9 Important Questions and Answers

Q. 1 What is the relationship between the rotations R_θ, $R_{-\theta}$, and R_θ^{-1}?

Ans. :

R_θ means clockwise rotation with the angle theta.

$R_{-\theta}$ means clockwise rotation with the negative angle theta.

R_θ^{-1} means clockwise rotation with inverse i.e. Anticlockwise rotation with angle theta. Also see (Section 5.3.2).

Q. 2 Write an algorithm to continuously rotate an object about a point. Small angles may to be used for each successive rotation.

Ans. :

Assume that the reference point is origin, so that we can apply our normal rotation matrices for rotating the object.

1. Since we want the object to be rotated continuously, we have to be in a non ending loop.
2. Initially we need to display the object at its initial stage.
3. Now perform rotation of the object with some constant angle, say 5° in anticlockwise or clockwise direction.
4. For this we need to multiply the object with rotation matrix and get the new Co-ordinated from this.
5. Now again plot the new Co-ordinates, which will be 5° rotation of an object in anticlockwise direction.
6. Repeat the same procedure till angle becomes 360°. Once the angle becomes 360°again reinitialize the angle to 5° and go to step 3.

By performing the above steps we will achieve the same effect as if the object is continuously rotated in anticlockwise direction.

Q. 3 Give the transformation matrix for rotating a point P(x,y), about X-axis by 90 degrees.

Ans. :

The transformation matrix for rotating a point P(x, y) about X axis in Anticlockwise by 90 degree is

∴ The matrix is

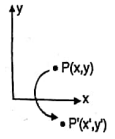


Fig. 1

$$\begin{vmatrix} Cos\,90 & Sin\,90 \\ -Sin\,90 & Cos\,90 \end{vmatrix} = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

Similarly for clockwise direction rotation, the matrix will be

$$\therefore \begin{vmatrix} Cos\,90 & -Sin\,90 \\ Sin\,90 & Cos\,90 \end{vmatrix}$$

$$\therefore \begin{vmatrix} 0 & -1 \\ 1 & 0 \end{vmatrix}$$

Q. 4 Prove that the multiplication of 2D transformation matrices for each of the following sequence of operations is commutative.

1. Two successive rotations.
2. Two successive translations.

OR

Q. 5 Prove that :

(i) 2D rotation and translation commute. **(5 Marks)**

Ans. :

In mathematics, a binary operation is commutative if changing the order of the operands does not change the result.

1. For two successive rotations

If we do rotation of a point P(x, y) by first 45° anticlockwise and then 60° anticlockwise the result which we are going to get is effectively rotation of (45° + 60° = 105°) anticlockwise.

Now suppose we are rotating a point P(x, y) by first 60° in anticlockwise direction and then 45° in anticlockwise direction, still the result which we are getting is same as that of the earlier case i.e. effectively (60° + 45° = 105°) anticlockwise.

If point (x, y) is (1, 1) then

$$|1\ 1| * \begin{vmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{vmatrix} = |0.707 - 0.707, 0.707 + 0.707|$$

$$= |0, 1.414|$$

Now this point (0, 1.414) is rotated by 60° in anticlockwise direction, then we get,

$$|0, 1.414| * \begin{vmatrix} \cos 60 & \sin 60 \\ -\sin 60 & \cos 60 \end{vmatrix} = 0 * 0.5 - 1.414 * 0.866,\ 0 * 0.866 + 1.414 * 0.5|$$

$$= |-1.22, 0.707|$$

Now this point (– 1.22. 0.707) is 45° + 60° = 105° anticlockwise rotated point.

Now let's consider another scenario where first we will rotate a point (1, 1) by 60° and then by 45°.

$$\therefore |1, 1| * \begin{vmatrix} \cos 60 & \sin 60 \\ -\sin 60 & \cos 60 \end{vmatrix} = |1 * 0.5 - 1 * 0.866,\ 1 * 0.866 + 1 * 0.5|$$

$$= |-0.366, 1.366|$$

Now this point (– 0.366, 1.366) is rotated by 45°

$$\therefore |-0.366, 1.366| * \begin{vmatrix} \cos 45 & \sin 45 \\ -\sin 45 & \cos 45 \end{vmatrix}$$

$$= |(-0.366 * 0.707) - (1.366 * 0.707), (-0.366 * 0.707) + (1.366 * 0.707)|$$

$$= |-0.258 - 0.965, -0.258 + 0.965|$$

$$= |-1.22, 0.707|$$

Hence it is proved that two successive rotations are commutative.

2. For two successive translations

Similarly we can do for two successive translations also. If we translate a point first by 2 units in X direction and then 3 units in X direction effectively we are translating a point by (2 + 3 = 5) units in X direction.

And if we translate the original point first by 3 units in X direction and then 2 units in X direction then also effectively we are translating a point by (3 + 2 = 5) units in X direction.

Hence it is proved that two successive translations are also commutative.

Q. 6 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.

Ans. :

Lets represent the given triangle and line.

Since the equation of line is y = x + 2 means slope m = 1 and y intercept = 2

$$\text{As slope} = \frac{dy}{dx}$$

$$\therefore m = \frac{dy}{dx}$$

$$\text{but } m = 1$$

$$\therefore dy = dx$$

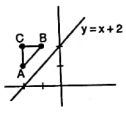


Fig. 2

It means a line is a slanting line making 45° angle. Since y-intercept is 2. One point is (0,2) which is drawn in the Fig. 2.

Now we have to reflect the triangle along the given line. But for that we have to first shift or translate a line so that a y intercept point i.e. (0,2) will become (0,0). Then only we can apply our standard reflection matrix.

$$\therefore \begin{vmatrix} -2 & 1 & 1 \\ -1 & 2 & 1 \\ -2 & 2 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{vmatrix} = \begin{vmatrix} -2 & -1 & 1 \\ -1 & 0 & 1 \\ -2 & 0 & 1 \end{vmatrix}$$

After this translation the line and triangle will be as shown in Fig. 3.

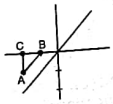


Fig. 3

Now we can perform reflection by using reflection at y = x.

$$\therefore \begin{vmatrix} -2 & -1 & 1 \\ -1 & 0 & 1 \\ -2 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} -1 & -2 & 1 \\ 0 & -1 & 1 \\ 0 & -2 & 1 \end{vmatrix}$$

Now we got the triangle as shown in Fig. 4.

But now we need to undo the translation which we have done at the start.

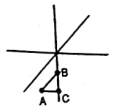


Fig. 4

$$\therefore \begin{vmatrix} -1 & -2 & 1 \\ 0 & -1 & 1 \\ 0 & -2 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{vmatrix} = \begin{vmatrix} -1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{vmatrix}$$

∴ The final co-ordinates of a reflected triangle are A (–1, 0), B(0, 1), and C(0, 0) which is represented as shown in Fig. 5

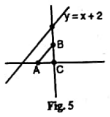


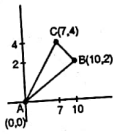
Fig. 5

Q. 7 Rotate a triangle ABC by an angle 30° where the triangle has co-ordinates A (0, 0), B (10, 2) and C (7, 4)

Ans. :

Let's us first draw the triangle ABC and represent it in matrix form.

The given co-ordinates of triangle ABC in matrix form are



$$\begin{vmatrix} 0 & 0 \\ 10 & 2 \\ 7 & 4 \end{vmatrix}$$

Fig. 6

Now we have to apply rotation by 30°. Since angle is positive, the rotation will be in anticlockwise direction.

$$\therefore \begin{vmatrix} 0 & 0 \\ 10 & 2 \\ 7 & 4 \end{vmatrix} \times \begin{vmatrix} \cos 30 & \sin 30 \\ -\sin 30 & \cos 30 \end{vmatrix}$$

$$= \begin{vmatrix} 0 & 0 \\ 10 & 2 \\ 7 & 4 \end{vmatrix} \times \begin{vmatrix} 0.86 & 0.5 \\ -0.5 & 0.86 \end{vmatrix} = \begin{vmatrix} 0 & 0 \\ 7.6 & 6.7 \\ 4.02 & 6.94 \end{vmatrix}$$

∴ After rotating triangle ABC by 30° we will get A (0, 0) B (7.6, 6.7) and C (4.02, 6.94)

Q. 8 Develop a single transformation matrix which does the following on given object :

(i) Reduces the size by 1/2

(ii) Rotates across Y axis by (– 30°)

Ans. :

First we have to reduces the size by 1/2.

It means we need to apply scaling by 0.5 in each direction.

$$\therefore \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Then rotate about y axis by (– 30°)

Since angle is negative the direction of rotation is anticlock wise only.

$$\therefore \begin{vmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} 0.86 & 0.5 & 0 \\ -0.5 & 0.86 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Combining scaling of rotation matrix we will get

$$\begin{vmatrix} 1/2 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 1 \end{vmatrix} \times \begin{vmatrix} 0.86 & 0.5 & 0 \\ -0.5 & 0.86 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.43 & 0.25 & 0 \\ -0.25 & 0.43 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Q. 9 Find the sequence of transformations to scale the object with respect to reference point A in XY plane. **(5 Marks)**

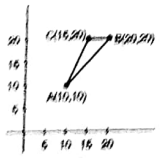


Fig. 7

Ans. :

The solution of this problem is we have to scale the given object. After scaling the coordinates of all points will get change. But we want to keep point A as same.

Since the scaling parameters s_x and s_y are not given in problem definition lets assume it as $s_x = s_y = 2$. So scaling matrix becomes

$$\begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

The scaling matrix is with respect to origin and we want to scale with respect to point A (10, 10) so we need the following sequence of transformations.

1. Translation 2. Scaling 3. Translation.

Step 1 : We have to translate the object in such a way that point A will become origin So we need a matrix,

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -10 & -10 & 1 \end{vmatrix} \text{ So } \begin{vmatrix} 10 & 10 & 1 \\ 20 & 20 & 1 \\ 15 & 20 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -10 & -10 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ 10 & 10 & 1 \\ 5 & 10 & 1 \end{vmatrix}$$

Now let's plot this

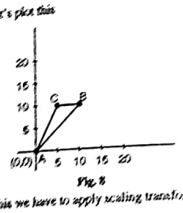


Fig. 8

After this we have to apply scaling transformation.

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 10 & 10 & 1 \\ 5 & 10 & 1 \end{vmatrix} \times \begin{vmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 \\ 20 & 20 & 1 \\ 10 & 20 & 1 \end{vmatrix}$$

Now lets plot this scale object as shown in Fig. 9

Now we have to perform translation again so that we will place point A at its original position (10, 10)

$$\therefore \begin{vmatrix} 0 & 0 & 1 \\ 20 & 20 & 1 \\ 10 & 20 & 1 \end{vmatrix} \times \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 10 & 10 & 1 \end{vmatrix} = \begin{vmatrix} 10 & 10 & 1 \\ 30 & 30 & 1 \\ 20 & 30 & 1 \end{vmatrix}$$

Now let's plot the final object which is scaled with respect to point A.

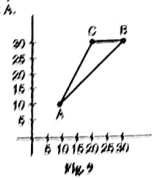


Fig. 9

4.10 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Basic Transformations – Translation, Scaling, Rotation, Raster Method for Transformation**

Q. Draw matrix for representing following operation : Scaling. *(Refer Section 4.3.1)* **(5 Marks)**

Q. Draw matrix for representing following operation : Rotation. *(Refer Section 4.3.2)* **(5 Marks)**

Q. Draw matrix for representing following operation : Translation. *(Refer Section 4.3.3)* **(5 Marks)**

☞ **Syllabus Topic : Homogeneous Co-ordinates**

Q. Derive matrices for Rotation about an arbitrary point. *(Refer Section 4.5)* **(May 2014, Dec. 2014, May 2017, 10 Marks)**

Q. Explain the steps used in rotation of 2-D object an arbitrary axis and derive the matrices for same. *(Refer Section 4.5)* **(May 2015, Dec. 2015, May 2016, 10 Marks)**

Q. Write short note on : Composite transformation. *(Refer Section 4.5)* **(May 2015, 5 Marks)**

Q. Derive 2D transformation matrix, for performing rotation of given point P(X, Y) by angle θ (theta) in anticlockwise direction about origin. *(Refer Section 4.5)* **(5 Marks)**

☞ **Syllabus Topic : Other Transformations – Reflection and Shear**

Q. Define shearing and give example. *(Refer Section 4.6.5)* **(May 2015, 5 Marks)**

Q. Write short note on : Shearing transformation. *(Refer Section 4.6.5)* **(May 2016, 10 Marks)**

Example 4.6.7 **(May 2013, 10 Marks)**

Example 4.8.15 **(May 2016, Dec. 2016, 5 Marks)**

□□□

Module 4

CHAPTER 5

2D Viewing and Clipping

Section Nos.	Name of the Topic
5.1	Introduction
5.2	Window to Viewport Transformation
5.3	Clipping

This chapter defines the term window and viewport as applied to computer graphics and describes how to clip away portions of a two dimensional picture so as to fit it within a window. This chapter has mainly three sections.

5.1 **Introduction** sets the background by defining a window, viewport and clipping.

5.2 **Window to Viewport Transformation** explains mapping of a rectangular portion of the scene, i.e. window onto a rectangular portion of the computer screen, i.e. viewport. This mapping requires a transformation between co-ordinate systems.

5.3 **Clipping** is a systematic procedure for "not drawing" the parts of objects that lie outside the window. We describe two algorithms in detail for line clipping. We extend these to polygon clipping. We also overview the different processes for clipping text and clipping circle.

5.1 Introduction

→ (May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017)

Q. Define window.
MU - May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 2 Marks

Q. What is Window? (4 Marks)

- A **window** is very familiar to all of us. It is just a rectangular portion through which we can see scene. It is an opening through which we see a portion of the world that exist outside. Normally when we are looking from window some portion of the scene is visible to us i.e. It may be buildings, some trees, roads, etc. But we have the limitation of the scene specified by the boundary of the window. So the scene gets cut off by the boundary of the window and we can see only part of the scene. It does not mean that whatever we have seen that part only exists and other does not exist. The other part also exists but we are not able to see that. It means we are limiting our view with an artificial device, window.
- The idea of window is important in computer graphics. Suppose some complex drawing or maps are drawn on computer, say Road map of Maharashtra state. And we want to read the road map for Pune city. Then it will be very difficult to read the map. So in that case it will be useful to display only that part of the drawing in which we are interested. So this gives the effect of looking at image through window. That means we are limiting our scope. Similarly if we want to see the road map of Nasik city then we have to concentrate on that part of map. So by changing the position of the window we can select other portion of the drawing for visualization. The method of selecting and enlarging the portions of a drawing is called **windowing**.
- We cannot limit what we want to see on computer screen just by drawing window frame. Instead, we must "not draw" that part of total picture which we want to suppress. We all know photographers camera. When we look through the camera towards some scene, we are able to see the scene, but in addition to that scene we see a rectangular area also. The final photograph gets developed with the scene which is inside the rectangular area. It does not mean that we should not see the image which is outside the rectangular area. So in other words we are not drawing the portion of scene which is outside the rectangular area. Determining which portion to omit or suppress is called **clipping**. Clipping eliminates the objects or portions of objects which are not visible through window.

Syllabus Topic : Viewing Transformation Pipeline and Window to Viewport Co-Ordinate Transformation

5.2 Window to Viewport Transformation

→ (May 2013, May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, May 2017)

Q. Differentiate between Object space and Image Space. MU - May 2013, 10 Marks

Q. Define view port and derive window to view port transformation. MU - May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 10 Marks.

Q. What is viewing transformation? MU - May 2015, 5 Marks

Q. Write short note on : Viewing transformation. MU - May 2016, 10 Marks

Q. What is viewport? Assume Window and viewport are rectangular. Derive the formulas required for transforming a point (x_w, y_w) in a window to point (x_v, y_v) in viewport. (5 Marks)

Q. Give the steps along with the transformation matrix for the mapping of the 2-D object from window to viewport. (10 Marks)

- Whenever we want to display any scene, we are considering two models of that scene. They are object model and image model. When we are saying object, we are referring a model of object which is stored in computer with the actual scale. It may be millimeter or in Kilometers. The space where object model reside is called **object space**. Image model is one which appears on display device. The image which we want to draw on display device must be measured in screen co-ordinates. See Fig. 5.2.1.
- There are different types of display devices with variations of screen co-ordinates. So to make screen coordinates device independent we have to use normalized co-ordinates. We must convert the object space units to image space co-ordinates.

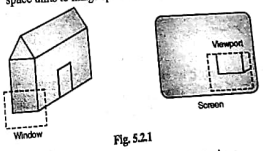


Fig. 5.2.1

- For this we have to use scaling transformations.
- Before proceeding with transformation, we will first see another term which is called as viewport. It may happen that we may not want to use entire screen for display. We just want some part of screen to display the image. So we will form a rectangular box on screen and in that box only we will display the image. This box is called as **viewport**. See Fig. 5.2.2.

- The object space contains the dimensions as actual which is called as world **co-ordinate system.**

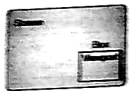


Fig. 5.2.2

- Now let us revise all the three terms. We can say, a world co-ordinate area selected for display is called ***window***.
- An area on a display device to which a window is mapped is called ***viewport***.
- So the window defines what is to be viewed, clipping means what to omit and viewport defines where it is to be displayed on display device. See Fig. 5.2.3.

Fig. 5.2.3

- If we are changing the position of window by keeping the viewport location constant, then the different part of the object is displayed at the same position on the display device. See Fig. 5.2.4. As the position of viewport is same, on screen only, the contents will get changed.

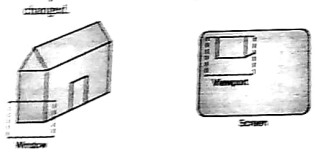


Fig. 5.2.4

- Similarly if we change the location of viewport then we will see the same part of the object drawn at different places on screen See Fig. 5.2.5.

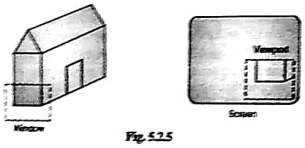


Fig. 5.2.5

- In general the mapping of a part of world co-ordinate scene to device co-ordinates is referred as **viewing transformation**. Sometimes it is also called as *window to viewport transformation* or *windowing transformation*. This viewing transformation is a process of three steps.

Step 1 : The object with its window is shifted or translated until the lower left corner of the window matches to the origin. See Fig. 5.2.6.

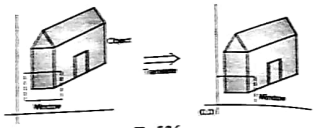


Fig. 5.2.6

Step 2 : The object and the window are now scaled until the window has the dimension same as viewport. In other words we are converting the object into image and window in viewport. For this we have to use scaling. See Fig. 5.2.7.

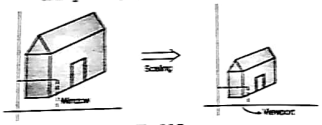


Fig. 5.2.7

Step 3 : Now perform another translation to move the viewport to its correct position on the screen. See Fig. 5.2. 8.

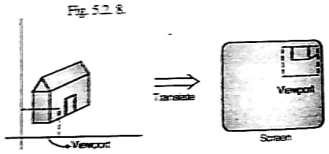


Fig. 5.2.8

- So the viewing transformation performs three steps :
  - Translation
  - Scaling
  - Translation.
- In reality we are using scaling to map window to viewport and translation to place viewport properly on screen.
- Let us take an example to derive the matrix for this transformation.

Example 5.2.1

Suppose there is a rectangle ABCD whose co-ordinates are A(1, 1), B(4, 1), C(4, 4), D(1, 4). And the window co-ordinates are (2, 2), (5, 2), (5, 5), (2, 5). And the given viewport location is (0.5, 0), (1, 0), (1, 0.5), (0.5, 0.5). Derive the viewing transformation matrix.

Solution :

Let us first draw a diagram of rectangle ABCD and window. See Fig. P. 5.2.1.

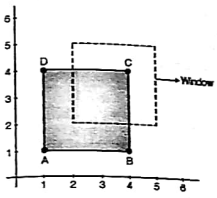


Fig. P. 5.2.1

- For viewing transformation we have to perform three steps.
- First step is **translation.**
- Here we have to shift lower left corner of window to origin. So the translation matrix will be.

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

- Second step is **scaling.**
- Here we have to scale window to viewport. i.e. old dimensions will be of windows dimensions and new will be of viewports.
- For scaling in x-direction S_x will be,

$$S_x = \frac{\text{new}}{\text{old}} = \frac{\text{width of view port}}{\text{width of window}}$$

$$= \frac{1-0.5}{5-2} = \frac{0.5}{3}$$

$$S_x = 0.16$$

Similarly, $$S_y = \frac{\text{height of viewport}}{\text{height of window}}$$

$$= \frac{1-0.5}{5-2} = \frac{0.5}{3}$$

$$S_y = 0.16$$

∴ Scaling matrix will be,

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

Third step is **translation.**

- Here we have to translate the viewport to the desired location on screen. So the translation matrix will be,

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

- So the overall transformation matrix will be the multiplication of translation matrix with respect to window, Scaling matrix and translation matrix with respect to viewport; which will be

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -2 & 1 \end{vmatrix} * \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0.5 & 0 & 1 \end{vmatrix}$$

Translation **Scaling** **Translation**

∴ Final viewing transformation matrix

$$= \begin{vmatrix} 0.16 & 0 & 0 \\ 0 & 0.16 & 0 \\ -0.16 & -0.32 & 1 \end{vmatrix}$$

- If we are multiplying the rectangle ABCD with this matrix then we will get desired output at specified position on screen.
- Let us generalize this matrix. Give the symbol W for window co-ordinates and V for viewport co-ordinates. See Fig. 5.2.9.

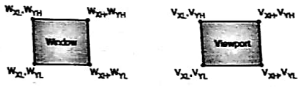


Fig. 5.2.9

- In Fig. 5.2.9 W_{XL}, W_{YL} represents windows lower left corner and V_{XL}, V_{YL} represents viewports lower left corner.
- So, the matrix for translation with respect to window will be,

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{XL} & -W_{YL} & 1 \end{vmatrix}$$

The matrix for scaling will be,

$$\begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

where, $$S_x = \frac{\text{width of viewport}}{\text{width of window}}$$

$$= \frac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}}$$

Similarly, $$S_y = \frac{\text{height of viewport}}{\text{height of window}}$$

$$\therefore S_y = \frac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}}$$

∴ Scaling matrix will be,

$$\begin{vmatrix} \frac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}} & 0 & 0 \\ 0 & \frac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}} & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

- Like that the translation matrix with respect to viewport will be,

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{XL} & V_{YL} & 1 \end{vmatrix}$$

∴ Collective generalized viewing transformation matrix will be,

$$= \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -W_{XL} & -W_{YL} & 1 \end{vmatrix} \cdot \begin{vmatrix} \frac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}} & 0 & 0 \\ 0 & \frac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}} & 0 \\ 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_{XL} & V_{YL} & 1 \end{vmatrix}$$

∴ If we solve this, we will get,

$$\begin{vmatrix} \frac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}} & 0 & 0 \\ 0 & \frac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}} & 0 \\ (V_{XL} - W_{XL}) \cdot \frac{V_{XH} - V_{XL}}{W_{XH} - W_{XL}} & (V_{YL} - W_{YL}) \cdot \frac{V_{YH} - V_{YL}}{W_{YH} - W_{YL}} & 1 \end{vmatrix}$$

- But if height and width of window is not in proportion with viewport then distortion occurs and the image may get elongated or minimized.

Syllabus Topic : Clipping operations – Point clipping , Line clipping algorithms : Cohen – Sutherland, Midpoint subdivision, Liang – Barsky , Polygon Clipping Algorithms : Sutherland – Hodgeman, Weiler – Atherton.

5.3 Clipping

➔ (Dec. 2015)

Q. State what is meant by clipping MU - Dec. 2015, 2 Marks

- **Clipping** means omitting the part of picture which lies outside the window and displaying the portion of picture which is inside the window. Clipping is easy for the human visual system, even with an imaginary window. We just do not see those parts of the world that lie outside the boundaries of the window. But for a computer it is slightly difficult. The computer must display the part of the image which is inside the window, in the viewport only. The images may be lines, polygons or text. There are different algorithms to clip different types of images.
- We know that a point is the smallest part on the screen which we can control. We cannot further subdivide the point into two or more parts. When we say **point clipping**, there are only two states: either the point is lying inside the window, which means we have to completely display it or the point is lying completely outside so we have to discard it. The point will not be partially visible and partially invisible.

5.3.1 Line Clipping

- It is simplest to clip a straight line. Fig. 5.3.1 shows lines of various types before clipping and after clipping.

Fig. 5.3.1

- Here we have to examine each and every line which we are going to display. We have to determine whether or not a line is completely inside the window, or lies completely outside the window, or crosses the boundary of window. If the line is completely inside then display it fully. If the line is totally outside then discard that line. But if the line is crossing to the window boundary, then we have to find the intersection point of the line with the edge of window and draw the portion which lies inside.

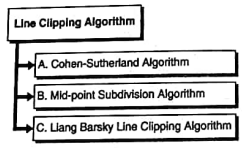


Fig.C5.1: Line Clipping Algorithm

➔ 5.3.1(A) Cohen-Sutherland Algorithm

➔ (Dec. 2015, May 2016, May 2017)

Q. State what is meant by clipping. Explain any one clipping algorithm. MU - Dec. 2015, 3 Marks

Q. Explain Cohen-Sutherland line clipping algorithm. MU - May 2016, May 2017, 10 Marks

Q. Give the line clipping algorithm which uses area codes for the end points of the line segment. (10 Marks)

- Cohen-Sutherland algorithm is one of the popular line clipping algorithm. This algorithm immediately removes the lines which are lying totally outside the window. This algorithm divides the plane in nine parts and assigns outcode or binary number to each part. See Fig. 5.3.2.

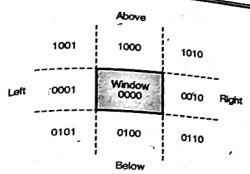


Fig. 5.3.2

- End point of each line is assigned a four bit binary code which is called as outcode. We can give abbreviated name to this four bit code as, "ABRL",

  Where, A = Above B = Below
  R = Right L = Left

- The highest bit among four digits i.e. 'A' will be set to 1 if the end point of a line is above the window. If the end point is not above the window then it is set to 0. Similarly if 'B' bit is set, means the end point is below the window. If the endpoint is inside the window then it is assigned a code as 0000. Zero to all four bits indicates that the end point is not above, not below, not right and not left of the window.

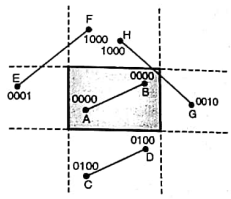


Fig. 5.3.3

- The Cohen-Sutherland algorithm tells that if the line is totally on one side of the window then immediately we have to discard that line. Similarly if the line is totally inside the window directly display that line without clipping. See Fig. 5.3.3.

Case 1 :

- Consider a line AB of Fig. 5.3.3. Here both end points of AB are surely inside the window. Therefore the outcode for end point A will be

Out code (A) = ABRL = 0000

Similarly, outcode (B) = ABRL = 0000

- The algorithm says that if the outcode's of both endpoints of a line are 0000. Then the line is fully visible. So we should not clip this line.

Case 2 :

- Consider a line CD of Fig. 5.3.3. Here endpoints of CD are surely outside the window. The outcode for endpoint C will be,

Outcode (C) = ABRL = 0100

Similarly, Outcode (D) = ABRL = 0100

- Now here for line CD, the outcode's of both endpoints of a line are not zero, so we have to perform logical AND operation of both outcode's.

Outcode (C) = 0100

Outcode (D) = 0100 ⊕ AND

= 0100

- If the result of AND operation is nonzero, then the line is surely outside the window. So, clip that line. Here line CD is clipped as its AND result is nonzero.

Case 3 :

- But if the logical AND result is zero, Then we cannot make firm statement about a line, whether it should be visible or not visible or partially visible. Consider line EF and GH of Fig. 5.3.3. For line EF outcode of E will be 0001 and outcode of F will be 1000. As both outcodes are not zero, so we have to perform logical AND.

E = 0001

AND F = 1000

= 0000

- The AND result is zero. It means the line EF is not lying completely on any one side of the window. Similarly for line GH,

Outcode (G) = 0010

Outcode (H) = 1000

Logical AND will be = 0000

- If we observe both lines GH and EF, the AND result of both lines is zero. But line GH is partially visible and line EF is fully invisible. So, for such lines we have to go one step ahead. Here we have to find out, to which edge of the window the line is intersecting and then we have to find the intersection point. To find out this we can make use of outcode.

Consider line EF. Outcode of E is 0001 and F is 1000.

Here we have to compare first bit of both outcodes.

E = 0001
↑
F = 1000
↑

- First bit of E is 1 and first bit of F is 0. That means point E of line EF is lying outside the left boundary of window. So we have to find out intersection point of line EF with left boundary of window. Call that intersection as P_1. See Fig. 5.3.4.

  Now we are having two lines E P_1 and P_1F.

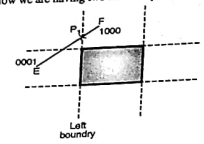


Fig. 5.3.4

- As point E is surely outside the left boundary of window so we are clipping line E P_1. Now we will concentrate on P_1F. Now we have to find outcode of P_1, which will be 1000. Here both outsides are nonzero i.e. P_1 = 1000 and F = 1000 so, logical AND will be nonzero i.e. 1000. It means line P_1F is lying completely on one side of window i.e. above the window, so discard the line. In this way we are eliminating whole line.

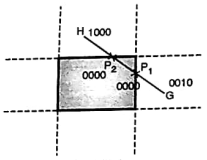


Fig. 5.3.5

- The same method is used for partially visible lines also such as GH.
- See Fig. 5.3.5. Here also both outcodes are nonzero so we have to take logical AND. The AND result will be zero. So we have to check individual bits of both outcodes.

  Outcode (G) = 0010
  ↑
  Outcode (H) = 1000
  ↑

- We will check first bit. Both outcodes first bit is zero. It means both points are not lieing outside the left boundary of window. It means no intersection. So we have to check next bits.

  Outcode (G) = 0010
  ↑
  Outcode (H) = 1000
  ↑

- Here second bit of G is 1. It means point G is outsides the right edge of window. So, find out intersection point with right edge of window, say point P_1. As point G is outside we have to discard line P_1G. Now our line will be P_1H. Outcode of P_1 will be 0000. Now compare next bits of H and P_1. Here as our line becomes P_1H we have to replace G by P_1.

  Outcode of (P_1) = 0000
  ↑
  Outcode of (H) = 1000
  ↑

  Both bits are zero so compare last bits of P_1 and H.

  Outcode of (P_1) = 0000
  ↑
  Outcode of (H) = 1000
  ↑

- Here point H is outside, so find intersection with top boundary. Call that point as P_2. As point H is surely outside (known from outcode), we have to discard line HP_2.
- And now our line becomes P_1 P_2. We have to find outcode of P_2, which will be 0000. Again compare outcode of P_1 and P_2. Both outcodes are zero. So line P_1P_2 is visible. So after clipping, the line GM will look like P_1P_2. See Fig. 5.3.6.

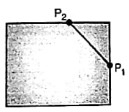


Fig. 5.3.6

- The important factor in this algorithm is to find out the intersection point with window edge. If we want to find intersection with left edge of the window, as shown in Fig. 5.3.7 then,
- We know the co-ordinates of window. Lower left corner of wind ow will be (x_L, y_L) and upper right corner will be (x_H, y_H). As we are interested in finding intersection with left edge only, the x co-ordinate of intersection point must be x_L i.e. the point must lie on left edge of window. Now we have to find y co-ordinate of intersection point. For this we will take help of line equation. See Fig. 5.3.7.

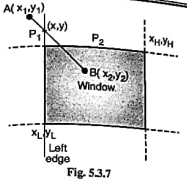


Fig. 5.3.7

For line AB the slope (m) = $(y_2 - y_1) / (x_2 - x_1)$

- Once the value of slope (m) is known, then we will find the slope between intersection point P_1 and any one end point of line AB, say A.

∴ Slope between line P_1 A = $(y - y_1) / (x - x_1)$

- But here the value of y is unknown and value of slope (m) is known.

$$\therefore \quad m = \frac{y - y_1}{x - x_1}$$

$$\therefore \quad y - y_1 = m(x - x_1) \quad \text{Here } x = x_L$$

$$\therefore \quad y = y_1 + m(x_L - x_1)$$

- by using above equation, we can find y co-ordinate for intersection point. So intersection points co-ordinates will be (x_L, y).
- Similarly we can find the intersection point with right edge of window. Here x co-ordinate will be x_H of window and y co-ordinate will be,

$$y = y_1 + m(x_R - x_1)$$

- Similarly for top boundary the intersection point will lie on top edge of window i.e. y_H as shown in Fig. 5.3.8 and its x co-ordinate will be calculated as,

$$m = \frac{y - y_1}{x - x_1}$$

$$x - x_1 = \frac{y - y_1}{m} \qquad \text{Here } y = y_H$$

$$\therefore \quad x = x_1 + \frac{y_H - y_1}{m}$$

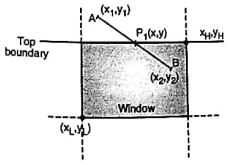


Fig. 5.3.8

- Similarly for bottom edge, y co-ordinate will be y_L and x co-ordinate will be,

$$x = x_1 + \frac{y_L - y_1}{m}$$

Cohen-Sutherland line clipping algorithm

1. Accept the window co-ordinates from user and store them in x_L, x_H, y_L, y_H.
2. Accept the endpoints A and B of line with components A_x, A_y, B_x, B_y.
3. Initialize A and B code as array of size four.
4. Calculate Acode and Bcode by comparing,
   if $A_x < x_L$ then Acode (4) = 1 else 0
   if $A_x > x_H$ then Acode (3) = 1 else 0
   if $A_y < y_L$ then Acode (2) = 1 else 0
   if $A_y > y_H$ then Acode (1) = 1 else 0
5. Check if Acode and Bcode are 0000. If yes, display line AB. If not then proceed.
6. Take logical AND of Acode and Bcode, check the result, if it is nonzero, discard the line and go to ⑧ else proceed.
7. Subdivide the line by finding intersection of a line with window boundary and give new outcodes to the intersection point and go to ⑤.
8. End.

```
/*****************************************
Program : A C++ program to implement Cohen - Sutherland
line clipping algorithm by using mouse
*****************************************/
#include <iostream.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>

typedef struct edge
{
    int xco,yco;
    int code[4];
}EDG;
typedef struct window
{
    int xc,yc;
}WIN;
int i,indx;
EDG ed[30];
WIN wi[4];
/*=============================
            .Class Definition.
_____________________________*/
```

```
class clipp
{
    public:
        int button,x,y;
        int edg_count;
        EDG edpt[50],edpt1[50];
    public:
        void accept_lines();
        void accept_window();
        void draw_window();
    void getmousepos(int *button,int *x,int *y);
        void initmouse();
        void showmouse();
        void init_graph();
        void cohen_suth(EDG ed1,EDG ed2);
        int dis_test(EDG ed,EDG ed1);
        EDG outcode(EDG ed1);
};
/*============================
Member Name :   outcode()
Purpose     :   To assign the four bit binary outcode
                for the endpoints.
parameter is :  An endpoint of a line
----------------------------------------------------------*/
EDG clipp::outcode(EDG edc)
{
    if(edc.xco<wi[2].xc)// checking for   left edge
        edc.code[3] = 1;
    else
        edc.code[3] = 0;
    if(edc.xco>wi[1].xc)// checking for right edge
        edc.code[2] = 1;
    else
        edc.code[2] = 0;
    if(edc.yco>wi[3].yc)//checking for below edge
        edc.code[1] = 1;
    else
        edc.code[1] = 0;
    if(edc.yco<wi[1].yc)// checking for above edge
        edc.code[0] = 1;
    else
        edc.code[0] = 0;
        return(edc);
}
/*============================
Member Name :   cohen_suth()
Purpose     :   To clip a given line with respect to a
                window by sing Cohen-sutherland
                algorithm.
parameters are :  Endpoints a line
----------------------------------------------------------*/
void clipp::cohen_suth(EDG ed1,EDG ed2)
{
        EDG edon,edtw;
        int d,flag,interx,intery;
        edon = outcode(ed1);
        edtw = outcode(ed2);
        d = dis_test(edon,edtw);
        switch(d)
        {
        case 0:             // totally inside
            setcolor(WHITE);
            line(edon.xco,edon.yco,edtw.xco,edtw.yco);
            break;
        case 1:             // Totally outside
                return;
        case 2:             // Partially inside
            next_inter:
            for(i=3;i>=0;i--)
            {
            if(edon.code[i]==0 && edtw.code[i]==0)
                    continue;
                else if(edtw.code[i]==1)
                goto new_code1;
                else
                goto new_code2;
            }
        goto drawline;
    new_code1:
        switch(i)
        {
        case 2:
            interx = wi[2].xc;
            intery =(((wi[2].xc-edon.xco)*edon.yco-
                    edtw.yco)) /
                    (edon.xco-edtw.xco))+edon.yco;
            break;

        case 3:
            interx = wi[1].xc;
            intery =(((wi[1].xc-edon.xco)*(edon.yco-
                    edtw.yco))/
                (edon.xco-edtw.xco))+edon.yco;
          break;
        case 0:
            intery = wi[1].yc;
interx =(((wi[1].yc-edon.yco)*(edon.xco-edtw.xco))/
                (don.yco-edtw.yco))+edon.xco;
            break;
        case 1:
            intery = wi[3].yc;
interx =(((wi[3].yc-edon.yco)*(edon.xco-edtw.xco))/
```



```
                (edon.yco-edtw.yco))+edon.xco;
            break;
        }
            edtw.xco = (interx);
            edtw.yco = (intery);
            edtw = outcode(edtw);
            goto next_inter;
    new_code2:
        switch(i)
        {
        case 2:
            interx = wi[2].xc;
intery =(((wi[2].xc-edtw.xco)*(edtw.yco-edon.yco))/
                (edtw.xco-edon.xco))+edtw.yco;
            break;
        case 3:
            interx = wi[1].xc;
intery =(((wi[1].xc-edtw.xco)*(edtw.yco-edon.yco))/
                (edtw.xco-edon.xco))+edtw.yco;
            break;
        case 0:
            intery = wi[1].yc;
interx =(((wi[1].yc-edtw.yco)*(edtw.xco-edon.xco))/
                (edtw.yco-edon.yco))+edtw.xco;
            break;
        case 1:
            intery = wi[3].yc;
interx =(((wi[3].yc-edtw.yco)*(edtw.xco-edon.xco))/
                (edtw.yco-edon.yco))+edtw.xco;
            break;
        }
            edon.xco = (interx);
            edon.yco = (intery);
            edon = outcode(edon);
            goto next_inter;
    }
    drawline:    //to draw the unclipped line.
    setcolor(WHITE);
    line(edon.xco,edon.yco,edtw.xco,edtw.yco);
}
/*============================
Member Name :   dis_test()
Purpose     :   To check whether a line is totally visible
                or totally invisible or partially visible.
parameters are :  Endpoints of a line.
----------------------------------------------------------*/
int clipp::dis_test(EDG ed,EDG ed1)
{
    int j;
    int temp = 0;
    EDG Etemp;
    temp = 0;
    for(j=0;j<4;j++)        //if both outcodes are 0000.
    {
    if(ed.code[j] == ed1.code[j])
            if(ed.code[j]==0)
                temp++;
    }
    if (temp==4)
            return(0);
    else
    {
            for(j=0;j<4;j++)
        {   //LOGICAL AND to check whether it is
            // completely invisible or partially visible.
      if(ed.code[j]==ed1.code[j]&&ed.code[j]==1&&ed1.co
de[j]==1)
            Etemp.code[j] = 1;
              else
            Etemp.code[j] = 0;
              }
              temp = 0;
              for(j=0;j<4;j++)
              {
      if(Etemp.code[j]==0)
                temp++;
              }
    }
    if(temp<4)              //completely invisible.
            return(1);
    else
            return 2;       //partially visible.
}
/*============================
Member Name :   init_graph()
Purpose     :   To initialise the graphics system.
----------------------------------------------------------*/
void clipp::init_graph()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
}
/*============================
Member Name :   accept_window()
Purpose     :   To accept the window from user by
                making use of mouse.
----------------------------------------------------------*/
void clipp::accept_window()
{
    outtextxy(70,90,"Enter Top Right and Bottom Left
Corners of the Window");
    button = x = y = 0;
```

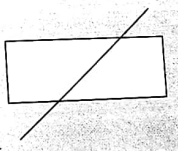
```
    i = 1;
    while(1)
    {
    getmousepos(&button,&x,&y);
    if(button==1)
    {
            delay(250);
            wi[i].xc = x;
            wi[i].yc = y;
            i+=2;
            if(i==5)
        break;
    }
    }
    wi[2].xc = wi[3].xc;
    wi[2].yc = wi[1].yc;
    wi[4].xc = wi[1].xc;
    wi[4].yc = wi[3].yc;
}
/*============================
Member Name :    draw_window()
Purpose     :    To display the window.
----------------------------------------------------------------*/
void clipp::draw_window()
{
        setcolor(WHITE);
        line(wi[1].xc,wi[1].yc,wi[2].xc,wi[2].yc);
        line(wi[2].xc,wi[2].yc,wi[3].xc,wi[3].yc);
        line(wi[3].xc,wi[3].yc,wi[4].xc,wi[4].yc);
        line(wi[4].xc,wi[4].yc,wi[1].xc,wi[1].yc);
        getch();
}
/*============================
Member Name :    accept_lines()
Purpose     :    To accept the lines from user by
                 making use of mouse.
----------------------------------------------------------------*/
void clipp::accept_lines()
{
    outtextxy(200,50,"Enter The Edges");
    outtextxy(100,70,"Right Click To Stop Entering The
Edges");
        x=y=button=i=edg_count=0;
        while(button!=2)
        {
            getmousepos(&button,&x,&y);
            if(button==1)
            {
                delay(250);
                i++;
                ed[i].xco = x;
                ed[i].yco = y;
                if((i%2)==0)
                    line(ed[i-1].xco,ed[i-
1].yco,ed[i].xco,ed[i].yco);
            }
        }
        edg_count = i;
}
/*============================
Member Name :    getmousepos()
Purpose     :    To get the pixel clicked by the mouse.
----------------------------------------------------------------*/
void clipp::getmousepos(int *button,int *x,int *y)
{
        int b1=0,xt,yt;
        asm{
            mov ax,3
            int 0x33
            mov b1,bx
            mov xt,cx
            mov yt,dx
            }
            *button = b1;
            *x = xt;
            *y = yt;
}
/*============================
Member Name :    initmouse()
Purpose     :    To initialize the mouse.
----------------------------------------------------------------*/
void clipp::initmouse()
{
        asm {
            mov ax,0
            int 0x33
            }
}
/*============================
Member Name :    showmouse()
Purpose     :    To show the mouse.
----------------------------------------------------------------*/
void clipp::showmouse()
{
        asm{
    mov ax,1
    int 0x33
            }
}
int main()
{
    int choice,count;
    clipp c;
```



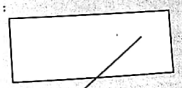
```
    clrscr();
    c.init_graph();
    c.initmouse();
    c.showmouse();
    c.edg_count = 0;
    outtextxy(100,10,"Cohen Sutherland Line Clipping
Algorithm");
    c.accept_lines();
    count = c.edg_count;
    c.accept_window();
    c.draw_window();
    cleardevice();
    outtextxy(150,90,"Clipped line will be :");
    c.draw_window();
    int i;
    for(i=1;i<count;i+=2)
    c.cohen_suth(ed[i],ed[i+1]);
    getch();
    closegraph();
    return 0;
}
/*======== End of Program ============*/
```

Output

```
Cohen Sutherland line clipping Algorithm
Enter The Edges
Right Click To Stop Entering The Edges
Enter Top Right and Bottom Left Corners of the Window
```



Clipped line will be :



➜ 5.3.1(B) Mid-point Subdivision Algorithm

Q. Explain Midpoint Subdivision Algorithm. (10 Marks)

- The only difficult part in Cohen-Sutherland algorithm is to find the intersection point of line with window boundary. So, we have to use different formulas for that. But we can avoid use of these formulas by using another method which is called as mid-point subdivision algorithm. In this algorithm we are using a binary search for the intersection point by always dividing the line at its mid point. We can call this algorithm as a special case of Cohen-Sutherland algorithm.
- The midpoint subdivision algorithm uses the same technique as Cohen-Sutherland. It uses the line endpoint outcodes and associated tests to immediately identify which lines are visible and which are invisible. If both the endpoints of a line has outcodes 0000 then the line is totally visible. If both the outcodes are not zero then we have to take logical AND of both outcodes and then we have to decide whether that line is visible or not. If the logical AND result is nonzero, it means the line is totally invisible. So discard that line. And if the logical AND result is zero, it means a line may be partially visible.

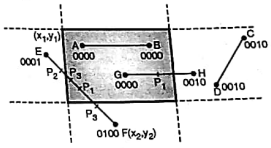


Fig. 5.3.9

- Consider line AB of Fig. 5.3.9. Here both endpoints of a line is having outcodes 0000. As both outcodes are 0000 so line is totally visible. But both outcodes of line CD are nonzero. So we have to take logical AND of its outcodes, which will be,

  Outcode (C) = 0010
  Outcode (D) = 0010
  Logical AND = 0010

- As the logical AND result is nonzero, line CD is totally invisible.
- Similarly if line is of type EF, then both outcodes are nonzero, so we are taking logical AND, the result of that will be,

  Outcode (E) = 0001
  Outcode (F) = 0100
  Logical AND = 0000

- Here the logical AND result is zero. So a line EF may be partially visible. Such lines are divided into two equal parts. And the tests are then applied to each half, until the intersection with the window edge is found or the length of divided segment becomes a point. And then the visibility of that line is determined.
- Refer line EF of Fig. 5.3.9. As both outcodes of EF line are nonzero, we have taken logical AND of that, which

is zero. As the logical AND result is zero, we have to divide line EF in two equal parts. Say the midpoint of EF line is P_1. Now P_1 will have co-ordinates as (x_m, y_m).

where $x_m = \frac{x_1 + x_2}{2}$ and $y_m = \frac{y_1 + y_2}{2}$

- Here (x_1, y_1) and (x_2, y_2) are the co-ordinates of line EF. Now point P_1 divides line EF into two parts, EP_1 and P_1F. Now we have to concentrate on line EP_1. Line PF will be considered later. We will find outcode of P_1 which will be 0000. Again the process begins. Here both outcodes of E and P_1 are not zero so we have to take logical AND.

Outcode (E) = 0001

Outcode (P_1) = 0000

Logical AND = 0000

- The AND result will be zero. So again divide line EP_1 into two parts EP_2 and P_2P_1, where P_2 will be midpoint of EP_1 line. Again find outcode of P_2, which will be 0001. Then we will concentrate on EP_2. Here both outcodes are not zero so we will take logical AND. The result of that will be

Outcode (E) = 0001

Outcode (P_2) = 0001

Logical AND = 0001

- Here for line EP_2, logical AND result is nonzero, so discard line EP_2. Now concentrate on line P_2P_1. Again continue the process till line becomes a point or till from logical test we come to certain decision i.e. whether line is visible or not.
- Same procedure should be applied to remaining half i.e. P_1F.
- If the line is of type GH (See Fig. 5.3.9), then after dividing this line into GP_1, and P_1H, we will come to know that for one of the part of the line, we can immediately take decision for visibility of line. Here line GP_1 is visible, so there is no need to further divide this line. We can directly concentrate on other half.

Program for Mid-point Subdivision Algorithm

```
/*********************************************
Program : A C++ program to implement mid-point line
clipping algorithm by using mouse
*********************************************/
#include <iostream.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>

typedef struct edge
{
    int xco,yco;
    int code[4];
}EDG;
typedef struct window
{
    int xc,yc;
}WIN;
int i,indx;
EDG ed[30];
WIN wi[4];
/*============================
        Class Definition.
============================*/
class clipp
{
    public:
        int button,x,y;
        int edg_count;
        EDG edpt[50],edpt1[50];
    public:
        void accept_lines();
        void accept_window();
        void draw_window();
    void getmousepos(int *button,int *x,int *y);
        void initmouse();
        void showmouse();
        void init_graph();
    void mpt_sub(EDG edo,EDG edt);
        int dis_test(EDG ed,EDG ed1);
        EDG outcode(EDG ed1);
};
/*============================
Member Name :   outcode()
Purpose     :   To assign the four bit binary outcode for
                the endpoints.
parameter is :  An endpoint of a line
============================*/
EDG clipp::outcode(EDG edc)
{
    if(edc.xco<wi[2].xc)// checking for left edge
        edc.code[3] = 1;
    else
        edc.code[3] = 0;
    if(edc.xco>wi[1].xc)// checking for right edge
        edc.code[2] = 1;
    else
        edc.code[2] = 0;
```



```
    if(edc.yco>wi[3].yc)// checking for below edge
        edc.code[1] = 1;
    else
        edc.code[1] = 0;
    if(edc.yco<wi[1].yc)// cheking for above edge
        edc.code[0] = 1;
    else
        edc.code[0] = 0;
    return(edc);
}
/*============================
Member Name :   mpt_sub().
Purpose     :   To clip a given line with respect to a
                window by using midpoint subdivision
                algorithm.
parameters are :  Endpoints a line
----------------------------*/
void clipp::mpt_sub(EDG edo,EDG edt)
{
    EDG midpt,edon,edtw;
    int d;
    edon = outcode(edo);
    edtw = outcode(edt);
    d = dis_test(edon,edtw);
    switch(d)
    {
     case 0:        //Totally inside setcolor(WHITE);
line(edon.xco,edon.yco,edtw.xco,edtw.yco);
            break;
     case 1:        // Totally outside
            break;
     case 2:        // partially inside
     midpt.xco = (edon.xco + edtw.xco)/2;
     midpt.yco = (edon.yco + edtw.yco)/2;
        mpt_sub(edon,midpt);
        midpt.yco = midpt.yco + 1;
        midpt.xco = midpt.xco + 1;
        mpt_sub(midpt,edtw);
            break;
    }
}
/*============================
Member Name :   dis_test()
Purpose     :   To check whether a line is totally visisble
                or totally invisible or partailly visible.
parameters are :  Endpoints of a line.
----------------------------*/
int clipp::dis_test(EDG ed,EDG ed1)
{   int j;
    int temp = 0;
    EDG Etemp;
    temp = 0;
    for(j=0;j<4;j++)    //if both outcodes are 0000.
    {
    if(ed.code[j] == ed1.code[j])
            if(ed.code[j]==0)
                temp++;
    }
    if (temp==4)
            return(0);
    else
    {
        for(j=0;j<4;j++)
    {   // LOGICAL AND to check whether it is completely
    // invisible or partially visible.
if(ed.code[j] == ed1.code[j] && ed.code[j]
==1&&ed1.code[j]==1)
            Etemp.code[j] = 1;
            else
            Etemp.code[j] = 0;
        }
        temp = 0;
        for(j=0;j<4;j++)
            {
            if(Etemp.code[j]==0)
            temp++;
            }
    }
    if(temp<4)          //completely invisible.
            return 1;
    else
    return 2;           //partially visible.
}
/*============================
Member Name :   init_graph().
Purpose     :   To initialise the graphics system.
----------------------------*/
void clipp::init_graph()
{
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
}
/*============================
Member Name     :   accept_window().
Purpose         :   To accept the window from user by
                    making use of mouse.
----------------------------*/
void clipp::accept_window()
{
    outtextxy(70,90,"Enter Top Right and Bottom Left
Corners of the Window");
```

```
        button = x = y = 0;
        i = 1;
        while(1)
{
    getmousepos(&button,&x,&y);
    if(button==1)
    {
            delay(250);
            wi[i].xc = x;
            wi[i].yc = y;
            i+=2;
            if(i==5)
        break;
    }
      }
      wi[2].xc = wi[3].xc;
      wi[2].yc = wi[1].yc;
      wi[4].xc = wi[1].xc;
      wi[4].yc = wi[3].yc;
}
/*============================
Member Name :   draw_window().
Purpose     :   To display the window.
----------------------------*/
void clipp::draw_window()
{
        setcolor(WHITE);
        line(wi[1].xc,wi[1].yc,wi[2].xc,wi[2].yc);
        line(wi[2].xc,wi[2].yc,wi[3].xc,wi[3].yc);
        line(wi[3].xc,wi[3].yc,wi[4].xc,wi[4].yc);
        line(wi[4].xc,wi[4].yc,wi[1].xc,wi[1].yc);
        getch();
}
/*============================
Member Name     :   accept_lines()
Purpose         :   To accept the lines from user by
                    making use of mouse.
----------------------------*/
void clipp::accept_lines()
{
    outtextxy(200,50,"Enter The Edges");
    outtextxy(100,70,"Right Click To Stop Entering The
Edges");
        x=y=button=i=edg_count=0;
        while(button!=2)
        {
    getmousepos(&button,&x,&y);
    if(button==1)
    {
            delay(250);
            i++;
            ed[i].xco = x;
            ed[i].yco = y;
            if((i%2)==0)
        line(ed[i-1].xco,ed[i-1].yco,ed[i].xco,ed[i].yco);
    }
        }
        edg_count = i;
}
/*============================
Member Name :   getmousepos()
Purpose     :   To get the pixel clicked by the mouse.
----------------------------*/
void clipp::getmousepos(int *button,int *x,int *y)
{
        int b1=0,xt,yt;
        asm{
            mov ax,3
            int 0x33
            mov b1,bx
            mov xt,cx
            mov yt,dx
            }
            *button = b1;
            *x = xt;
            *y = yt;
}
/*============================
Member Name :   initmouse()
Purpose     :   To initialize the mouse.
----------------------------*/
void clipp::initmouse()
{
        asm {
            mov ax,0
            int 0x33
            }
}
/*============================
    Member Name :   showmouse()
    Purpose     :   To show the mouse.
----------------------------*/
void clipp::showmouse()
{
    asm {
            mov ax,1
            int 0x33
        }
}
```



```
int main()
{
    int choice,count;
    clipp c;
    clrscr();
    c.init_graph();
    c.initmouse();
    c.showmouse();
    c.edg_count = 0;
outtextxy(150,10,"Midpoint Subdvision Algorithm");
    c.accept_lines();
    count = c.edg_count;
    c.accept_window();
    c.draw_window();
    cleardevice();
    outtextxy(150,70,"Clipped line will be :");
    c.draw_window();
    for(i=1;i<=(count);i+=2)
    c.mpt_sub(ed[i],ed[i+1]);
    flushall();
    getch();
    closegraph();
    return 0;
}
/*========== End of Program ==========*/
```

Output

```
Midpoint Subdivision Algorithm
Enter The Edges
Right Click To Stop Entering The Edges
Enter Top Right and Bottom Left Corners of the Window
Clipped line will be :
```

➔ 5.3.1(C) Liang Barsky Line Clipping Algorithm

➔ **(May 2013, May 2014, Dec. 2014, May 2015, Dec. 2016)**

Q. Explain Liang Barsky line clipping algorithm **MU - May 2013, May 2014, Dec. 2014, Dec. 2016, 10 Marks**

Q. Explain Liang- Barsky line clipping algorithm with suitable example. **MU - May 2015, 10 Marks**

- The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen-Sutherland. The ideas for clipping line of Liang-Barsky and Cyrus-Beck are the same. The only difference is Liang-Barsky algorithm has been optimized for an upright rectangular clip window. So we will study only the idea of **Liang-Barsky.**
- Liang and Barsky have created an algorithm that uses floating-point arithmetic but finds the appropriate end points with at most four computations. This algorithm uses the parametric equations for a line and solves four inequalities to find the range of the parameter for which the line is in the viewport.

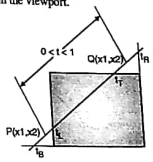


Fig. 5.3.10

- Let P(X1,Y1), Q(X2,Y2) be the line which we want to study. Refer Fig. 5.3.10. The parametric equation of the line segment from gives x-values and y-values for every point in terms of a parameter tthat ranges from 0 to 1. The equations are

$$X = X1 + (X2 - X1)*t = Y1 + dY*t$$

And

$$Y = Y1 + (Y2 - Y1)*t = X1 + dX*t$$

We can see that when t = 0, the point computed is P(x1, y1); and when t = 1, the point computed is Q(x2, y2).

☞ Algorithm

1. Set tmin=0 and tmax=1
2. Calculate the values of tL, tR, tT, and tB (tvalues).
   - o if t<tmin or t>tmax then ignore it and go to the next edge
   - o otherwise classify the tvalue as entering or exiting value (using inner product to classify)
   - o if t is entering value set tmin = t; if t is exiting value set tmax = t
3. If tmin<tmax then draw a line from (x1 + dx*tmin, y1 + dy*tmin) to (x1 + dx*tmax, y1 + dy*tmax) If the line crosses over the window, you will see (x1 + dx*tmin, y1 + dy*tmin) and (x1 + dx*tmax, y1 + dy*tmax) are intersection between line and edge.

Example 1

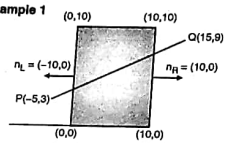


Fig. 5.3.11

Let us consider the case of Line Passing through Window in Fig. 5.3.11.

$tB = (0-3)/(9-3) = -1/2$ $t < 0$ then ignore

$tT = (10-3)/(9-3) = 7/6$ $t > 1$ then ignore

$tL = (0-(-5))/(15-(-5)) = 1/4$

$tR = (10-(-5))/(15-(-5)) = 3/4$

The next step we consider if tvalue is entering or exiting by using inner product.

$(Q-P) = (15+5, 9-3) = (20, 6)$

At left edge

$(Q-P)nL = (20, 6)(-10,0) = -200 < 0$ entering so we set tmin = 1/4

Example 2

Now let's consider another case of Line Not Passing through Window in Fig. 5.3.12.

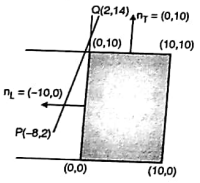


Fig. 5.3.12

$tB = (0-2)/(14-2) = -1/6$ t<0 then ignore

$tR = (10-(-8))/(2-(-8)) = 18/10$ t>1 then ignore

$tT = (10-2)/(14-2) = 8/12$

$tL = (0-(-8))/(2-(-8)) = 8/10$

The next step we consider if tvalue is entering or exiting by using inner product.

$(Q-P) = (2+8, 14-2) = (10,12)$

At top edge $(Q-P)nT = (10,12)(0,10) = 120 > 0$ exiting so we set tmax = 8/12

At left edge $(Q-P)nL = (10,12)(-10,0) = -100 < 0$ entering so we set tmin = 8/10

Because tmin > tmax then we don't draw a line.

Example 5.3.1

MU - May 2013, May 2014 , Dec. 2016, 5 Marks

Apply the algorithm to the line with co-ordinates (30, 60) and (60, 25) against the window $(X_{min}, Y_{min}) = (10, 10)$ and $(X_{max}, Y_{max}) = (50, 50)$.

Solution :

Let's first draw the window and line as shown in Fig. P. 5.3.1.

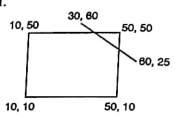


Fig. P. 5.3.1

Given things are XL = 10, YB = 10, XR = 50, YT = 50.

Let's call a line as AB with its coordinates as X1 = 30, Y1 = 60, X2 = 60, Y2 = 25

Now we have to find Dx and Dy as

$Dx = X2 - X1 = 60 - 30 = 30$

$Dy = Y2 - Y1 = 25 - 60 = -35$

Let's calculate the values of parameters P and Q as

$P1 = -Dx$ i.e. $= -30$

$P2 = Dx$ i.e. $= 30$

$P3 = -Dy$ i.e. $= -(-35) = 35$

$P4 = Dy$ i.e. $= -35$

Now $Q1 = X1 - XL = 30 - 10 = 20$

$Q2 = XR - X1 = 50 - 30 = 20$

$Q3 = Y1 - YB = 60 - 10 = 50$

$Q4 = YT - Y1 = 50 - 60 = -10$

Now let's find out values of P for i = 1 to 4

$P1 = Q1/P1 = 20/(-30) = (-2/3)$

$P2 = Q2/P2 = 20/30 = 2/3$

$P3 = Q3/P3 = 50/35 = 10/7$

$P4 = Q4/P4 = (-10)/(-35) = 2/7$

Lets' initialize t1 = 0 and t2 = 1. With this we will find t1 and t2's new values

$t1 = Max(-2/3, 10/7, 0) = 10/7$

$t2 = Min(2/3, 2/7, 1) = 2/7$

Now put these values in following expression to find our intersection point as

$X1' = X1 + Dx * t1 = 30 + 30 * (10/7) = 72.71$

$Y1' = Y1 + Dy * t1 = 60 + (-35) * (10/7) = 10$

And with t2 it will be

$X2' = X1 + Dx * t2 = 30 + 30 * (2/7) = 38.57$

$Y2' = Y1 + Dy * t2 = 60 + (-35) * (2/7) = 50$

From this we will come to know that a point (38.57, 50) is an intersection point with respect to top edge of the window boundary. So we need to discard the line from point (30,60) to (38.57,50) and consider the line (38.57,50) to (60,25). We need to repeat the same procedure for finding the intersection point with right edge of the window also and then we will get the line which is surely lying inside the window.

5.3.2 Polygon Clipping

→ **(Dec. 2014, Dec. 2015)**

Q. Explain any one polygon clipping method in detail. **MU - Dec. 2014, 10 Marks**

Q. Write short note on : Polygon clipping method. **MU - Dec. 2015, 10 Marks**

- In the last section we have seen how to clip a line. Now we will see how to clip a polygon. Basically polygon is a set of lines only. But it is a closed figure. So can we use line clipping algorithm to clip a polygon? See Fig. 5.3.13.

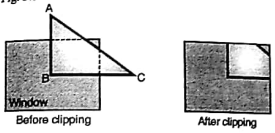


Fig. 5.3.13

- If we are using line clipping algorithm, then each line of polygon is separately considered and clipped. After clipping we will get again a set of lines. But can we call that set of lines as a polygon? Surely not, because these lines are not forming a close figure. That's why we have to modify the line clipping algorithm. So for polygon clipping, we require an algorithm which will produce a closed Figure.

See Fig. 5.3.14.

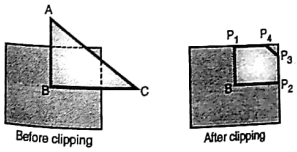


Fig. 5.3.14

- Clipping a polygon may produce a polygon with fewer vertices or one with more vertices than the original one. It may even produce several unconnected polygons.

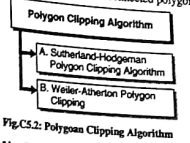


Fig.C5.2: Polygoan Clipping Algorithm

→ 5.3.2(A) Sutherland-Hodgeman Polygon Clipping Algorithm

→ **(May 2014, May 2015, May 2016)**

Q. Explain any one polygon clipping algorithm. **MU - May 2014, May 2016, 10 Marks**

Q. Explain Sutherland – Hodgeman polygon clipping algorithm in detail. **MU - May 2015, 10 Marks**

Q. Explain Sutherland-Hodgman Polygon clipping algorithm, with example. What modification is required on it so that it should also work on concave polygon. **(10 Marks)**

Q. Explain Sutherland Hodgeman polygon clipping algorithm with example. **(10 Marks)**

- We can clip a polygon by considering whole polygon against each boundary edge of the window. We know that to represent a polygon we need a set of vertices. We will pass this set of vertices or a polygon to a procedure which will clip the polygon against left edge of the window.
- This left clip procedure generates new set of vertices which indicates left clipped polygon. Again this new set of vertices is passed to the right boundary clipper procedure. Again we will get new set of vertices. Then we will pass this new set of vertices to bottom boundary clipper and lastly to top boundary clipper procedure.

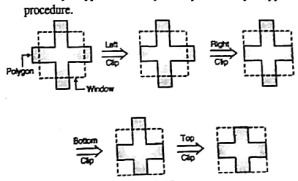


Fig. 5.3.15

- Fig. 5.3.15 shows four different stages which are required to clip a polygon. At the end of every clipping stage a new set of vertices is generated and this new set or modified polygon is passed to the next clipping stage.

- After clipping a polygon with respect to all the four boundaries we will get final clipped polygon.
- When we clip a polygon with respect to any particular edge of the window at that time we have to consider following four different cases, as each pair of adjacent polygon vertices is passed to a window boundary clipper.

Case 1 :

- If the first vertex is outside the window boundary and the second vertex is inside the window, then the intersection point of polygon with boundary edge of window and the vertex which is inside the window is stored in a output vertex list. (i.e. it will act as new vertex points). Refer Fig. 5.3.16.

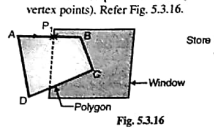


Fig. 5.3.16

- For edge AB instead of storing vertex A and B we are storing P_1 and B in output vertex list.

Case 2 :

- If both, first and second vertices of a polygon are lying inside the window, then we have to store the second vertex only in output vertex list. Refer Fig. 5.3.17.

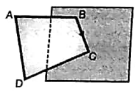


Fig. 5.3.17

Case 3 :

- If the first vertex is inside the window and second vertex is outside the window i.e. opposite to case 1, then we have to store only intersection point of that edge of polygon with window in output vertex list. Refer Fig. 5.3.18.

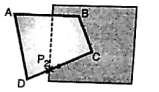


Fig. 5.3.18

Case 4 :

- If both first and second vertices of a polygon are lying outside the window then no vertex is stored in output vertex list. Refer Fig. 5.3.19.

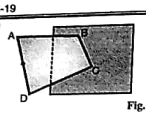


Fig. 5.3.19

- Once all vertices have been considered for one clip window boundary, the output list of vertices is clipped against the next window boundary.
- So the block diagram for this algorithm will be as shown in Fig. 5.3.20. This block diagram is a self explanatory.

Fig. 5.3.20

- Let us take an example to understand the polygon clipping algorithm more clearly. Suppose we are having polygon ABC which we want to clip against a rectangular window. See Fig. 5.3.21.

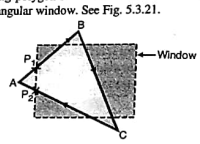


Fig. 5.3.21

- To store a polygon ABC we will use an array to hold the vertices. So array will contain three vertices A, B and C. At the time of drawing a polygon, edge is drawn between vertex A and B then edge is drawn between B and C. And finally to form a close figure we have to draw an edge between C and A i.e. last and first vertex in an array. So vertex list will become {A, B, C} and edges will be AB, BC, CA.

***Step 1* :** Clip left.

- We will consider all edges of polygon with respect to left boundary of window. For edge AB, we will store intersection point of AB edge with left boundary i.e. P_1, and inside point B. For edge BC, we will store only point C, as both B and C are inside the window (Considering only left boundary of window). For edge CA, we will store only intersection point P_2. Diagrammatic representation of left clipped polygon is shown in Fig. 5.3.22.

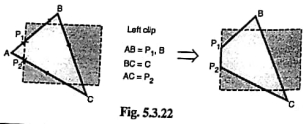


Fig. 5.3.22

After left clipping our output vertex list will become,

$\{P_1, B, C, P_2\}$

- Now we have to draw edges from P_1 to B, B to C, C to P_2 and to close figure, P_2 to P_1.

***Step 2* :** Clip Right.

- The output vertex list will not change after performing clip right procedure, because there is no edge which lies on right side of window.

***Step 3* :** Clip bottom.

- Now modified list of vertices is passed to this procedure. See Fig. 5.3.23 which is self explanatory.

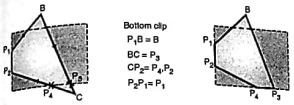


Fig. 5.3.23

- After bottom clipping set of vertices will be $\{B, P_3, P_4, P_2, P_1\}$

***Step 4* :** Clip top.

- The modified list of vertices is passed to this procedure. Fig. 5.3.24 shows details of top clipping.

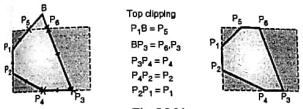


Fig. 5.3.24

So the final output vertex list will become,

$\{P_5, P_6, P_3, P_4, P_2, P_1\}$

- We have to form edges between adjacent pair of vertices and close the Figure by joining first and last vertex.

Sutherland-Hodgeman Polygon Clipping Algorithm

```
/*********************************************
Program : A C++ program to implement Sutherland -
Hodgeman polygon clipping algorithm by using mouse
*********************************************/
#include <iostream.h>
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
#include <conio.h>
typedef struct edge
{
    int xco,yco;
}EDG;
typedef struct window
{
    int xc,yc;
}WIN;
int i,indx;
EDG ed[30];
WIN wi[4];
/*============================
            Class Definition.
----------------------------*/
class clipp
{
    public:
        int button,x,y;
        int edg_count;
        EDG edpt[50],edpt1[50];
    public:
        void accept_window();
        void draw_window();
        void getmousepos(int *button,int *x,int *y);
        void initmouse();
        void showmouse();
        void init_graph();
        void accept_poly();
        void suth_hodge();
};
/*============================
Member Name :   suth_hodge()
Purpose     :   To clip a given polygon with respect to a
                window by using Sutherland-Hodgeman
                algorithm.
----------------------------*/
void clipp::suth_hodge()
{
    int i=0,j=1,k=0;
    int interx,intery;
                        //considering the left
boundary.
    for(i=1;i<=edg_count;i++)
    {
    if(ed[i].xco<wi[1].xc && ed[i+1].xco>wi[1].xc)
    {                           //outside to inside.
        interx = wi[1].xc;
intery =(((wi[1].xc-ed[i].xco)*(ed[i].yco-ed[i+1].yco))/
        (ed[i].xco-ed[i+1].xco))+ed[i].yco;
        circle(interx,intery,2);
        edpt[j].xco =interx;
        edpt[j].yco =intery;
        j++;
        edpt[j].xco =ed[i+1].xco;
```

```
            edpt[j].yco =ed[i+1].yco;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
        else if(ed[i].xco>wi[1].xc && ed[i+1].xco>wi[1].xc)
        {                                   //inside to inside.
            edpt[j].xco =ed[i+1].xco;
            edpt[j].yco =ed[i+1].yco;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
        else if(ed[i].xco>wi[1].xc && ed[i+1].xco<wi[1].xc)
        {                                   //inside to outside.
            interx = wi[1].xc;
    intery =(((wi[1].xc-ed[i].xco)*(ed[i].yco- ed[i+1].yco))/
            (ed[i].xco-ed[i+1].xco))+ed[i].yco;
            edpt[j].xco =interx;
            edpt[j].yco =intery;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
        }
            draw_window();
    setcolor(WHITE);  //displaying the left clipped polygon.
        outtextxy(150,80,"Left clipped polygon will be :");
            for(i=1;i<j-1;i++)

    line(edpt[i].xco,edpt[i].yco,edpt[i+1].xco,edpt[i+1].yco);
    line(edpt[i].xco,edpt[i].yco,edpt[1].xco,edpt[1].yco);
            edpt[j].xco = edpt[1].xco;
            edpt[j].yco = edpt[1].yco;
            getch();
                                        //considering right boundary.
        k = 1;
        for(i=1;i<j;i++)
        {
        if(edpt[i].xco<wi[2].xc && edpt[i+1].xco>wi[2].xc)
        {                                   //inside to outside
        interx = wi[2].xc;
    intery =(((wi[2].xc-edpt[i].xco)*(edpt[i].yco-edpt[i+1].yco))/
            (edpt[i].xco-edpt[i+1].xco)+ edpt[i].yco;
            edpt1[k].xco = interx;
            edpt1[k].yco = intery;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        if(edpt[i].xco>wi[2].xc && edpt[i+1].xco <wi[2].xc)
        {                                   //outside to inside
        interx = wi[2].xc;
        intery =(((wi[2].xc-edpt[i].xco)*(edpt[i].yco-
    edpt[i+1].yco))/
            (edpt[i].yco-edpt[i+1].yco))+ edpt[i].yco;
            edpt[j].xco = interx;
            edpt[j].yco = intery;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
```

```
            (edpt[i].xco-edpt[i+1].xco))+ edpt[i].yco;
            edpt1[k].xco = interx;
            edpt1[k].yco = intery;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
            edpt1[k].xco = edpt[i+1].xco;
            edpt1[k].yco = edpt[i+1].yco;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        if(edpt[i].xco<wi[2].xc && edpt[i+1].xco<wi[2].xc)
        {                                   //inside to inside
            edpt1[k].xco = edpt[i+1].xco;
            edpt1[k].yco = edpt[i+1].yco;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        }
            getch();
    cleardevice();  //displaying the right clipped polygon.
    outtextxy(150,80,"Right clipped polygon will be :");
            draw_window();
            setcolor(WHITE);
            for(i=1;i<k-1;i++)
    line(edpt1[i].xco,edpt1[i].yco,edpt1[i+1].xco,edpt1[i+1].yco)
    ;
    line(edpt1[i].xco,edpt1[i].yco,edpt1[1].xco,edpt1[1].yco);
            edpt1[k].xco = edpt1[1].xco;
            edpt1[k].yco = edpt1[1].yco;
            getch();    //considering Bottom boundary.
        j = 1;
        for(i=1;i<k;i++)
        {
        if(edpt1[i].yco<wi[1].yc&& edpt1[i+1].yco<wi[1].yc)
        {                                   //inside to inside.
            edpt[j].xco = edpt1[i+1].xco;
            edpt[j].yco = edpt1[i+1].yco;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
        if(edpt1[i].yco<wi[1].yc&& edpt1[i+1].yco>wi[1].yc)
        {           //inside to outside.
        intery = wi[1].yc;
        interx =(((wi[1].yc-edpt1[i].yco)*(edpt1[i].xco-
    edpt1[i+1].xco)) /
            (edpt1[i].yco-edpt1[i+1].yco))+ edpt1[i].xco;
            edpt[j].xco = interx;
            edpt[j].yco = intery;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
        }
```



```
        if(edpt1[i].yco>wi[1].yc&& edpt1[i+1].yco<wi[1].yc)
                                    //outside to inside.
        {
        intery = wi[1].yc;
        interx =(((wi[1].yc-edpt1[i].yco)*(edpt1[i].xco-
    edpt1[i+1].xco)) /
            (edpt1[i].yco-edpt1[i+1].yco))+edpt1[i].xco;
            edpt[j].xco = interx;
            edpt[j].yco = intery;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;
            edpt[j].xco = edpt1[i+1].xco;
            edpt[j].yco = edpt1[i+1].yco;
            circle(edpt[j].xco,edpt[j].yco,2);
            j++;

        }
        }
            getch();
    cleardevice();  //displaying the top clipped polygon.
    outtextxy(150,80,"Bottom clipped polygon will be :");
            draw_window();
            setcolor(WHITE);
            for(i=1;i<j-1;i++)
    line(edpt[i].xco,edpt[i].yco,edpt[i+1].xco,edpt[i+1].yco);

    line(edpt[i].xco,edpt[i].yco,edpt[1].xco,edpt[1].yco);
            edpt[j].xco = edpt[1].xco;
            edpt[j].yco = edpt[1].yco;
            getch();   //considering Top boundary.
        k = 1;
        for(i=1;i<j;i++)
        {
        if(edpt[i].yco>wi[3].yc&& edpt[i+1].yco>wi[3].yc)
        {                                   //inside to inside.
            edpt1[k].xco = edpt[i+1].xco;
            edpt1[k].yco = edpt[i+1].yco;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        if(edpt[i].yco>wi[3].yc&& edpt[i+1].yco<wi[3].yc)
        {                                   //inside to outside.
        intery = wi[3].yc;
    interx =(((wi[3].yc-edpt[i].yco)*(edpt[i].xco-edpt[i+1].xco)) /
            (edpt[i].yco-edpt[i+1].yco))+edpt[i].xco;
            edpt1[k].xco = interx;
            edpt1[k].yco = intery;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        if(edpt[i].yco<wi[3].yc&& edpt[i+1].yco>wi[3].yc)
        {                                   //outside to inside.
        intery = wi[3].yc;
```

```
    interx =(((wi[3].yc-edpt[i].yco)*(edpt[i].xco-edpt[i+1].xco)) /
            (edpt[i].yco-edpt[i+1].yco))+edpt[i].xco;
            edpt1[k].xco = interx;
            edpt1[k].yco = intery;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
            edpt1[k].xco = edpt[i+1].xco;
            edpt1[k].yco = edpt[i+1].yco;
            circle(edpt1[k].xco,edpt1[k].yco,2);
            k++;
        }
        }
    getch(); //displaying the bottom clipped and final polygon.
        cleardevice();
        outtextxy(150,80,"Top clipped polygon will be :");
        draw_window();
            setcolor(WHITE);
            for(i=1;i<k-1;i++)
    line(edpt1[i].xco,edpt1[i].yco,edpt1[i+1].xco,edpt1[i+1].yco)
    ;
    line(edpt1[i].xco,edpt1[i].yco,edpt1[1].xco,edpt1[1].yco);
        getch();
    }
    /*============================
    Member Name :     accept_poly()
    Purpose     :     To accept the polygon from user by
                      making use of mouse.
    ----------------------------------------------------------*/
    void clipp::accept_poly()
    {
        int i,j;
        outtextxy(200,50,"Enter The Edges");
    outtextxy(100,70,"Right Click To Stop Entering The Edges");
        x=y=button=i=edg_count=0;
        while(button!=2)
        {
        getmousepos(&button,&x,&y);
        if(button==1)
        {
            delay(250);
            i++;
            ed[i].xco = x;
            ed[i].yco = y;
            if(i>=2)
            {
        line(ed[i].xco,ed[i].yco,ed[i-1].xco,ed[i-1].yco);
        edg_count++;
            }
        }
    }
```

```
/*============================
Member Name :   init_graph()
Purpose     :   To initialise the graphics system.
----------------------------------------------------------------*/
void clipp::init_graph()
{
      int gd=DETECT,gm;
      initgraph(&gd,&gm,"c:\\tcplus\\bgi");
}
/*============================
Member Name :   accept_window().
Purpose     :   To accept the window from user by
                making use of mouse.
----------------------------------------------------------------*/
void clipp::accept_window()
{
      outtextxy(70,90,"Enter Bottom left and Top Right
Corners of the Window");
      button = x = y = 0;
      i = 1;
      while(1)
      {
      getmousepos(&button,&x,&y);
      if(button==1)
      {
          delay(250);
          wi[i].xc = x;
          wi[i].yc = y;
          i+=2;
          if(i==5)
          break;
      }
      }
          wi[2].xc = wi[3].xc;
          wi[2].yc = wi[1].yc;
          wi[4].xc = wi[1].xc;
          wi[4].yc = wi[3].yc;
}
/*============================
Member Name :   draw_window()
Purpose     :   To display the window.
----------------------------------------------------------------*/
void clipp::draw_window()
{
          setcolor(RED);
    line(wi[1].xc,wi[1].yc,wi[2].xc,wi[2].yc);
          line(wi[2].xc,wi[2].yc,wi[3].xc,wi[3].yc);
          line(wi[3].xc,wi[3].yc,wi[4].xc,wi[4].yc);
          line(wi[4].xc,wi[4].yc,wi[1].xc,wi[1].yc);
          getch();
}
```

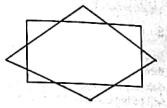
```
/*============================
Member Name :   getmousepos()
Purpose     :   To get the pixel clicked by the mouse.
----------------------------------------------------------------*/
void clipp::getmousepos(int *button,int *x,int *y)
{
          int b1=0,xt,yt;
          asm {
              mov ax,3
              int 0x33
              mov b1,bx
              mov xt,cx
              mov yt,dx
              }
              *button = b1;
              *x = xt;
              *y = yt;
}
/*============================
Member Name :   initmouse()
Purpose     :   To initialize the mouse.
----------------------------------------------------------------*/
void clipp::initmouse()
{         asm {
              mov ax,0
              int 0x33
              }
}
/*============================
Member Name :   showmouse()
Purpose     :   To show the mouse.
----------------------------------------------------------------*/
void clipp::showmouse()
{    asm{
     mov ax,1
     int 0x33
     }  }
int main()
{
      int choice,count;
      clipp c;
      clrscr();
      c.init_graph();
      c.initmouse();
      c.showmouse();
      c.edg_count = 0;
      outtextxy(100,10,"Sutherland Hodgeman Polygon
Clipping Algorithm");
      c.accept_poly();
      c.accept_window();
      c.draw_window();
```



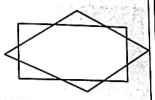
```
      cleardevice();
      outtextxy(150,70,"clipped polygon will be");
      c.suth_hodge();
      closegraph();
      return 0;
}
/*========== End of Program ==========*/
```

Output

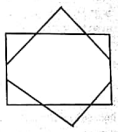
Sutherland Hodgeman Polygon Clipping Algorithm
Enter The Edges
Right Click To Stop Entering The Edges
Enter Bottom left and Top Right Corners of the Window



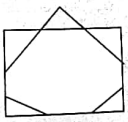
Left clipped polygon will be :



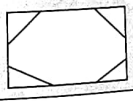
Right clipped polygon will be :



Bottom clipped polygon will be :



Top clipped polygon will be :



- All convex polygons get correctly clipped by the Sutherland-Hodgeman algorithm, as shown in Fig. 5.3.25, but some concave polygons may be displayed with extra edges.

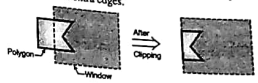
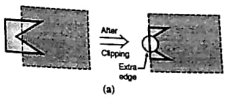


Fig. 5.3.25

- If the concave polygon is as shown in Fig. 5.3.25 then it will get clipped properly as shown in same Figure. But if the same concave polygon is as shown in Fig. 5.3.26(a) then extra edges appears in final clipped Figure.



(a)

(b)

Fig. 5.3.26

- The problem of extra edge occurs since we are storing all the vertices in a single vertex list and to form a closed figure, we are forming an edge between last and first vertex. The actual clipped output should be as shown in Fig. 5.3.26(b).
- To overcome this problem, one way is to split the concave polygon. It means we will divide a single concave polygon in two or more convex polygons and process each convex polygon separately. But dividing a concave polygon is not a simple method. So we will go for more general polygon clipping algorithm such as Weiler-Atherton polygon clipping algorithm.

→ 5.3.2(B) Weiler-Atherton Polygon Clipping

→ (May 2013, Dec. 2016, May 2017)

Q. Explain Weiler-Atherton algorithm for polygon clipping. What are its advantages over other polygon clipping algorithms ? **MU - May 2013, May 2017, 10 Marks**

Q. Explain Weiler-Atherton algorithm for polygon clipping. **MU - Dec. 2016, 10 Marks**

Q. Explain how Weiler-Atherton algorithm works for convex polygons? **(10 Marks)**

Q. How Weiler-Athorton algorithm solves the problem of concave polygon clipping ? **(6 Marks)**

- This algorithm is one of the generalized polygon clipping algorithm. In Sutherland-Hodgeman algorithm, we always follow the path of polygon. But in this case sometime we are selecting polygon path and sometime window path. When to follow which path, that will depend on polygon processing direction, and whether a pair of polygon vertices currently being processed represents an outside to inside pair or an inside to outside pair.
- We can follow polygon processing path either clockwise or anticlockwise. When we are following a path of polygon in clockwise direction at that time we have to use following rules.
- We have to follow the polygon boundary, same as that of Sutherland-Hodgeman algorithm, if the vertex pair is outside to inside.
- We have to follow the window boundary in clockwise direction, if the vertex pair is inside to outside.
- Consider the concave polygon as shown in Fig. 5.3.27.
- Let us process the polygon in clockwise path. Here our vertex list will be the set of all vertices i.e. {A, B, C, D}.

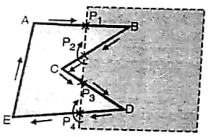


Fig. 5.3.27

- For edge AB, as we are moving from outside to inside so store intersection point P_1 and second point i.e. B. As we are following polygon in clockwise direction, so we have to consider next edge as BC. Here we are moving from inside to outside. So we are storing only intersection point P_2. Upto this point it is similar to normal polygon clipping algorithm. But, as we are moving from inside to outside so we have to follow window boundary in clockwise direction. And the next point on this path will be P_1. So we have to store that. It means we are storing P_2 and P_1. For next edge i.e. CD, again we are storing intersection point P_3 and next point D. For edge DE we are storing intersection point P_4 and following window boundary in clockwise direction i.e. storing P_3. For edge EA, as both points are outside, no vertex is stored.See Fig. 5.3.28.

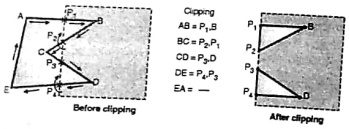


Fig. 5.3.28

- Here in Weiler-Atherton algorithm we are maintaining as such two different vertex list in single list. Here after clipping, the vertex list will be {P_1, B, P_2, P_1, P_3, D, P_4, P_3}. We have to draw edges by joining vertices as P_1 to B, B to P_2, P_2 to P_1. Now there is no need to form an edge between P_1 to P_3. We have to continue with edge P_3 to D, D to P_4 and P_4 to P_3. Again here there is no need to close the figure by drawing edge between P_3 to P_1 because already the figure is closed. Same thing is for edge P_1 to P_3. Here how we will come to know when to not form edge. We may use some logic here such as, when any vertex is stored twice in vertex list then don't form edge from that vertex to next vertex. Now we will not have edge P_1 P_3 and edge P_3 P_1. That's why we are saying, we are maintaining two sub array in single array of vertex.

5.3.3 Curve Clipping

- In curve clipping, consider the object as a circle, then there will be different situations. We will see one by one. Here we are going to form a boundary box which will be big enough to hold the circle.

Case I : If the circle is as shown in Fig. 5.3.29.

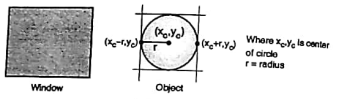


Fig. 5.3.29

- Now we will check whether the bounding box of circle is getting overlapped with window or not. If there is no overlap, it means the circle is completely lieing outside the window. So discard the whole circle.

Case II : If circle is as shown in Fig. 5.3.30.

- Case II indicates if the center of circle is one of the corner point of the window. Then we have to draw only one quadrant of that circle which will be inside the window. Here we know center of circle (x_c, y_c) and radius r from that we can find out (x_c– r, y_c) point and (x_c, y_c – r) point. Then we have to draw the pixels on curve by normal circle generation algorithm. Here we should not use property of symmetry.

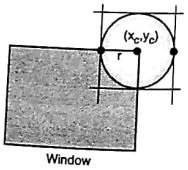


Fig. 5.3.30

Case III : If the circle is as shown in Fig. 5.3.31.

- If the center of the circle is outside the window and still circle overlaps the window then we will check four points i.e. (x_c – r, y_c), (x_c + r , y_c), (x_c, y_c – r), (x_c, y_c + r). Which point is inside the window by comparing window co-ordinates with these four co-ordinates. From this we will come to know that (x_c – r, y_c) is inside the window, because its x co-ordinate is in between windows x_{left} and x_{right} co-ordinates.
- Then we will display that point and find the next point along the path of circle. Again for every point we have to check whether the calculated point is inside the window or not, till the intersection point of the circle with window is reached.

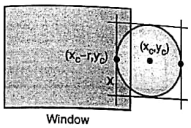


Fig. 5.3.31

- The same procedure should be applied if the center of the circle is inside the window and some part of circle is outside the window.

Case IV :

- If the circle is as shown in Fig. 5.3.32;For this case where circles center point is inside and its some part is outside the window, we are applying same method which is discussed in case III.

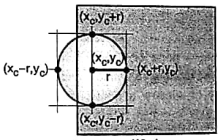


Fig. 5.3.32

Case V :

- The circle is fully visible when it is as shown in Fig. 5.3.33.This is the simplest case. Here we have to compare the bounding box of circle with window co-ordinates. If the box is completely inside the window then display the circle fully.

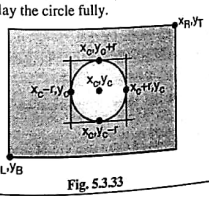


Fig. 5.3.33

5.3.4 Text Clipping

Q. Write short note Text clipping methods. (5 Marks)
Q. Explain Text clipping with help of appropriate examples. (10 Marks)

- We cannot use line clipping or polygon clipping methods to clip messages or text. There are several techniques that can be used to provide text clipping. The clipping technique used will depend on the methods used to generate the characters.

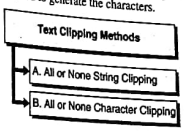


Fig.C5.3: Text Clipping Methods

→ 5.3.4(A) All or None String Clipping

- This method of text clipping is the simplest method. Refer Fig. 5.3.34.

Fig. 5.3.34

- In this case a rectangular box is created around the string. The boundary position of this rectangle is then compared to the window boundaries, and the string is rejected if these two boundaries get overlapped. Here, the main drawback is, if a single character of a text is outside then we are discarding the whole text. If the rectangular box of the text is completely inside the window boundary i.e. all the characters of text are inside the window, then only that text will get displayed.
- This method is fast but it is not efficient. See Fig. 5.3.34. In this figure two strings are there, "ABCD" and "PQR". String "PQR" is completely inside the window, so it will be displayed. Whereas in case of string "ABCD", only one character is outside and rest three characters are inside, still we are rejecting the whole string.

→ 5.3.4(B) All or None Character Clipping

- This technique provides alternative way for text clipping. In this case instead of rejecting the whole string, we are discarding only those characters that are not completely inside the window. See Fig. 5.3.35.

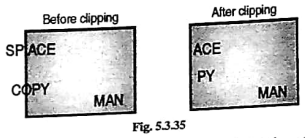


Fig. 5.3.35

- In this case the boundary limits of individual characters are compared to the window. The character which either overlaps or lying completely outside the window get's clipped. The string "MAN" is completely inside the window, so it will get displayed properly. But for string "SPACE", first two characters are outside the window and next three characters are inside the window. In "all or none string clipping" technique we are discarding whole string. But here we are discarding only those characters which are lying outside the window so only characters 'S' and 'P' will get discarded and rest three characters which are inside the window will get displayed. Similarly for string "COPY" character 'C' is completely outside and characters 'P' and 'Y' are completely inside. But a character 'O' is partially outside and partially inside. So we are clipping both 'C' and 'O' characters and displaying only 'P' and 'Y' characters. This method is more efficient than the earlier method.
- There is a third way by which we can clip the text. Here the method deals with the component of the individual character. Here we treat characters as a set of lines or pixels. See Fig. 5.3.36.

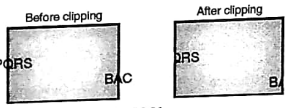


Fig. 5.3.36

- If the individual character overlaps the window boundary, then we clip of the part of character which is outside the window. In above example for string "PQRS", character 'Q' is partially inside and partially outside the window. So after clipping, the part of character which is outside gets clipped and the inside part will get displayed.
- This is the most accurate method for clipping text. But it is time consuming also, because here we have to check each and every pixel of character to decide whether it is inside or outside the window.

5.3.5 Interior and Exterior Clipping

- Until now whatever clipping techniques we have seen is interior clipping. It means we are displaying the objects or portion of objects which is inside the window and discarding the portion which is outside. But in exterior clipping we are doing exactly reverse of the interior clipping. See Fig. 5.3.37.

Fig. 5.3.37

- In case of exterior clipping we are discarding the object or part of the object which is inside the window and displaying the part which is outside the window. This exterior clipping is mainly used in multiple window concepts.

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 Derive window-viewport or windowing or viewing transformation matrix.

Ans. :

- This question is very important from theory examinations point of view. Here we have to first explain the meaning of window and viewport. Then the necessity of window to viewport transformation and then we have to explain following points.
- Translation of lower left corner of window to origin. Form a translation matrix for this
- Then scale the window to viewport. Use scaling factor s_x and s_y.

$$S_x = \frac{\text{width of viewport}}{\text{width of window}}$$

$$S_y = \frac{\text{height of viewport}}{\text{height of window}}$$

- Form a scaling matrix for this.
- Then again translate the viewport to desired location on screen by using translation matrix.
- Combine all the three transformation matrices to a single matrix.
- For this question we can make use of diagrams.

Q. 2 Explain Cohen-Sutherland algorithm for line clipping.

Ans. : The solution for this question is very straight forward. Here we have to explain following points :

- End points outcode.
- Test to decide the line is totally visible or totally invisible.
- Logical ANDing process for partially visible lines.
- Take a suitable example by drawing window and lines, before and after clipping to explain above mentioned points. Consider all types of lines, such as totally visible, totally invisible and partially visible line.

Q. 3 Explain Sutherland-Hodgeman Polygon clipping algorithm.

Ans. : The solution for this question should include,

- Block diagram for polygon clipping.
- Explain different stages of clipping by taking an example and drawing diagrams.
- Explain all the four cases of vertex pairs of a polygon i.e. :

I. When we move from outside to inside vertex, store intersection point and inside point.

II. When we move from inside to inside, store only second point.

III. When we move from inside to outside, store only intersection point.

IV. When we move from outside to outside no point is stored.

- Take an example and explain it by clipping with respect to all the four sides of window.

Q. 4 Explain Generalized polygon clipping algorithm.

Ans. : Here we have to explain the limitation of Sutherland-Hodgeman polygon clipping algorithm using diagrams. Then we have to explain Weiler-Atherton algorithm by considering an example of a concave polygon. Draw the diagrams wherever necessary.

Important Assignments

This chapter is important from practical examination point of view. We have already explained and given, enough number of programs at the time of explaining algorithms. In addition to these programs, in final practical examination, few modifications are also expected. The list of such programs is given below :

Q. 1 Write a program to clip lines by using Cohen-Sutherland /midpoint subdivision algorithm.

Q. 2 Write a program to clip any polygon by using Weiler-Atherton algorithm.

Q. 3 Write a program to clip a polygon against sliding window. Here polygon should be stationary and as we press left, right, up and down arrow keys, our window co-ordinates should change.

Q. 4 Write a program to clip lines against stationary rectangular window. As we press left, Right arrow keys the line co-ordinates should change.

Q. 5 Write a program to clip a circle/ellipse against a rectangular window.

5.4 Important Questions ans Answers

Q. 1 Clip the line PQ having coordinates A(4,1) and B(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. Mention the limitations of algorithm. How it can be overcome?

Ans. :

From the given information let's first draw a line PQ and window ABCD.

From Fig. 1 it is clear that point P is lying outside the window and point Q is lying inside the window. According to Cohen- Sutherland algorithm we need to find out codes of the end point of a line PQ.

$\therefore$ outcode of P = 0100

and outcode of Q = 0000

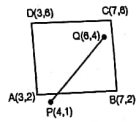


Fig. 1

Since both out codes are not zero we need to logically AND them.

$$\begin{array}{r} 0100 \\ \therefore \quad 0000 \\ \hline 0000 \end{array}$$

Now the logical AND result is zero. From this we can say that a line PQ may be visible or may be partially invisible.

So, according to Cohen, Sutherland algorithm we will check the out code of end point of P bit by bit.

0100

↑

Bit 3 from right side is set to 1. It means the point P is below the window. From this information we have to find intersection point of a line PQ with lower boundary of the window.

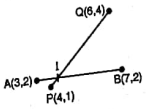


Fig. 1(a)

We have to find intersection point I of lines PQ and AB. Here the Y value of intersection point I is known i.e. 2 and we have to find X value.

For that first let's find the slope of line PQ.

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4-1}{6-4} = \frac{3}{2} = 1.5$$

$$\therefore m = 1.5$$

Now by using, $X = X_1 + \frac{y_{window} - y_1}{m} = 4 + \frac{2-1}{1.5}$

$$= \frac{6+1}{1.5} = \frac{7}{1.5}$$

$$\therefore X = \frac{7}{1.5} = 4.66$$

So the co-ordinates of intersection point I will be (4.66, 2).

By finding out the intersection point we have divided a line PQ into PI and IQ.

Since point P of PI sub-line is outside the window i.e. below the window we have to discard this line PI.

Now find out outcodes for point I which is on bottom boundary of window. Outcode of I is 0000 and outcode of Q is 0000. Since both outcode's of line IQ are 0000 the line IQ is completely visible inside the window.

Q. 2 Let R be the rectangular window whose lower left hand corner is at L(– 3,1) and upper right corner is at R(2,6). Find the region codes for the endpoints and use Cohen-Sutherland algorithm to clip the line segments.

Coordinates for line segments are

For line AB A (– 4,2) and B(– 1,7)

CD C (– 1,5) and D(3,8)

EF E (– 2,3) and F(1,2).

Ans. :

Lets first draw the window and linear segments from the given data.

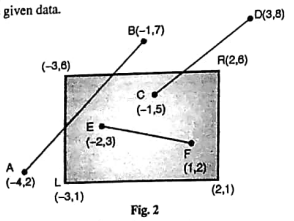


Fig. 2

From the above Fig. 8, lets first derive the region codes for linear AB, CD and EF.

Region code for A = ABRL = 0001

B = ABRL = 1000

Region code for C = ABRL = 0000

D = 1010

Region code for E = ABRL = 0000

F = 0000

From the above region code it is clear that line EF is fully visible since it's both end points region codes are 0000.

Now let's see line AB whose one ends region code is 0001 and other ends region code is 1010. Since both region codes are not 0000, this line AB is not fully visible.

∴ We need to take logical AND of both region codes of A and B.

A = 0001

AND

B = 1000

= 0000

Since logical AND of region codes of line AB are 0000 we need to see each region code in detail. Since the L bit of ABRL with respect to point A is '1', it means point 'A' is on left side of the window. It means we have to find intersection point of a line AB with left boundary of window.

Let's call this intersection point as I_1. The x-coordinate of point I_1 will be same as left boundary of window i.e. $x = -3$. Now we have to find y-coordinate of I_1 point. For this we need to find slope of line AB which will be

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-2}{(-1)-(-4)} = \frac{5}{3} = 1.6$$

∴ Slope of line AB = 1.6

We know that the slope between two end points of a line is same as that of slope between one end point and any other point which is lying on the line.

∴ Slope between point A and intersection point I_1 will be

$m = \frac{y_2 - y_1}{x_2 - x_1}$ Where x, y are related to 'A' x_2 is known i.e. – 3 y_2 we have to find

$$\therefore m(x_2 - x_1) = (y_2 - y_1)$$

$$\therefore y_2 = m(x_2 - x_1) + y_1$$

$$= 1.6((-3) - (-4)) + 2$$

$$= 1.6(1) + 2 = 3.6$$

$$\therefore y_2 = 3.6$$

∴ The intersection point is (– 3, 3.6).

Since this intersection point I_1 is at left boundary its region code will be (0,0,0,0). If we consider line segment AI_1 then A point is surely left side of the window and I_1 is on left boundary of window. So we have to discard AI_1.

Similarly we have to find the another intersection point I_2 with respect to top boundary of the window. Now I_2 points region code will become 0000. So we will discard I_2B since B point is above the window.

Now our line segment is I_1 I_2. Both I_1 and I_2's region codes are 0000. So the line I_1 I_2 is completely visible and we have to display I_1I_2 line.

Similarly for line CD, one point C is inside the window, whereas point D is outside the window.

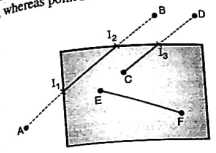


Fig. 2(a)

So here also we have to find region codes of C and D which will be

C = 0000 and D = 1010

Here we need to find intersection of line CD with respect o top boundary. Let's call that point as I_3 and then discard the line I_3D. Since D's region code is nonzero, which means point D is surely lying outside the window.

As intersection point I_3 region code is 0000 and another end-point 'C's region code is also 0000, this line segment CI_3 is fully visible.

Q. 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 Cohen Sutherland algorithm to clip the lines

p_1 p_2 with p_1 (10, 30), p_2 (80, 90) and

q_1q_2 with q_1 (10, 10), q_2 (70, 60).

Ans. : Let's first draw the window and lines p_1p_2 and q_1q_2 from the given data,

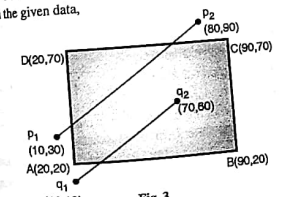


Fig. 3

From the above Fig. 3 lets 1st derive the region codes of p_1p_2 and q_1q_2.

Region code of p_1 = ABRL = 0001

Region code of p_2 = ABRL = 1000

Similarly region code of q_1 = ABRL = 0101

Region code of q_2 = ABRL = 0000

Now let's use Cohen-Sutherland algorithm to clip these lines. First consider line p_1p_2.

Since both region codes are not 0000 we have to take logical AND of these region codes.

p_1 = 0001

p_2 = 1000

AND = 0000

Since logical AND result is 0000 line p_1p_2 may be partially visible.

Let's consider point p_1 whose region code is 0001. Here last bit which signifies left boundary of window is set to 1. It means point p_1 is on left side of the window boundary i.e. point p_1 is surely lying outside the window.

So we will find the intersection of line p_1p_2 with left edge of window.

Since left edge of window is having x coordinate as 20. The intersection point x-coordinate will also be 20 only. Now we have to find y co-ordinate of intersection point. For this we will use slope intercept form of line.

i.e. let's find slope of line p_1p_2.

$$\therefore m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{90-30}{80-10} = \frac{60}{70} = 0.8$$

So, slope of line p_1p_2 is 0.8.

We know that the slope between two endpoints of a line is same as that of slope between one endpoint and any other point which is lying on the line.

∴ Slope between point p_1 and intersection point will be

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

where x_1, y_1 are related with p_1, x_2 is known i.e. 20, y_2 we have to find.

$$\therefore m(x_2 - x_1) = y_2 - y_1$$

$$\therefore y_2 = m(x_2 - x_1) + y_1$$

$$\therefore y_2 = 0.8(20 - 10) + 30$$

$$= 0.8(10) + 30 = 8 + 30$$

$$y_2 = 38$$

Therefore the intersection point is (20, 38). Let's call this point as I_1. Since I_1 point is exactly at the boundary. Its region code will be (0, 0, 0, 0). Now we have to discard the line segment p_1 I_1 Since p_1 point is outside window.

Similarly we have to find another intersection point I_2 with respect to top boundary. Now I_2 points region.

Code will be 0000. So we will discard I_2p_2 since p_2 point is above the window.

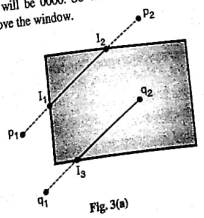


Fig. 3(a)

Now our line segment is $I_1 I_2$. Both I_1 and I_2's region codes are 0000. So the line I_1I_2 is completely visible and we have to display I_1I_2 line.

Similarly for line q_1q_2, one point q_2 is inside the window, whereas point q_1 is outside the window. So here also we have to find the intersection point with respect to the window boundary. Let's call it as point I_3 and then discard the line q_1I_3 since q_1's region code is nonzero, which means point q_1 is surely laying outside the window. As intersection point I_3's region code is 0000 and another endpoint q_2's region code is also 0000 this line segment $I_3\, q_2$ is visible.

5.5 Exam Pack (University and Review Questions)

Q. Define window. *(Refer Section 5.1)* **(May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 2 Marks)**

Q. What is Window? *(Refer Section 5.1)* **(4 Marks)**

☞ **Syllabus Topic : Viewing Transformation Pipeline and Window to Viewport Co-Ordinate Transformation**

Q. Differentiate between Object space and Image Space. *(Refer Section 5.2)* **(May 2013, 10 Marks)**

Q. Define view port and derive window to view port transformation. *(Refer Section 5.2)* **(May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 10 Marks)**

Q. What is viewing transformation? *(Refer Section 5.2)* **(May 2015, 5 Marks)**

Q. Write short note on : Viewing transformation. *(Refer Section 5.2)* **(May 2016, 10 Marks)**

Q. What is viewport? Assume Window and viewport are rectangular. Derive the formulas required for transforming a point (x_w, y_w) in a window to point (x_v, y_v) in viewport. *(Refer Section 5.2)* **(5 Marks)**

Q. Give the steps along with the transformation matrix for the mapping of the 2-D object from window to viewport. *(Refer Section 5.2)* **(10 Marks)**

☞ **Syllabus Topic : Clipping operations - Point clipping , Line clipping algorithms : Cohen - Sutherland, Midpoint subdivision, Llang - Barsky, Polygon Clipping Algorithms : Sutherland – Hodgeman, Weiler - Atherton.**

Q. State what is meant by clipping. *(Refer Section 5.3)* **(Dec. 2015, 2 Marks)**

Q. State what is meant by clipping. Explain any one clipping algorithm. *(Refer Section 5.3.1(A))* **(Dec. 2015, 3 Marks)**

Q. Explain Cohen-Sutherland line clipping algorithm. *(Refer Section 5.3.1(A))* **(May 2016, May 2017, 10 Marks)**

Q. Give the line clipping algorithm which uses area codes for the end points of the line segment. *(Refer Section 5.3.1(A))* **(10 Marks)**

Q. Explain Midpoint Subdivision Algorithm. *(Refer Section 5.3.1(B))* **(Dec. 2010, 10 Marks)**

Q. Explain Liang Barsky line clipping algorithm *(Refer Section 5.3.1(C))* **(May 2013, May 2014, Dec. 2014, Dec. 2016, 10 Marks)**

Q. Explain Liang- Barsky line clipping algorithm with suitable example. *(Refer Section 5.3.1(C))* **(May 2015, 10 Marks)**

Q. Explain any one polygon clipping method in detail. *(Refer Section 5.3.2)* **(Dec. 2014, 10 Marks)**

Q. Write short note on : Polygon clipping method. *(Refer Section 5.3.2)* **(Dec. 2015, 10 Marks)**

Q. Explain any one polygon clipping algorithm. *(Refer Section 5.3.2(A))* **(May 2014,May 2016, 10 Marks)**

Q. Explain Sutherland – Hodgeman polygon clipping algorithm in detail. *(Refer Section 5.3.2(A))* **(May 2015, 10 Marks)**

Q. Explain Sutherland-Hodgman Polygon clipping algorithm, with example. What modification is required on it so that it should also work on concave polygon. *(Refer Section 5.3.2(A))* **(10 Marks)**

Q. Explain Sutherland Hodgeman polygon clipping algorithm with example. *(Refer Section 5.3.2(A))* **(10 Marks)**

Q. Explain Weiler-Atherton algorithm for polygon clipping. What are its advantages over other polygon clipping algorithms ? *(Refer Section 5.3.2(B))* **(May 2013, May 2017, 10 Marks)**

Q. Explain Weiler-Atherton algorithm for polygon clipping. *(Refer Section 5.3.2(B))* **(Dec. 2016, 10 Marks)**

Q. Explain how Weiler-Atherton algorithm works for convex polygons? *(Refer Section 5.3.2(B))* **(10 Marks)**

Q. How Welier-Athorton algorithm solves the problem of concave polygon clipping ? *(Refer Section 5.3.2(B))* **(6 Marks)**

Q. Write short note Text clipping methods. *(Refer Section 5.3.4)* **(5 Marks)**

Q. Explain Text clipping with help of appropriate examples. *(Refer Section 5.3.4)* **(10 Marks)**

□□□

CHAPTER 6

Module 5

3D Object Representation and Fractal Geometry

Section Nos.	Name of the Topic
6.1	Introduction
6.2	3D Object Representation Methods
6.3	Curve Generation
6.4	Bezier Curve
6.5	Fractals

Chapter 6 introduces three-dimensional object representation and fractal geometry. Generally we are using straight lines to draw images on computer screen. But we can draw images by using several sample points, which follows the path of curve. This chapter also describes curves and fractals. This chapter has five sections.

6.1 **Introduction** introduces the need of curves, fractals and 3D object representation.

6.2 **3D Object Representation** method explains different methods such as B-REP, CSG, Sweep and their comparison.

6.3 **Curve Generation** explains different methods to generate any curve. It also explains the concept of B-splines.

6.4 **Bezier curves** are another type of approximating curves, useful in computer graphics. It also explains generation of Bezier surfaces.

6.5 **Fractals** explain what a fractal is and also explains different fractals such as Hilbert's curve and Triadic Koch curve. This section also explains how to generate fractal lines and surfaces.

Now our line segment is I_1 I_2. Both I_1 and I_2's region codes are 0000. So the line I_1I_2 is completely visible and we have to display I_1I_2 line.

Similarly for line q_1q_2, one point q_2 is inside the window, whereas point q_1 is outside the window. So here also we have to find the intersection point with respect to the window boundary. Let's call it as point I_3 and then discard the line q_1I_3 since q_1's region code is nonzero, which means point q_1 is surely laying outside the window. As intersection point I_3's region code is 0000 and another endpoint q_2's region code is also 0000 this line segment I_3 q_2 is visible.

5.5 Exam Pack (University and Review Questions)

Q. Define window. *(Refer Section 5.1)* **(May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 2 Marks)**

Q. What is Window? *(Refer Section 5.1)* **(4 Marks)**

Syllabus Topic : Viewing Transformation Pipeline and Window to Viewport Co-Ordinate Transformation

Q. Differentiate between Object space and Image Space. *(Refer Section 5.2)* **(May 2013, 10 Marks)**

Q. Define view port and derive window to view port transformation. *(Refer Section 5.2)* **(May 2014, Dec. 2014, Dec. 2015, May 2016, May 2017, 10 Marks)**

Q. What is viewing transformation? *(Refer Section 5.2)* **(May 2015, 5 Marks)**

Q. Write short note on : Viewing transformation. *(Refer Section 5.2)* **(May 2016, 10 Marks)**

Q. What is viewport? Assume Window and viewport are rectangular. Derive the formulas required for transforming a point (x_w, y_w) in a window to point (x_v, y_v) in viewport. *(Refer Section 5.2)* **(5 Marks)**

Q. Give the steps along with the transformation matrix for the mapping of the 2-D object from window to viewport. *(Refer Section 5.2)* **(10 Marks)**

Syllabus Topic : Clipping operations - Point clipping , Line clipping algorithms : Cohen - Sutherland, Midpoint subdivision, Liang - Barsky, Polygon Clipping Algorithms : Sutherland – Hodgeman, Weiler - Atherton.

Q. State what is meant by clipping. *(Refer Section 5.3)* **(Dec. 2015, 2 Marks)**

Q. State what is meant by clipping. Explain any one clipping algorithm. *(Refer Section 5.3.1(A))* **(Dec. 2015, 3 Marks)**

Q. Explain Cohen-Sutherland line clipping algorithm. *(Refer Section 5.3.1(A))* **(May 2016, May 2017, 10 Marks)**

Q. Give the line clipping algorithm which uses area codes for the end points of the line segment. *(Refer Section 5.3.1(A))* **(10 Marks)**

Q. Explain Midpoint Subdivision Algorithm. *(Refer Section 5.3.1(B))* **(Dec. 2010, 10 Marks)**

Q. Explain Liang Barsky line clipping algorithm *(Refer Section 5.3.1(C))* **(May 2013, May 2014, Dec. 2014, Dec. 2016, 10 Marks)**

Q. Explain Liang- Barsky line clipping algorithm with suitable example. *(Refer Section 5.3.1(C))* **(May 2015, 10 Marks)**

Q. Explain any one polygon clipping method in detail. *(Refer Section 5.3.2)* **(Dec. 2014, 10 Marks)**

Q. Write short note on : Polygon clipping method. *(Refer Section 5.3.2)* **(Dec. 2015, 10 Marks)**

Q. Explain any one polygon clipping algorithm. *(Refer Section 5.3.2(A))* **(May 2014,May 2016, 10 Marks)**

Q. Explain Sutherland – Hodgeman polygon clipping algorithm in detail. *(Refer Section 5.3.2(A))* **(May 2015, 10 Marks)**

Q. Explain Sutherland-Hodgman Polygon clipping algorithm, with example. What modification is required on it so that it should also work on concave polygon. *(Refer Section 5.3.2(A))* **(10 Marks)**

Q. Explain Sutherland Hodgeman polygon clipping algorithm with example. *(Refer Section 5.3.2(A))* **(10 Marks)**

Q. Explain Weiler-Atherton algorithm for polygon clipping. What are its advantages over other polygon clipping algorithms ? *(Refer Section 5.3.2(B))* **(May 2013, May 2017, 10 Marks)**

Q. Explain Weiler-Atherton algorithm for polygon clipping. *(Refer Section 5.3.2(B))* **(Dec. 2016, 10 Marks)**

Q. Explain how Weiler-Atherton algorithm works for convex polygons? *(Refer Section 5.3.2(B))* **(10 Marks)**

Q. How Welier-Athorton algorithm solves the problem of concave polygon clipping ? *(Refer Section 5.3.2(B))* **(6 Marks)**

Q. Write short note Text clipping methods. *(Refer Section 5.3.4)* **(5 Marks)**

Q. Explain Text clipping with help of appropriate examples. *(Refer Section 5.3.4)* **(10 Marks)**

□□□

CHAPTER 6

Module 5

3D Object Representation and Fractal Geometry

Chapter 6 introduces three-dimensional object representation and fractal geometry. Generally we are using straight lines to draw images on computer screen. But we can draw images by using several sample points, which follows the path of curve. This chapter also describes curves and fractals. This chapter has five sections.

6.1 **Introduction** introduces the need of curves, fractals and 3D object representation.

6.2 **3D Object Representation** method explains different methods such as B-REP, CSG, Sweep and their comparison.

6.3 **Curve Generation** explains different methods to generate any curve. It also explains the concept of B-splines.

6.4 **Bezier curves** are another type of approximating curves, useful in computer graphics. It also explains generation of Bezier surfaces.

6.5 **Fractals** explain what a fractal is and also explains different fractals such as Hilbert's curve and Triadic Koch curve. This section also explains how to generate fractal lines and surfaces.

6.1 Introduction

- Until now we have seen, how to draw lines, circles, polygons etc. To draw a line, we are having different algorithms. By using these algorithms we can easily form any kind of straight line. But in nature the objects are not strictly following these shapes. The objects which are available in nature are generally having rough and curved surfaces. Many artificial objects are also having curved surfaces. These objects are represented by different type of curves and fractals.
- At the same time many real-world objects are inherently smooth, therefore need infinitely many points to model it. This is not feasible for a computer with finite storage. More often we merely approximate the object with pieces of planes, spheres, or other shapes that are easy to describe mathematically.

Syllabus Topic : Boundary Representation and Space partitioning representation- Polygon Surfaces , Sweep Representation, Constructive Solid Geometry , Octree

6.2 3D Object Representation Methods

- Objects are represented as a collection of surfaces. 3D object representation is divided into two categories.

Types of 3D object representation models

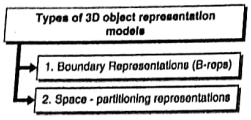


Fig. C6.1 : Types of 3D object representation models

➜ 1. Boundary Representations (B-reps)

It describes a 3D object as a set of surfaces that separates the object interior from the environment.

➜ 2. Space–partitioning representations

It is used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes).

When we are representing a 3D object in computer graphics, the graphics scenes can contain many different kinds of objects. Not a single method is in a position to describe all these methods. If we are using accurate models they produce a realistic display of scenes. We can make use of following methods also.

- Polygon and quadric surfaces
- Spline surfaces and construction techniques
- Procedural methods
- Physically-based modeling methods
- Octree encodings

6.2.1 Polygon Surfaces

- It is a set of surface polygons that enclose interiors of the object. It is a basic form of a 3D object representation. In some systems, all objects are described as polygon surfaces. Polygons are easy to process, so rendering and display of objects is speed up. Thus, some systems allow objects to be described in other ways, such as splines, but reduce all objects to polygons for processing. For a polygon, the representation is exact. For other surfaces, polygons are tiled in a polygon-mesh which approximates the object. Usually, the basic polygon is a triangle.
- Polygon surfaces are commonly used for boundary representation. Here all surfaces are described with linear equations, which simplify and speed up surface rendering and display of objects. By using this we can precisely define a polyhedron. For non-polyhedron objects, the wireframe outline can be display quickly. Polygon surfaces are commonly used in design and solid modeling.

6.2.2 Polygon Tables

- We can specify a polygon surface with a set of vertex coordinates and associated attribute parameters. This information of polygon is placed in tables such as geometric tables and attributes tables. The geometric tables for polygon contain vertex coordinates and parameters to identify the spatial orientation of the polygon surface. On the other hand attribute tables for an object includes parameters specifying the degree of transparency of the object and its surface reflectivity and texture characteristics.
- Geometric data for the polygon is represented using three tables i.e. a vertex table, en edge table and a polygon table. The coordinate values for each vertex in the object are stored in the vertex table. The edge table contains pointers back into the vertex table to identify the vertices for each polygon edge. And the polygon table contains pointers back into the edge table to identify the edges for each polygon.

6.2.3 Plane Equation

- This equation for a plane surface can be expressed as
  $A_x + B_y + C_z + D = 0$
- Where (x, y, z) is any point on the plane, and the coefficients A, B, C and D are constants describing the spatial properties of the plane.

- To obtain the values of A, B, C and D we have to solve a set of three plane equations using the co-ordinate values for 3 nonlinear points in the plane, say three successive vertices of the polygon, (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3). Then solving following simultaneous equations for ratios A/D, B/D and C/D we get the values of A, B, C and D

  $(A/D)\, x_1 + (B/D)\, y_1 + (C/D)\, z_1 = -1$

  $(A/D)\, x_2 + (B/D)\, y_2 + (C/D)\, z_2 = -1$, and

  $(A/D)\, x_3 + (B/D)\, y_3 + (C/D)\, z_3 = -1$

- Using Cramer's rule we can obtain the solution for above equations in the determinant form as

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

- The coefficient of the plane equation can be obtained using a vector cross product calculation. Here, we have to select vertex positions V_1, V_2 and V_3 in counter clockwise order when viewing the surface from outside to inside in a right handed Cartesian system. With three vertex positions we can from two vectors.

  V_1 and V_2 and V_1 to V_3 the cross product of these vectors is

$$N = (V_2 - V_1) \times (V_3 - V_1)$$

- This cross product generates values for coefficients A, B and C for the plane equation. We can then obtain the value for parameter D by substituting these values and the co-ordinates for one of the polygon vertices in the plane equation.
- It is important to note that the resultant vector N is the normal vector to the surface. The plane equation can be expressed in vector form using the normal N and the position P of any point in the plane as

$$N.P = -D$$

- For any point (x, y, z) with parameters A, B, C, D, we can say that
  - Point is not on the plane if $A_x + B_y + C_z + D \neq 0$.
  - Point is inside the surface if $A_x + B_y + C_z + D < 0$
  - Point is outside the surface if $A_x + B_y + C_z + D > 0$

6.2.4 Quadric Surfaces

- Quadric surfaces are most commonly used class of objects. The reason for frequent use of quadric surface are :
  - Ease of computing the surface normal,
  - Ease of testing whether a point is on the surface,
  - Ease of computing z if x and y are given and
  - Ease of calculating intersections of one surface with other.
- Quadric surfaces are described with 2nd degree equations. They include spheres, ellipsoids, paraboloids and hyperboloids.

6.2.5 Solid Modeling

- One of the major field in computer graphics is modeling of objects and pictures. Most geometric objects we see every day are solids.
- Designing representations for solids is a difficult job and compromises are often necessary. In this section, we introduce sweep representation, boundary representations and constructive solid geometry.

6.2.5(A) Sweep Representation

➜ (May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016)

Q. Describe the following 3-D representation method: Sweep representation MU - May 2014, 4 Marks

Q. Write short note: Sweep representation MU - Dec. 2014, May 2015, Dec. 2015, May 2016, 10 Marks

- Sweep representations are used to construct 3D object from 2D shape. There are two ways to achieve sweep.

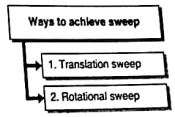


Fig. C6.2 : Ways to achieve sweep

➜ 1. Translation sweep

- In translation sweep, the 2D shape is swept along a linear path normal to the plane of the area to construct 3D object. To obtain the wireframe representation we have to replicate the 2D shape and draw a set of connecting lines in the direction of shape, as shown in Fig. 6.2.1.

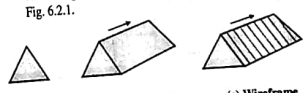


Fig. 6.2.1

➜ 2. Rotational sweep

- In rotational sweep, the 2D shape is rotated about an axis of rotation specified in the plane of 2D shape to produce three dimensional objects.

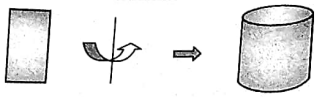


Fig. 6.2.2

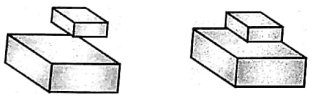
- Fig. 6.2.2 shows a rotational sweep creates a cylindrical object. Sweep representations are widely-used in computer vision. However, the generation of arbitrary objects becomes rather difficult using this technique.

6.2.5(B) Constructive Solid Geometry

→ (May 2014)

Q. Describe the following 3-D representation methods : CSG. **MU - May 2014, 3 Marks**

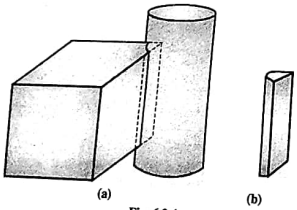
- Another technique for solid modeling is to combine the volumes occupied by overlapping 3D objects using Boolean set operations. This modeling tech. is called Constructive Solid Geometry (CSG). It creates a new volume by applying Boolean operators such as union, intersection, or difference to two specified objects.
- The Fig. 6.2.3, 6.2.4, 6.2.5 show the examples for forming new shapes using Boolean set operations. The Fig. 6.2.3(a) shows two rectangle blocks are place adjacent to each other. We can obtain object with the union operation as shown in Fig. 6.2.3(b).



(a) Objects (b) Combined objects

Fig. 6.2.3 : Combined object by union operator

- The Fig. 6.2.4 shows the result of intersection operation obtained by overlapping cylinder and cube. With the difference operation, we can obtain the resulting solid as shown in Fig. 6.2.5.



(a) (b)

Fig. 6.2.4

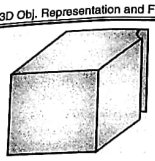


Fig. 6.2.5

- CSG representations are particularly useful for capturing design intent in the form of features corresponding to material addition or removal such as bosses, holes, pockets etc. Some of the important properties of CSG include conciseness, guaranteed validity of solids, computationally convenient Boolean algebraic properties, and natural control of a solid's shape in terms of high level parameters defining the solid's primitives and their positions and orientations.
- The CSG method uses three dimensional objects such as blocks, pyramids, cylinders, cones, spheres, and closed spline surfaces to generate other solid objects.

6.2.5(C) Boundary Representation (B-REPS)

→ (May 2014)

Q. Describe the following 3-D representation methods : B-REPS. **MU - May 2014, 3 Marks**

- Boundary representation, B-rep in short, can be considered as an extension to the wireframe model. The merit of a B-rep is that a solid is bounded by its surface and has its interior and exterior. The surface of a solid consists of a set of well-organized faces, each of which is a piece of some surface. Faces may share vertices and edges that are curve segments. Therefore, B-rep is an extension to the wireframe model by adding face information to the latter.
- There are two types of information in a B-rep topological and geometric. Topological information provides the relationships among vertices, edges and faces similar to that used in a wireframe model. In addition to connectivity, topological information also includes orientation of edges and faces. Geometric information are usually the set of equations of the edges and faces.
- The orientation of each face is important. Normally a face is surrounded by a set of vertices using the right-handed rule, the ordering of these vertices for describing a particular face must guarantee that the normal vector of that face is pointing to the exterior of the solid. Normally, the order is counter clockwise. If that face is given by an equation, the equation must be rewritten so that the normal vector at every point on the part that is being used as a face points to the exterior of the solid. Therefore, by inspecting normal vectors one can immediately tell the inside/outside of a solid under B-rep.

6.2.5(D) Octrees

- Octrees are the hierarchical structure used to represent solid objects in some graphics systems. The tree structure of octrees is organized so that each node corresponds to a region of three dimensional space.
- Octrees are in turn derived from quad-trees, an encoding scheme used for representing two dimensional images. The fundamental idea behind both the quad-tree and octree is the divide-and-conquer power of binary subdivision. A quad-tree is derived by successively dividing a 2D plane (usually square) into quadrants. After first subdivision, each node in the quad-tree has four data elements, one for each of the quadrants in the region, as shown in the Fig. 6.2.6. These quadrants can be of two types : homogeneous or heterogeneous.

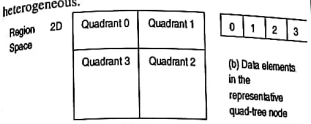
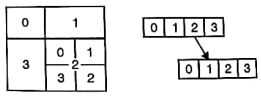


Fig. 6.2.6

- The quadrant is said to be homogeneous quadrant the color information for that quadrant is stored in the corresponding data element of the node. In addition, a flag is set in the data element to indicate that the quadrant is homogeneous. The heterogeneous quadrant contains pixels of different colors. Such quadrant is itself divided into quadrants as shown in the Fig. 6.2.7. The new generated quadrants are checked for homogeneous quadrant. If some of them are heterogeneous then they are again subdivided and the process is repeated until all quadrants are homogeneous or until a predetermined cut-off depth is reached.



(a) Region 2D space (b) Quad-tree representation

Fig. 6.2.7

- Quad-tree encoding scheme provides considerable saving in storage, since each color 'area' is represented by a single node. For an area containing 2n by 2n pixels, a quad-tree representation contains at most n levels.
- In case of black and white image homogeneous quadrants are either completely fill (F) or completely empty (E). Fig. 6.2.8 shows the quad-trees for 2 such images.

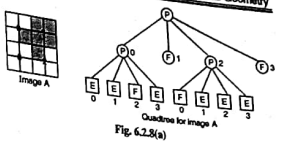


Fig. 6.2.8(a)

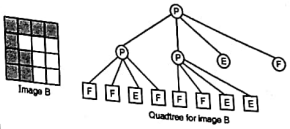
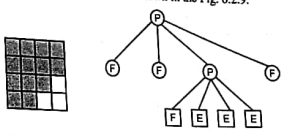
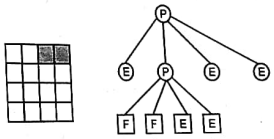


Fig. 6.2.8(b)

- Performing Boolean set operations such as union and intersection on above two quad-trees are get resultant images and quad-trees as shown in the Fig. 6.2.9.



a) Resultant image and quad-tree after union operation (A∪B)



(b) Resultant image and quad-tree after intersection operation (A∩B)

Fig. 6.2.9

6.2.5(E) Comparison of Representations

→ (Dec. 2016)

Q. Write a short note on Comparison of 3D object representation methods. **MU - Dec. 2016, 10 Marks**

Table 6.2.1 : Comparison of representation

Parameter	Sweeps	B-reps	CSG	Octree
Accuracy	High	Sometimes requires approximations	High	High

Parameter	Sweeps	B-reps	CSG	Octree
Domain	Limited	Can represent wide class of objects	Limited to some extend	Limited
Uniqueness	Unique	Unique	Not unique	Unique
Validity	Easy to validate	Most difficult to validate	Easy to validate	Easy to validate
Closure	Not closed under Boolean operations	May suffer from closure problems under Boolean operations.	Closed	Closed
Compactness and efficiency	More compact and efficient	Compact and evaluation is not necessary	More compact needs evaluation of Boolean operations and transformation.	Compact

Syllabus Topic : B-Spline Curve

6.3 Curve generation

- Curves are one of the most essential objects to create high-resolution graphics. While using many small polylines allows creating graphics that appear smooth at fixed resolutions, they do not preserve smoothness when scaled and also require a tremendous amount of storage for any high-resolution image. Curves can be stored much easier, can be scaled to any resolution without losing smoothness, and most importantly provide a much easier way to specify real-world objects.
- All of the popular curves used in graphics are specified by parametric equations. Instead of specifying a function of the form $y = f(x)$, the equations $y = fy(u)$ and $x = fx(u)$ are used. Using parametric equations allows curves that can double back and cross themselves, which are impossible to specify in a single equation in the $y = f(x)$ case. Parametric equations are also easier to evaluate: changing u results in moving a fixed distance along the curve, while in the traditional equation form much work is needed to determine whether to step through x or y, and determining how large a step to take based on the slope.
- For generation of the curves, we are having mainly two options :

  - Use of a curve generation method
  - Approximate the curve

Curve generation method

- Here we can use any readymade curve generation algorithm to draw a curve such as DDA. We can make use of circle generation algorithm also to form different curves. By using algorithm, such as DDA, bresenham circle generation or midpoint circle generation, we will get actual curve. When we are using any circle generation algorithm to draw a curve, at that time, we should not draw a complete circle, but we should draw only some part of that circle which will act as curve.
- When we are using DDA algorithm to generate a circular arc, we need radius and some angle θ to draw a curve. See. Fig. 6.3.1.

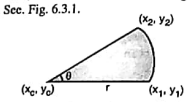


Fig. 6.3.1

- If we want to draw a curve from point (x_1, y_1) to (x_2, y_2) i.e. of angle θ, then we have to find the co-ordinate of (x_1, y_1) as

$$x_1 = x_c + r \quad \text{where r is radius}$$

$$\text{And} \quad y_1 = y_c$$

- This (x_1, y_1) will be the 1st point on the curve. Now to plot a 2nd point on the curve we have to say.

$$x = x_c + r \cos\theta$$

$$y = y_c + r \sin\theta$$

- Where θ will be some stepping angle. Like this we have to find all the points from (x_1, y_1) to (x_2, y_2), on the path of curve, by increasing every time angle θ by some stepping value, till the θ will reach to final angle value.
- Similarly we can use any circle generation algorithm also to draw a curve. See Fig. 6.3.2.

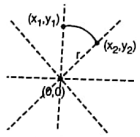


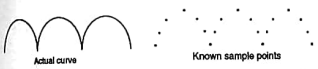
Fig. 6.3.2

- When we are using any circle generating algorithm, we should not use symmetry property of the circle, because here we have to plot only some part of circle. After finding any particular point on the path of curve (by using any circle generation method), we have to plot only that point, we should not replicate it in remaining quadrants or octants.

- So by using either curve generation or circle generation method we can get true curve. But this method has some drawbacks also, such as :
  - We need more information than just its end points.
  - When we scale a line, it remains line only i.e. the basic properties are not changing. But if we scale a curve it may behave differently i.e. when a circle is scaled in one direction, it acts as an ellipse. So basic properties are getting change.
  - Clipping will be difficult for curves.
  - Every curve which we want to draw is not simple to implement by this method.
- So we have to use another method to draw curves, which is called as approximation method.

Approximation

- Instead of using readymade algorithm, we are approximating the curve by small straight lines. For this we have to use **interpolation technique**. We can draw an approximation to a curve if we have an array of sample points. We know that curve is a set of points or pixels. The sample points are nothing but some points on the estimated path of the curve. From these few sample points, we can guess the shape of the curve. If the sample points are closer to each other, we can easily guess a correct curve. Our guess may not be exactly right but it can be acceptable one.
- Suppose we want to draw a curve of shape, which is shown in Fig. 6.3.3(a) and we are having set of sample points as shown in Fig. 6.3.3(b) then,



(a) **(b)**

Fig. 6.3.3

- From the known sample points, we can draw a curve which can pass through the nearby sample points. Now we will try to fit the known curve in these sample points. We can fill the gaps between the sample points. For this we have to find the co-ordinates between two sample points on the curve and then we have to connect these points by using small line segments as shown in Fig. 6.3.4.

Fig. 6.3.4

- But how we are going to find the points which are lying on the path of the curve? Generally to draw a line or circle we need equation of line or circle. With the help of that equation we can find whether the specified point is lying on the boundary of that line or circle. After placing a point in the equation of line or circle, if that equation becomes true, then it means the specified point lies on that line or circle. For line, we are having equation $y = mx + b$. Similarly for circle, we can use equation $r^2 = x^2 + y^2$.
- Similarly, if we want to draw any curve, we must know the equation of that curve. So that we can place a point in that equation and if the equation becomes true it means the specified point is lying on the curve. But here we do not know the equation of the curve.
- There are many forms of the function, which we can make use by adjusting parameters to find the points between the sample points. Normally functions are expressed in the form of $y = f(x)$ i.e. y is a function of x, but to generate a curve by approximate method, we are using parametric form.

$$x = f_x(a)$$

$$y = f_y(a)$$

$$z = f_z(a)$$

- Here we are using parametric form because it considers all three directions equally and it allows multiple values so that curve can cross itself.
- Suppose we want a curve that passes through n sample points.

$$(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots (x_n, y_n, z_n)$$

- As we are using parametric form, the equation of the curve should be $x = f_x(a)$. It means we have to construct some function for this curve. As this curve is passing from n sample points the function must be the sum of the n terms i.e. one term for each sample point. Therefore,

$$f_x(a) = \sum_{i=1}^{n} x_i B_i(a)$$

$$f_y(a) = \sum_{i=1}^{n} y_i B_i(a)$$

$$f_z(a) = \sum_{i=1}^{n} z_i B_i(a)$$

- Here for each function $f_x(a)$, $f_y(a)$ and $f_z(a)$ we are using one common function $B_i(a)$. This function $B_i(a)$ is called as **Blending function.**
- To achieve a perfect curve, it must pass through all the sample points. It means each sample point tries to pull the curve in his direction, as shown in Fig. 6.3.5.

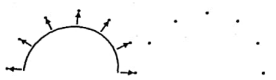


Fig. 6.3.5

- The curve will pass through the i^{th} sample point if for same value of 'a', $B_i(a) = 1$. It means the curve is passing through that point. The curve will not pass through the i^{th} sample point if the blending function's value is zero. If at a different value of 'a', one of the other sample points has complete control, then the curve will pass through this point as well. Now we have to develop a blending function such that its value could be 1 for the starting sample point. The value of 'a' is – 1 for 1^{st} point, 0 for 2^{nd}, 1 for 3^{rd} and so on.
- So our requirement is the blending function's values for the 1^{st} sample point must be equal to 1. It is slightly reverse way. Here we know result i.e. functions value must be 1 and from this we have to develop a function.

$$\text{i.e. } B_i(a) = 1$$

So we have to find,

$$B_i(a) = ?= 1$$

- To generate a smooth curve, it must pass through each sample point. For this we need a function which will be 1 for 'a' = – 1 i.e. for 1^{st} sample point and 0 for 'a' = 0, 1, 2, ..., n – 2 i.e. for others.

$\therefore$ It could be, $a(a-1)(a-2)\ldots(a-(n-2))$

at $a = -1$,

$$(-1)(-2)(-3)\ldots(1-n)$$

$$= \text{some constant}$$

But if we need $B_i(a) = 1$, then we will write as

$$B_i(a) = \frac{\text{"XYZ"}}{const} = 1$$

In order to get $B_i(a) = 1$ the value of "XYZ" must be $a(a-1)(a-2)\ldots(a(n-2))$

$$\therefore B_i(a) = \frac{a(a-1)(a-2)\ldots(a(n-2))}{(-1)(-2)(-3)\ldots(n-1)}$$

Let's consider a case where there are four sample points. In such a situation we need four blending functions.

$$B_1(a) = \frac{a(a-1)(a-2)}{(-1)(-2)(-3)}$$

$$B_2(a) = \frac{(a+1)(a-1)(a-2)}{(1)(-1)(-2)}$$

$$B_3(a) = \frac{(a+1)(a)(a-2)}{(2)(1)(-1)}$$

$$B_4(a) = \frac{(a+1)(a)(a-1)}{(3)(2)(1)}$$

- Using these functions and four sample points, we can construct a curve which passes through the four sample points.

$$x = x_1 B_1(a) + x_2 B_2(a) + x_3 B_3(a) + x_4 B_4(a)$$

$$y = y_1 B_1(a) + y_2 B_2(a) + y_3 B_3(a) + y_4 B_4(a)$$

$$z = z_1 B_1(a) + z_2 B_2(a) + z_3 B_3(a) + z_4 B_4(a)$$

- We can also interpolate polygons. By using our blending functions, the sides of any polygon can also be rounded, rather the polygons are easy to handle. We just step around the polygon smoothing out each side by replacing it with several small line segments. We start out with a polygon that has only a few sides and end up with a polygon which has many more sides and appear smoother.

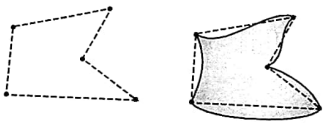
6.3.1 B-spline Curves and Corners

→ (May 2017)

Q. Write short note on: B-spline.

MU - May 2017. 10 Marks

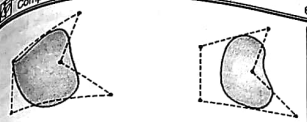
- We have already seen interpolation technique to generate a smooth curve, it must pass through all sample points. It means the sum of blending functions must be 1, for every sample point for integer values of 'a' we may get blending function as 1 but for fractional values of 'a' we may not be able to pass the curve through the sample points.
- Suppose there are five sample points as shown in Fig. 6.3.6(a). We may fix up a curve from these sample points as shown in Fig. 6.3.6(b), if the blending function is 1 for all values of 'a'.



(a) (b)

Fig. 6.3.6

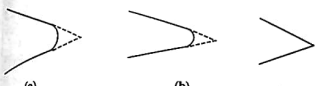
- If we see Fig. 6.3.6(b) carefully, we will come to know that, each section of the curve is connected to the other section of curve at sample point. But the slopes of two sections may be different so we are getting corners in the curve.
- It means the curve which we are getting is not smooth. In order to go near to the smoothness of the curve, we will pull the curve in such a way that the curve will pass through one sample point only. It means the blending function will be 1 for only one sample point. By using this technique we may eliminate the sharp corners. See Fig. 6.3.7(a).



(a) (b)

Fig. 6.3.7

- But this will not give us natural appearance of the curve. So here, we will not try to force the curve to pass through the sample points, but rather gently pull it into the neighbor-hood of the sample point. So by doing this we will get a smooth curve which is not passing through any of the sample points but still maintains the original shape of the curve as shown in Fig. 6.3.7(b).
- A set of blending functions which takes this approach is called **B-splines** Basically spline means a strip, which we have to move around the sample points. Generally the B spline blending functions were designed to eliminate sharp corners in the curve and the curve does not usually pass through the sample points. But if we need sharp corners we can produce it by using many identical sample points.



(a) (b) (c)

Fig. 6.3.8

- Fig.6.3.8(a) shows the curve using one sample point. Fig. 6.3.8(b) shows the curve using two identical sample points. It means we are calling the blending function for same samples point twice. And Fig. 6.3.8(c) shows if three identical sample points are used the curve will be forced to pass through the sample point.

6.3.2 Properties of B-spline Curve

→ (Dec. 2016)

Q. Give the properties of B-spline curve.

MU - Dec. 2016. 10 Marks

1. The sum of the B-spline basis functions for any parameter value u is 1 i.e.

$$\sum_{i=1}^{n+1} N_{i,k}(u) \equiv 1$$

2. Each basis function is positive or zero for all parameter values, i.e., $N_{i,k} >= 0$.
3. Except for k = 1 each basis function has precisely one maximum value.
4. The maximum order of the curve is equal to the number of vertices of defining polygon.
5. The degree of B-spline polynomial is independent of the number of vertices of defining polygon.
6. B-spline allows local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
7. The curve exhibits the variation diminishing property. Thus the curve does not oscillate about any straight line move often than its defining polygon.
8. The curve generally follows the shape of defining polygon.
9. Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
10. The curve line within the convex hull of its defining polygon.

Syllabus Topic : Bazier Curve, Bazier Surface

6.4 Bezier Curve

→ (May 2014, Dec. 2014, Dec. 2015, May 2016)

Q. Write short note on : Bezier curves.

MU - May 2014, May 2016. 10 Marks

Q. Explain what is meant by Bezier curve. Also explain how a Bezier surface can be generated from Bezier curve.

MU - Dec. 2014. 3 Marks. Dec. 2015. 10 Marks

Q. State mathematical equations for Bezier Curves.

(5 Marks)

Q. Give mathematical equations for Bezier curve. 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 curve.

(6 Marks)

- It is a different way of specifying a curve, rather same shapes can be represented by B-spline and Bezier curves. The cubic Bezier curve require four sample points, these points completely specify the curve. Fig. 6.4.1 shows sample Bezier curves.

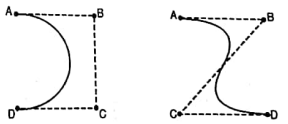


Fig. 6.4.1

- The curve begins at the first sample point and ends at fourth point. If we need another Bezier curve then we need another four sample points. But if we need two Bezier curves connected to each other, then with six sample points we can achieve it. For this, the third and fourth point of first curve should be made same as first and second point of second curve.

- The equations for the Bezier curve are as follows :

$$x = x_4 a^3 + 3x_3 a^2 (1-a) + 3x_2 a (1-a)^2 + x_1 (1-a)^3$$
$$y = y_4 a^3 + 3y_3 a^2 (1-a) + 3y_2 a (1-a)^2 + y_1 (1-a)^3$$
$$z = z_4 a^3 + 3z_3 a^2 (1-a) + 3z_2 a (1-a)^2 + z_1 (1-a)^3$$

- Here as the value of 'a' moves from 0 to 1, the curve travels from the first to fourth sample point. But we can construct a Bezier curve without referencing to the above expression. It is constructed by simply taking midpoints. See Fig. 6.4.2.

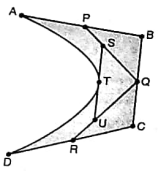


Fig. 6.4.2

- The points A, B, C, D are the original Bezier curve control points. Here we are having three lines, AB, BC and CD, then we have to find the midpoints of these lines as 'P', 'Q' and 'R' respectively. After that we have to join PQ and QR. Then again find the mid point of these newly generated lines as 'S' and 'U'. Then form a line segment between 'S' and 'U'. And find midpoint of this line as 'T'. Now point 'T' will be on Bezier curve. This point 'T' divides the curve into two section, one is (A, P, S and T) and second will be (D, R, U and T).
- Thus by taking midpoints, we can find a point on the curve and also split the curve into two sections. We can continue to split the curve into smaller sections, until we have sections so short that they cannot be replaced by straight lines or till the size of section is not greater than the size of pixel.

Cubic Bezier Curve

- Cubic curves are commonly used in graphics because curves of lower order commonly have too little flexibility, while curves of higher order are usually considered unnecessarily complex and make it easy to introduce undesired wiggles. To join multiple curves into one curve smoothly, we need to specify the positions and their tangent-vectors of both ends, i.e. the continuity requirements
- Four points P_0, P_1, P_2 and P_3 in the plane or in three-dimensional space define a cubic Bezier curve. The curve starts at P_0 going toward P_1 and arrives at P_3 coming from the direction of P_2. Usually, it will not pass through P_1 or P_2; these points are only there to provide directional information. The distance between P_0 and P_1 determines "how long" the curve moves into direction P_2 before turning towards P_3. See Fig. 6.4.3.
- A cubic Bezier curve indirectly specifies the endpoint tangent-vector by specifying two intermediate points that are not on the curve.

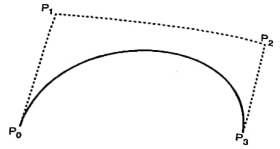


Fig. 6.4.3

6.4.1 Properties of Bezier Curve

➔ (May 2013, May 2014, Dec. 2014, May 2015, May 2016, Dec. 2016)

Q. State the properties of Bezier curves. **MU - May 2013, 10 Marks**

Q. Explain properties of Bezier Curve. **MU - May 2014, May 2015, May 2016, Dec. 2016, 10 Marks**

Q. State properties of Bezier curve and hence explain how a Bezier surface can be generated from Bazier curve. **MU - Dec. 2014, 7 Marks**

Q. Give properties for Bezier curve. 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 curve. **(6 Marks)**

1. The basic functions are real in nature.
2. Bezier curve always passes through the first and last control points i.e. curve has same end points as the guiding polygon.
3. The degree of the polynomial defining the curve segment is one less than the number of defining polygon point.
   Therefore, for 4 control points, the degree of the polynomial is three, i.e. cubic polynomial.
4. The curve generally follows the shape of the defining polygon.
5. The direction of the tangent vector at the end points is the same as that of the vector determined by first and last segments.
6. The curve lies entirely within the convex hull formed by four control points.
7. The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
8. The curve exhibits the variation diminishing property. This means that the curve does not oscillate about any straight line more than the defining polygon.
9. The curve is invariant under an affine transformation.

6.4.2 Advantage of B-splines over Bezier Curves

B-spline is a way to construct a curve that is interpolated between three or more points, it is a kind of spline generated by so-called basis functions. Bezier curves are B-splines, But the control points are different. B-splines are not used very often in 2D graphics software but are used quite extensively in 3D modeling software.

1. The control points of a B-spline affect only their local region of the curve or surface.
2. B-splines compute faster than Bezier curves.
3. They are smoother and easier to control.
4. The continuous curve of a b-spline is defined by control points.
5. The control points give a better idea of the shape of the curve.
6. They do not have the ripple associated with the cubic splines.
7. Numerically more stable.
8. Fewer bumps and wiggles.

Comparison between Bezier curve and B-spline curve generation methods

Following table represents the comparison of Bezier curve and B-spline curve generation methods.

Sr. No.	Bezier Curve Generation	B-Spline Curve Generation
1.	Any shape of the curve is achieved by Bezier curve.	By using Spline, we can achieve smooth curves
2.	Depending on the sequence of control points the shape of the curve gets change	Depending on the sequence of control points the shape of the curve is not changing
3.	Sharp corners are achieved by passing the curve from the sample points	We can achieve the sharp corners by using the blending function repeatedly for the same point
4.	The is always passing from minimum two points, i.e. starting and ending point.	In a special case the curve may not pass from any of the sample point

Program for Bazier Curve generation

```
/*********************************************
Program : A program to generate a Bezier Curve by using
mouse interface
*********************************************/
#include<stdio.h>
#include<process.h>
#include<conio.h>
#include<iostream.h>
#include<dos.h>
#include<graphics.h>
union REGS i,o;
float arr[4][2];
int cnt;
/*==========================
Function Name:      bezier().
Purpose  :  To draw a curve by using mid point method.
==========================*/
void bezier(float xb,float yb,float xc,float yc,float xd,float yd,int cnt)
{
    float xp,yp,xq,yq,xr,yr,xs,ys,xu,yu,xt,yt;
    if(cnt!=0)
    {
        xp=(arr[0][0]+xb)/2;
        yp=(arr[0][1]+yb)/2;
        putpixel(xp,yp,4);
        xq=(xb+xc)/2;
        yq=(yb+yc)/2;
        putpixel(xp,yp,4);
        xr=(xc+xd)/2;
        yr=(yc+yd)/2;
        putpixel(xp,yp,4);
        xs=(xp+xq)/2;
        ys=(yp+yq)/2;
        putpixel(xp,yp,4);
        xu=(xq+xr)/2;
        yu=(yq+yr)/2;
        putpixel(xp,yp,4);
        xt=(xs+xu)/2;
        yt=(ys+yu)/2;
        putpixel(xp,yp,4);
        cnt = cnt--
        delay(10);
        bezier(xp,yp,xs,ys,xt,yt,cnt);
        delay(10);
        bezier(xu,yu,xr,yr,xd,yd,cnt);
        delay(10);
    }
    else
    {
        line_draw(xb,yb);
        line_draw(xc,yc);
        line_draw(xd,yd);
    }
}
```

```
/*============================
Function Name:  line_draw().
Purpose     : To join the sample points of the curve by straight
line.
============================*/
void line_draw(float x,float y)
{
      line(arr[0][0],arr[0][1],x,y);
      arr[0][0]=x;
      arr[0][1]=y;
}
/*============================
      Member Name :     initmouse().
      Purpose     :     To initialize the mouse.
============================*/
initmouse()
{  i.x.ax=0;
     int86(0x33,&i,&o);
     return(o.x.ax);
}
/*============================
Member Name :     showmouse().
      Purpose :     To show the mouse.
============================*/
void showmouse()
{
     i.x.ax=1;
     int86(0x33,&i,&o);
}
/*============================
      Member Name :     hidemouse().
      Purpose     :     To hide the mouse.
============================*/
void  hidemouse()
{
     i.x.ax=2;
     int86(0x33,&i,&o);
}
void  restrictmouse(int x1,int y1,int x2,int y2)
{
     i.x.ax=7;
     i.x.cx=x1;
     i.x.dx=x2;
     int86(0x33,&i,&o);
     i.x.ax=8;
     i.x.cx=y1;
     i.x.dx=y2;
     int86(0x33,&i,&o);
}
void main()
{
     clrscr();
     int gd=DETECT,gm, P = 8
     initgraph(&gd,&gm,"c:\\tcplus\\bgi");
     if(initmouse()==0)
          exit(0);
     showmouse();
     int j;
     for(j=0;j<4;j++)
     {
          while(1)
          {
               i.x.ax=3;
               int86(0x33,&i,&o);
               if(o.x.bx==1)
               {
                    arr[j][0]=o.x.cx;
                    arr[j][1]=o.x.dx;
                    break;
               }
          }
          while(1)
          {
               i.x.ax=3;
               int86(0x33,&i,&o);
               if(o.x.bx==0)
                    break;
          }
     }
     for(j=0;j<3;j++)
          line(arr[j][0],arr[j][1],arr[j+1][0],arr[j+1][1]);
bezier(arr[1][0],arr[1][1],arr[2][0],arr[2][1],arr[3][0],arr[3][1],
P);
     getch();
}
/*======== End of Program ==========*/
```

Example 6.4.1

Obtain the curve parameters for drawing a smooth Bezier curve for the following control points : A (0, 0), B(10, 40), C(70, 30), D(60, – 20).

Solution :

Given : Four control points A(0, 0), B(10, 40), C(70, 30), D(60, – 20) are given and from these points we need to draw a Bezier curve. For drawing a Bezier curve we are having two ways. In first method we can use following equation of Bezier curve

$$x = x_4 a^3 + 3x_3 a^2 (1-a) + 3x_2 a (1-a)^2 + y_1 (1-a)^3$$

$$y = y_4 a^3 + 3y_3 a^2 (1-a) + 3y_2 a (1-a)^2 + y_1 (1-a)^3$$

Here the values of x_1 y_1 to x_4 y_4 are already mentioned in given. The value of a will vary from 0 to 1 as the curve travels from the first sample point to fourth one.

In second approach we will not use equations rather we will use the concept of midpoints. Let's draw the four sample points and join them by straight line.

Fig. P. 6.4.1(a)

Now we have to find midpoint of a line AB by

$$x_m = \frac{x_1 + x_2}{2} \text{ and}$$

$$y_m = \frac{y_1 + y_2}{2}$$

$\therefore (x_m, y_m)$ for line AB will be

$$x_m = \frac{0+10}{2} \; ; \; x_m = 5$$

$$y_m = \frac{0+40}{2} \; ; \; y_m = 20$$

$\therefore$ We will get new point (5, 20). Let's give name to this point as P_1.

Similarly we will find midpoints of lines BC and CD. We will call them as P_2 and P_3 respectively. By joining these mid points by straight line we will get following Fig. P. 6.4.1(b).

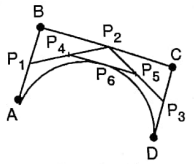


Fig. P. 6.4.1(b)

Now we will find the midpoints of this new line P_1, P_2 and P_2, P_3 as P_4 and P_5 respectively.

By joining these two points P_4 and P_5 by straight line we win get following Fig. P. 6.4.1(c). We will find midpoint of line P_4, P_5 as point P_6.

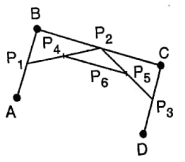


Fig. P. 6.4.1(c)

Earlier we were having only four sample points A,B,C,D. Now we are having A, P_1, P_4, P_6 as one set of four sample points and P_6, P_5, P_3, D as another set of four sample points. Here point P_6 is shared by both set of sample points. So we can consider the curve as a continuous curve.

Now for the next iteration we have to consider one set of four sample points and again carry out the same procedure till the length of line segment becomes equal to one pixel. At last stage we will get the following curve as a final Bezier curve.

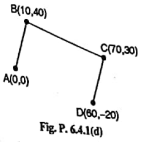


Fig. P. 6.4.1(d)

6.4.3 Bezier Surfaces

- Bezier surfaces are an extension of the idea of Bezier curves. They share many of their properties. As with the Bezier curve, a Bezier surface is defined by a group of control points. But Bezier surfaces does not pass from central control point instead it is elongated in its direction. They are visually intuitive, and for many applications, mathematically convenient.

Properties of Bazier curve

- The properties of the Bezier curve also applied to the Bezier surface.
  1. Generally the surface does not pass from all the control points. But from corner points the surface does pass.
  2. The surface lies within the convex hull of its defining points.
- Along the edges of the grid patch the Bezier surface matches that of a Bezier curve through the control points along that edge. See Fig. 6.4.2.
- Whenever the first and last control point is one and the same at that time we get closed surfaces. We can get first order continuity, if the tangents also match between the first two and last two control points for the closed surface. We can form a cylinder/cone from a Bezier surface, but it is not possible to form a sphere from Bezier Surface.

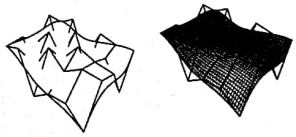


Fig. 6.4.2

Syllabus Topic : Fractal-Geometry - Fractal Dimension, Koch Curve

6.5 Fractals

→ (May 2017)

Q. Write short note on : Fractals
MU - May 2017, 5 Marks

Q. What are Fractals ? Give classification of Fractals. What is Fractal dimension ? **(8 Marks)**

- The objects which are having smooth surfaces and regular shapes are generally described by using equations. But natural objects such as mountains, trees, ocean waves and clouds have irregular shapes. It will be very difficult to draw these shapes by using normal equations. There are many methods of modeling these natural objects, but one of the most interesting from a mathematical perspective is that of fractals. So we can describe natural objects by using fractals, where procedures rather than equations are used to model the objects. Procedurally defined objects have characteristics quite different from objects described with equations.
- One of the basic properties that characterize fractals is self-similarity. The self similarity property of an object can take different forms, depending on the choice of fractal representation. Self-similarity means if we zoom into a piece of a fractal we will keep seeing the same structures repeated over and over.
- From a certain distance we will see a coastline as a simple, quite smooth line. See Fig. 6.5.1. But as we go near to that line, it will appear more rough. If we take closer view we will see the whole line as jaggy and rough. There is no limit how many times we can zoom in.

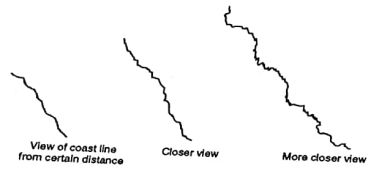


Fig. 6.5.1

- Imagine that one object is made up of clay. If we break that object in terms of line or line segments, then we will give the dimension $D_t = 1$. If the object is broken into a plane, then we will give the dimension as $D_t = 2$. And if the object is broken into 3D objects like cube, sphere etc. then we will given the dimension as $D_t = 3$. Here the variable D_t is called as **topological dimension**.
- Now imagine that we have a thread of say 5 cm. If we cut this in five equal parts i.e. each sub thread's length will be 1 cm. It means we have scaled the thread i.e. line, by 1/5. Now if we want to build original object from the scaled one, then how many scaled objects will be require to form the original object? Obviously 5. Fig. 6.5.2 explains this concept in more detail.
- Fig. 6.5.2 shows a line AB having length (L) = 5 cm. If we break this line in five equal parts, i.e. Number of pieces (N) = 5, then each parts length (l) will be 1 cm.

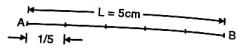


Fig. 6.5.2

$$\therefore \quad l = \frac{L}{N} = \frac{5}{5} = 1 \text{ cm}$$

Here we can define a scaling factor as,

$$= \frac{1}{S}$$

Where S = Number of scaled objects

In this case $S = 5$

$\therefore$ We can say $N = S^1$ ...(6.5.1)

- Now consider a square, as shown in Fig. 6.5.2(a) and if we want to scale it by ½, then how many small squares will get generated? The answer is four. See Fig. 6.5.2(b).

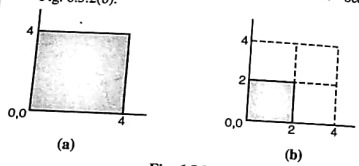


Fig. 6.5.2

- Now if we want to build the original square again, then we need N number of scaled objects i.e. 4.

So here $S = 2$ and $N = 4$. $\therefore \quad N = S^2$ ...(6.5.2)

- Similarly if we have a 3D object, like cube, and if we scale it as half, as shown in Fig. 6.5.3, then we need 8 scaled objects to form original object.

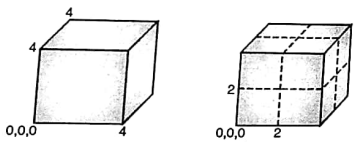


Fig. 6.5.3

Here $S = 2$ and $N = 8$.

$\therefore \quad N = S^3$ ...(6.5.3)

- If we observe the Equations (6.5.1), (6.5.2) and (6.5.3) we will come to know that there is some relation i.e. the exponent goes on increasing.

So we can generalize these equations as $N = S^{D_f}$

- It means if we scale an object by 'S' and must assemble 'N' of scaled objects to reconstruct the full sized/original object, then the dimension 'D_f' of the object is given by,

$$D_f = \frac{\log N}{\log S}$$

where 'D_f' is called as fractal dimension.

6.5.1 Hilbert's Curve

- Let us see how to generate Hilbert's curve. This curve is also called as Peano curve. It is very easy to implement. The curve begins with initial square. The generation of curve requires successive approximations. In the first approximation, we are dividing the square into four quadrants and then drawing the curve which connects the center points of each quadrant, as shown in Fig. 6.5.4(a).

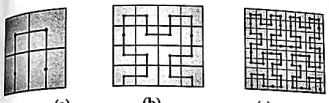


Fig. 6.5.4

- The second approximation will be to further subdivide each of the four quadrants and draw the curve which connects the center points of each of these finer subdivisions before moving the next major quadrant as shown in Fig. 6.5.4(b). The third approximation again subdivides each quadrant and process continues. Looking at curve carefully, we will come to know that the curve is not getting overlap at any point. Curve is arbitrarily close to every point in the square. There is no limit to this subdivision. The curve fills the square. Ideally the length of the curve is infinite. With each subdivision the length increases by factor 4. Since this curve is equivalent to line only, its topological dimension (D_t) must be equal to 1.
- At each subdivision the scale changes by 2 i.e. at every approximation we are dividing plane into four quadrants. But the length of the curve changes by 4, at each subdivision. So we need 4 scaled curves to form original curve.

i.e. $N = S^{D_f}$

$\therefore 4 = 2^{D_f}$

$\therefore D_f = 2$

- So, the Hilbert curve has topological dimension (D_t) = 1 and fractal dimension (D_f) = 2. So we can say, it is a line only, but it is so folded that it looks likes a 2D object.

6.5.2 Triadic Koch Curve

Q. Define Koch curve ? How do you construct the Koch curve ? → (May 2015, Dec. 2016)
MU - May 2015, 10 Marks

Q. Write a short note on Construction of Koch curve.
MU - Dec. 2016, 10 Marks

- To draw a Triadic Koch curve, we have to start with a line segment. Then we are dividing that line segment into three equal parts. After that we are replacing the central third part by the two adjacent sides of an equilateral triangle as shown in Fig. 6.5.5. This gives us a curve whose end co-ordinates are same as that of original segment but is built of 4 equal length segments, each 1/3 the original length. So the new length of the curve will be 4/3 of the original length. We have to repeat this process for each of the 4 segments.

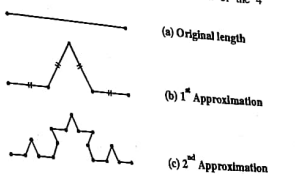


Fig. 6.5.5

- The Fig. 6.5.5(b) and (c) shows appearance of Koch curve after 1st approximation and 2nd approximations, respectively. After second approximation the length of the curve becomes 16/9 times the original.

Now we can easily say that,

- o The length of Koch curve is infinite.
- o This curve doesn't fill the whole area, like Hilbert's curve.
- o It doesn't deviate much from its original shape.
- o If we reduce the scale of the curve by 3, we find the curve looks just like the original one; but we must gather 4 such curves to make the original, so we will say

$$4 = 3^{D_f}$$

Solving for D_f, we get

$$D_f = \frac{\log 4}{\log 3}$$

$$\therefore D_f = 1.26$$

- Therefore for Koch curve topological dimension is one and fractal dimension is 1.26. From this discussion we

can say that, curves whose fractal dimension is greater than its topological dimension then they are called as **fractals**. So Hilbert's curve and Koch curve's are fractals, since their fractal dimension are greater than their topological dimensions.

Program for Koch curve generation

```
/*********************************************
Program: Write a C++ program to generate fractal pattern by
using Koch curve.

*********************************************/

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

/* ============================
Class Definition
----------------------------*/


class curve
{
   public:
      int x1,y1,x2,y2;
float n1,n2,n3,dist,dist2,dist3,midx,midy,center,angle,nx,ny;
float nx1,ny1,sx,sy,sx1,sy1,sx2,sy2,sx3,sy3;
float xb1,xb2,yb1,yb2,tp1x,tp2x,tp1y,tp2y,newpx,newpy;

void koch()
{
cout<<"\nEnter line co-ordinates:";
cin>>x1>>y1>>x2>>y2;
line(x1,y1,x2,y2);

//.....1st Approximation.....
dist=(x2-x1)/3;
setcolor(BLACK);
line(x1+dist,y1,x2-dist,y2);
midx=((x1+dist)+(x2-dist))/2;
setcolor(WHITE);
center=dist/2;
angle=1.73*center;
nx=x1+dist+center;
ny=y1-angle;

//.....2nd Approximation.....
xb1 = x1+dist;
yb1 = y1;
xb2 = nx;
yb2 = ny;

tp1x = (2*xb1 + xb2)/3;
tp1y = (2*yb1 + yb2)/3;
tp2x = (2*xb2 + xb1)/3;
tp2y = (2*yb2 + yb1)/3;
newpy = tp2y;
newpx = (tp2y + (1.73)*tp1x - tp1y)/1.73;
setcolor(WHITE);
line(newpx,newpy,tp2x,tp2y);
line(newpx,newpy,tp1x,tp1y);
line(xb1,yb1,tp1x,tp1y);
line(tp2x,tp2y,xb2,yb2);

xb1 = x2-dist;
yb1 = y1;
xb2 = nx;
yb2 = ny;
tp1x = (2*xb1 + xb2)/3;
tp1y = (2*yb1 + yb2)/3;
tp2x = (2*xb2 + xb1)/3;
tp2y = (2*yb2 + yb1)/3;

newpy = tp2y;
newpx = (tp1y + (1.73)*tp1x - tp2y)/1.73;
setcolor(WHITE);
line(newpx,newpy,tp2x,tp2y);
line(newpx,newpy,tp1x,tp1y);
line(xb1,yb1,tp1x,tp1y);
line(tp2x,tp2y,xb2,yb2);

//.....3rd Approximation.....

setcolor(BLACK);
dist2=(dist)/3;
line(x1+dist2,y1,x1+dist-dist2,y1);
line(x2-(2*dist2),y2,x2-dist2,y2);
setcolor(WHITE);
center=dist2/2;
angle=1.73*center;
nx=x1+dist2+center;
ny=y1-angle;
center=dist2/2;
angle=1.73*center;
nx1=x2-dist2-center;
ny1=y2-angle;
line(x1+dist2,y1,nx,ny);
line(x1+dist-dist2,y2,nx,ny);
line(x2-dist2,y2,nx1,ny1);
```



```
line(x1+dist2,y2,nx1,ny1);
}

void main()
{
curve k;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
k.koch();
getch();
closegraph();
}
```

Output

6.5.3 Fractal Lines

- We know that a curve is composed of N self-similar pieces and each piece is properly scaled. Now if we want to draw a coastline, by using fractal line, ideally it is infinite, but as we are using computer to generate that curve, we must terminate our program some where. So the terminating condition could be the length of the curve. i.e. when the length of the curve becomes less than the size of pixel, we should terminate our program.
- To generate the fractal line we have to perform following steps :
- If the user provides two endpoints of a line as (x_1, y_1, z_1) and (x_2, y_2, z_2) then we have to find the halfway point for the fractal line.
- The halfway point is calculated from the midpoint of a line. See Fig. 6.5.6.

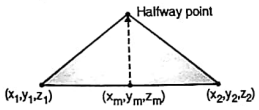


Fig. 6.5.6

The midpoint (x_m, y_m, z_m) will be calculated as,

$$x_m = \frac{x_1 + x_2}{2}$$

$$y_m = \frac{y_1 + y_2}{2}$$

$$z_m = \frac{z_1 + z_2}{2}$$

But in order to achieve the halfway point we have to add some offset to each co-ordinate midpoint.

i.e. $x_m = x_m + dx$

where $x_m = \frac{x_1 + x_2}{2}$

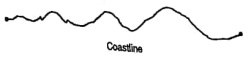
i.e. $x_m = \frac{x_1 + x_2}{2} + dx$

Similarly $y_m = \frac{y_1 + y_2}{2} + dy$

$$z_m = \frac{z_1 + z_2}{2} + dz$$

Here this offset dx, dy and dz must be random values.

- Basically the necessity of random value is that, in nature the objects are not smooth. In case of coastline, we will never see a natural coastline as a straight line. At some position it may be inside and at other position it may be at opposite side as shown in Fig. 6.5.7. So to get this random effect we are using offset which is random value.



Coastline

Fig. 6.5.7

We can calculate this random offset as shown below,

dx = L * W * Guass
dy = L * W * Guass
dz = L * W * Guass

Where
L = length of segment
W = Weight factor for roughness of the curve (i.e. fractal dimension)

- Guass = some random average value (i.e. between – 1 and 1)
- This halfway point divides the original line into two parts. Again repeat the same procedure for each part separately.

Fig. 6.5.8

- To draw a fractal line the algorithm will contain a recursive procedure and the parameters for that procedure will be starting point, ending point, number of iterations and offset. Fig. 6.5.8 shows the generation of coastline by using fractal lines.

6.5.4 Fractal Surfaces

- We can draw coastline or lightning bolt by fractal lines. But if we want to draw a 3D object like, say mountain, then we have to use fractal surfaces.
- The concept of fractal lines can be extended to generate fractal surfaces. There are many ways by which we can do this. Here we are explaining a method which is based on triangle. We know that triangle has three vertex points. We can generate a fractal surface for the area between these three vertices. As shown in Fig. 6.5.9.

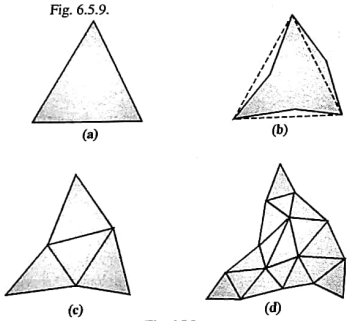


Fig. 6.5.9

- The fractal surfaces can be generated by performing following steps :
  - We will use fractal line algorithm to each edge of the triangle.
  - Compute its halfway point by the same logic as that of fractal lines. As shown in Fig. 6.5.9(b).
  - Connect these halfway points by line segments. By doing this we are now having four small triangles as shown in Fig. 6.5.9(c).
- Now we can call this procedure recursively to each of small triangle until the triangles becomes too small as shown in Fig. 6.5.9(d).

Application of Fractal lines

(i) To draw coast line in animation and movies.

(ii) To draw lightening in animation and movies.

Application of Fractal surfaces

(i) To draw mountains to give 3D effect in movies.

(ii) To draw any object which gives 3D effect in applications.

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 What is fractal dimension and topological dimension ?

Ans. :

The answer for this question should include following points :

- What are fractals.
- Explain topological dimension by giving example of line, square and cube.
- Derive an equation D = log N/log S.
- Explain fractal dimension.

Q. 2 Explain Hilbert curve or Koch curve.

Ans. :

The answer for this questions should include :

- Diagram of Hilbert / Koch curve at initial approximation.
- Explanation for generation of the curve.
- Calculations for fractal dimension.
- Diagram for 2nd and 3rd approximation.
- State advantages and disadvantages.

Q. 3 Explain how to generate fractal lines and from that fractal surfaces.

Ans. :

The answer for this question should include :

- Explanation of fractal lines.
- Generation of offset and halfway point.
- Draw the diagrams wherever necessary.
- Explanation of fractal surfaces.
- Explain triangle method.
- Draw diagrams and explain.

Q. 4 Explain curve generation by using approximation method.

Ans. :

To answer this question we have to include following points.

- Need of sample points.
- Need of equation of the curve.
- Blending function.

- Final formulas for blending functions.
- Explain above points with the help of diagrams.
- Write a short note on Bezier curve.

Q. 5 **Ans. :** The answer for this question should include.

- Need of sample points.
- Information about blending functions.
- Bezier curve generations by using midpoint method.
- Take an example and explain.
- Draw the diagrams whenever necessary.

Important Assignments

This chapter is important from practical examination point of view. We have already explained and given enough number of programs at the time of explaining algorithms. In addition to these programs, in final practical examination, few modifications are also expected. The list of such programs is given below :

Q. 1 Write a program to draw a Bezier curve from four sample points.

Q. 2 Write a program to draw a Bezier curve for 'n' number of sample points. Accept 'n' from user.

Q. 3 Write a program to generate Hilberts curve.

Q. 4 Write a program to generate Triadic Koch curve.

Q. 5 Write a program to generate lightning bolt by using fractal line concept.

Q. 6 Write a program to generate mountain by using fractal surface.

6.6 Exam Pack (University and Review Questions)

Syllabus Topic : Boundary Representation and Space partitioning representation- Polygon Surfaces, Sweep Representation, Constructive Solid Geometry , Octree,

Q. Describe the following 3-D representation method: Sweep representation. *(Refer Section 6.2.5(A))* **(May 2014, 4 Marks)**

Q. Write short note: Sweep representation. *(Refer Section 6.2.5(A))* **(Dec. 2014, May 2015, Dec. 2015, May 2016, 10 Marks)**

Q. Describe the following 3-D representation methods : CSG. *(Refer Section 6.2.5(B))* **(May 2014, 3 Marks)**

Q. Describe the following 3-D representation methods : B-REP *(Refer Section 6.2.5(C))* **(May 2014, 3 Marks)**

Q. Write a short note on Comparison of 3D object representation methods. *(Refer Section 6.2.5(E))* **(Dec. 2016, 10 Marks)**

Syllabus Topic : B-Spline Curve

Q. Write short note on: B-spline. *(Refer Section 6.3.1)* **(May 2017, 10 Marks)**

Q. Give the properties of B-spline curve. *(Refer Section 6.3.2)* **(Dec. 2016, 10 Marks)**

Syllabus Topic : Bezier Curve, Bezier Surface

Q. Write short note on : Bezier curves. *(Refer Section 6.4)* **(May 2014, May 2016, 10 Marks)**

Q. Explain what is meant by Bezier curve. Also explain how a Bezier surface can be generated from Bezier curve. *(Refer Section 6.4)* **(Dec.2014, 3 Marks, Dec.2015, 10 Marks)**

Q. State mathematical equations for Bezier Curves. *(Refer Section 6.4)* **(5 Marks)**

Q. Give mathematical equations for Bezier curve. 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 curve. *(Refer Section 6.4)* **(5 Marks)**

Q. State the properties of Bezier curves. *(Refer Section 6.4.1)* **(May 2013, 10 Marks)**

Q. Explain properties of Bezier Curve. *(Refer Section 6.4.1)* **(May 2014, May 2015, May 2016, Dec. 2016, 10 Marks)**

Q. State properties of Bezier curve and hence explain how a Bezier surface can be generated from Bezier curve. *(Refer Section 6.4.1)* **(Dec. 2014, 7 Marks)**

Q. Give properties for Bezier curve. 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 curve. *(Refer Section 6.4.1)* **(6 Marks)**

Syllabus Topic : Fractal-Geometry - Fractal Dimension, Koch Curve

Q. Write short note on : Fractals *(Refer Section 6.5)* **(May 2017, 5 Marks)**

Q. What are Fractals ? Give classification of Fractals. What is Fractal dimension. *(Refer Section 6.5)* **(8 Marks)**

Q. Define Koch curve ? How do you construct the Koch curve ? *(Refer Section 6.5.2)* **(May 2015, 10 Marks)**

Q. Write a short note on Construction of Koch curve. *(Refer Section 6.5.2)* **(Dec. 2016, 10 Marks)**

□□□

Module 6

CHAPTER 7

3D Geometric Transformations and Viewing

Section Nos.	Name of the Topic
7.1	Introduction
7.2	Basic 3D Transformations
7.3	Projections
7.4	3D Viewing Parameters
7.5	3D Clipping

Chapter 7 introduces three-dimensional co-ordinate system. This chapter covers 3D object representation methods, transformations of 3D objects, display of 3D objects on 2D screen and 3D clipping. This chapter has five sections.

7.1 **Introduction** introduces three dimensional Co-ordinate system and how to plot a 3D point on paper. It also explains 3D geometry.

7.2 **Basic 3D transformations** uses homogeneous co-ordinates to extend the 2D formulas to the 3D case. It also explains rotation about arbitrary axis and reflections.

7.3 **Projections** explain different ways to display 3D object on a 2D plane or screen. It explains parallel and perspective projections in detail.

7.4 **3D Viewing Parameters** explains different parameters which are used to display 3D object on view plane.

7.5 **3D Clipping** extends the concept of clipping to three dimensions. It explains how to display the portion of 3D object which is within a view volume.

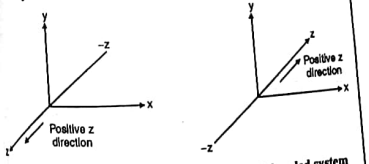
7.1 Introduction

Some graphics applications are two dimensional, such as charts, graphs certain maps and some artist's creations may be strictly two dimensional entities. But we live in a world of 3D. In many design applications we must deal with objects describing in 3D. If the architect would like to see how the structure will actually look, then a 3D model can allow him to view the structure from different view points. In this chapter we shall generalize our system to handle models of three dimensional objects.

Until now we have seen two dimensional objects and the operations which we can perform on them. We know that to represent a 2D object we need x and y co-ordinates. It means to draw a 2D object we need width and height of that object. But in nature objects are having three dimensions. So to represent such objects naturally we need three parameters. Out of these three parameters, two we know. They are width, i.e. x co-ordinate, and height, i.e. y co-ordinate. In 3D we need one additional parameter, which is called as depth and is represented by z co-ordinate. In 2D, only x and y-axis are there whereas in 3D, we are having 3 axes i.e. x, y and z. These three axes are arranged in such a way that they are normal to each other.

Now it is important to recognize that there are two different orientations for the z co-ordinate axis. A right handed system (RHS) has the z-axis pointing towards the viewer. If our right hand's four fingers (i.e. other than thumb) are curling from x to y direction, then the thumb of right hand indicates positive z direction.

A left handed system (LHS) has the z-axis pointing away from the viewer. Fig. 7.1.1 shows both right handed and left handed system. If our left hand's four fingers (i.e. other than thumb) are curling from x to y direction, then the thumb of left hand points positive z direction. As most of geometry uses right handed system so we adopt the same for our use.



(a) Right handed system (b) Left handed system

Fig. 7.1.1

Now we can represent any point by a set of three values (x, y, z). Here x represents horizontal distance, i.e. width, along x-axis; y represents vertical distance i.e. height, along y-axis; and z represents depth along z-axis. Fig. 7.1.2 shows how to plot a 3D point. Let us plot a point A (3, 2, 1).

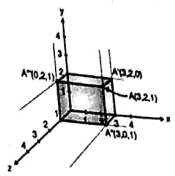


Fig. 7.1.2

To plot a 3D point we will follow some steps :

Step 1 : Plot a point A''(3, 0, 1).

To plot this point we have to draw one parallel line to z-axis at a distance of 3 units i.e. x = 3. Now draw parallel line to x-axis at a distance of 1 unit i.e. z = 1. The point where these two lines intersects will be a point having x = 3 and z = 1 but y = 0.

Step 2 : As we want a point as (3, 2, 1) we have to lift the point A'' (3, 0, 1) up by 2 units i.e. y = 2. So draw a parallel line to y-axis from A'' (3, 0, 1) point. Similarly, draw a point A'(3, 2, 0) and draw a parallel line from this point A'(3, 2, 0) to z-axis. Similarly, plot a point A''' (0, 2, 1) and then draw a parallel line form this point to x-axis. The point at which these three lines intersect is our required point A (3, 2, 1). Fig. 7.1.3 shows different 3D points.

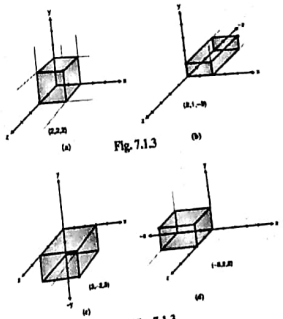


Fig. 7.1.3

Like this we can represent a line also. In 2D equation of a line is,

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}$$

i.e. As x changes, y also changes accordingly.

- Similarly in 3D as x changes both y and z will get changed accordingly. So to represent a line in 3D we need a pair of equations.

$$\frac{y-y_1}{x-x_1} = \frac{y_2-y_1}{x_2-x_1}$$

and

$$\frac{z-z_1}{x-x_1} = \frac{z_2-z_1}{x_2-x_1}$$

- It requires the (x_1, y_1, z_1) and (x_2, y_2, z_2) i.e. the co-ordinates of two points. We can represent the equation of a line in parametric form also.

$$x = x_1 + (x_2 - x_1)\,u$$

$$y = y_1 + (y_2 - y_1)\,u$$

$$z = z_1 + (z_2 - z_1)\,u$$

Syllabus Topic : 3D Transformations - Translation, Rotation , Scaling and Reflection. Composite transformations - Rotation about an arbitrary axis 3D transformation pipeline

7.2 Basic 3D Transformations

- Three dimensional transformations are similar to our normal 2D transformations. Rather in 3D we are extending 2D transformations by considering one additional parameter i.e. z co-ordinate. Like 2D transformations, we are having scaling, translation, rotation transformations for 3D also. In 3D, to explain all transformations, we are making use of homogeneous co-ordinate system.

7.2.1 Scaling

- Scaling a 3D point or object is much identical to scaling a 2D point or object. The scaling transformation relocates a point with relation to the origin. The formulas for 2D and 3D scaling are the same with addition of z co-ordinate. Basically scaling is nothing but, changing the size of an object. The normal scaling matrix in 3D will be,

$$\begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{vmatrix}$$

- where s_x, s_y and s_z represents the scaling factors for the three dimensions. 3D scaling matrix with homogeneous co-ordinate system will be,

$$\begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- In normal 2D, scaling matrix with homogenous co-ordinates is having size 3 × 3, whereas in 3D, it is of size 4 × 4. Here we are having additional factor s_z.

Scaling a point is done by multiplication of scaling matrix with a point.

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where,

$$x_{new} = x_{old} * s_x$$

$$y_{new} = y_{old} * s_y$$

$$z_{new} = z_{old} * s_z$$

- By properly selecting the values of s_x, s_y and s_z we can either shrink or elongate the object.

7.2.2 Translation

- Translation means shifting a point or moving whole object. In 2D, for translation we have used a 3 × 3 matrix.

i.e. $\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix}$ where t_x and t_y are translation factors in x and y direction

- Similarly in 3D we need three translation factors. So the transformation matrix in 3D will be of size 4 × 4 and it will be,

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{vmatrix}$$

- Here additional parameter t_z represents the translating factor for z direction. For translation we have to perform multiplication of a point or object with this matrix.

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where $x_{new} = x_{old} + t_x$

$y_{new} = y_{old} + t_y$

$z_{new} = z_{old} + t_z$

7.2.3 Rotation

➔ (May 2017)

Q. Write a short note on 3D rotation. **MU - May 2017, 10 Marks**

- Rotation in 3D is slightly different than the rotation in 2D. In 2D we are performing rotation in only one plane i.e. XY plane. In XY plane we can rotate a point or object in clockwise or in anticlockwise direction. But in case of 3D, as there are three axes we are having three different planes, i.e. XY, XZ and YZ planes. So in 3D we must specify to which axis and by what angle we want to rotate the object.

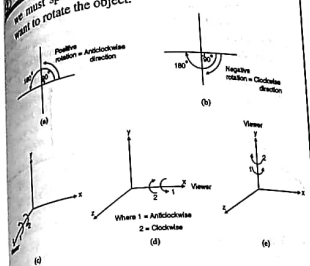


Fig. 7.2.1

Here in 3D also, we are making use of clockwise and anticlockwise directions at the time of rotation. The Fig. 7.2.1 helps to understand this. In 2D it is simple to detect clockwise and anticlockwise directions, but in 3D to detect clockwise and anticlockwise directions, we are looking from positive half of the axis towards the origin then we can decide clockwise and anticlockwise directions by normal convention. By convention, positive angle produces anticlockwise rotations, and negative angle produces clockwise rotations.

7.2.3(A) Rotation about z-axis

- We have already seen rotation of 2D object. For that we are having two different transformation matrices, one for clockwise rotation and another for anticlockwise rotation. If we want to rotate a 2D point, say (x, y) by angle θ in anticlockwise direction then the transformation matrix will be,

$$\begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix}$$

- And by multiplying this matrix with point (x, y) we will get,

$$|x_{old}, y_{old}| * \begin{vmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{vmatrix} = |x_{new}, y_{new}|$$

Where,

$$x_{new} = x_{old} * \cos\theta - y_{old} * \sin\theta$$

$$y_{new} = x_{old} * \sin\theta + y_{old} * \cos\theta$$

- We can utilize this concept to a 3D rotation about z-axis. When we are saying that we want to rotate a point, in anticlockwise direction, about z-axis it is very similar to performing a 2D rotation in anticlockwise direction as shown in Fig. 7.2.2.

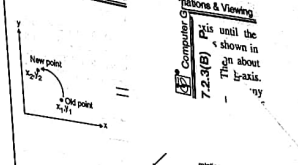


Fig. 7.2.2

- From the Fig. 7.2.2, we can say, if we perform rotation about z-axis (Clockwise/Anticlockwise), z co-ordinates remains unchanged and x and y co-ordinates changes as in case of 2D. So after performing rotation about z-axis, in anticlockwise direction, x_{new}, y_{new} and z_{new} will become,

$z_{new} = z_{old}$ i.e. unchanged

$x_{new} = x_{old} * \cos\theta - y_{old} * \sin\theta$ } Same as 2D

$y_{new} = x_{old} * \sin\theta + y_{old} * \cos\theta$

- Now let's derive a transformation matrix for rotation about z-axis, in anticlockwise direction. In homogeneous co-ordinates it will become,

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where, $x_{new} = x_{old} * \cos\theta - y_{old} * \sin\theta$

$y_{new} = x_{old} * \sin\theta + y_{old} * \cos\theta$

$z_{new} = z_{old}$

- Therefore the transformation matrix will be,

$$R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For anticlockwise)

The sign will change according to the angle of θ. If θ is negative, i.e. for clockwise, it will be

$$R_z = \begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For clockwise)

Where R_z stands for rotation about z-axis.

i.e. As x-rotation about x-axis

– Similarly rotation about x-axis can be obtained from the transformation matrix of rotation about z-axis with a cyclic permutation of the co-ordinate parameters x, y and z. It means we have to replace z co-ordinate by x co-ordinate; y by z and x by y co-ordinates.

$$x \rightarrow y \rightarrow z \qquad \ldots(7.2.1)$$

We can pictorially represent the Equation (7.2.1) as shown in Fig. 7.2.3.

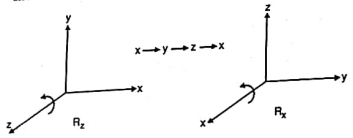


Fig. 7.2.3

– The equations of rotation about z-axis is,

$$x_{new} = x_{old} \cdot \cos\theta - y_{old} \cdot \sin\theta$$
$$y_{new} = x_{old} \cdot \sin\theta + y_{old} \cdot \cos\theta$$
$$z_{new} = z_{old}$$

– By making use of Equation (1) in above equations we get,

$$y_{new} = y_{old} \cdot \cos\theta - z_{old} \cdot \sin\theta$$
$$z_{new} = y_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$
$$x_{new} = x_{old}$$

– Here x-co-ordinate remains unchanged and y and z values gets changed.

– By using these equations we will get rotation about x-axis in anticlockwise direction. Now let's derive transformation matrix for rotation about x-axis.

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where, $x_{new} = x_{old}$

$$y_{new} = y_{old} \cdot \cos\theta - z_{old} \cdot \sin\theta$$
$$z_{new} = y_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$

– Therefore the transformation matrix will be,

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \text{ (For Anticlockwise)}$$

and $R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$ (For clockwise)

Where R_x stands for rotation about x-axis.

7.2.3(C) Rotation about y-axis

– To form a transformation matrix for rotation about y-axis, we are making use of transformation matrix of rotation about x-axis with a cyclic permutation of the co-ordinate parameters x, y and z. Refer Fig. 7.2.4.

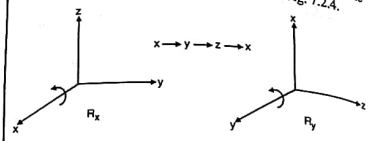


Fig. 7.2.4

– The equations of rotation about x-axis is,

$$x_{new} = x_{old}$$
$$y_{new} = y_{old} \cdot \cos\theta - z_{old} \cdot \sin\theta$$
$$z_{new} = y_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$

– By making use of cyclic permutations of x, y and z in above equations we get,

$$y_{new} = y_{old}$$
$$z_{new} = z_{old} \cdot \cos\theta - x_{old} \cdot \sin\theta$$
$$x_{new} = z_{old} \cdot \sin\theta + x_{old} \cdot \cos\theta$$

– Here y co-ordinate remains unchanged and x and z values gets changed. By using these equations we will get rotation about y-axis, in anticlockwise direction. Now let's derive transformation matrix for rotation about y-axis.

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where,

$$x_{new} = x_{old} \cdot \cos\theta + z_{old} \cdot \sin\theta$$
$$y_{new} = y_{old}$$
$$z_{new} = -x_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$

Therefore the transformation matrix will be,

$$R_y = \begin{vmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \text{ (For Anticlockwise)}$$

And $R_y = \begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$ (For clockwise)

– Remember that sign of transformation matrix for rotation about y-axis, in anticlockwise direction, is opposite to that of rotation about x-axis and z-axis, in anticlockwise direction.

7.2.3(D) Rotation about an Arbitrary Axis

Q. Derive all the necessary matrices required to perform 3D Rotation about arbitrary axis. (10 Marks)

– Until now we have seen how to rotate a point or object with respect to co-ordinate axes, such as x, y and z. But suppose we want to rotate a point or object at angle θ about an arbitrary line which does not coincide with any of the co-ordinate axes, then we have to set up a composite transformation involving combinations of translations and the co-ordinate axes rotations.

– We obtain the required composite matrix, for rotation about an arbitrary axis, by first setting up the transformation sequence that moves the selected rotation axis onto one of the co-ordinate axes. Then we set up the rotation matrix about that co-ordinate axis for the specified rotation angle. The last step is to obtain the inverse transformation sequence that returns the rotation axis to its original position.

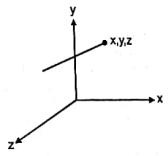


Fig. 7.2.5

– If we want to rotate a point (x, y, z), by an angle θ, which is in space and lying on an arbitrary axis as shown in Fig. 7.2.5. Then we have to follow some steps in sequence.

Step 1 : Translate the arbitrary axis so that it will pass through the origin. When we are translating axis, a point also gets translated. See Fig. 7.2.6

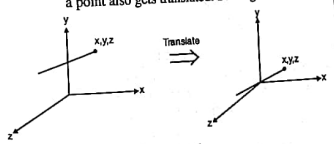


Fig. 7.2.6

Step 2 : Perform rotation about x-axis until the arbitrary axis completely lies in XZ plane. See Fig. 7.2.7.

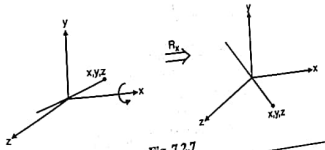


Fig. 7.2.7

Step 3 : Now perform rotation about y-axis until the arbitrary axis coincides with z-axis as shown in Fig. 7.2.8. We can also perform rotation about y-axis until arbitrary axis coincides with x-axis. Here we are matching the arbitrary axis to any one co-ordinate axis, so that we can make use of standard rotation matrices.

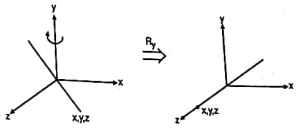


Fig. 7.2.8

Step 4 : Once the arbitrary axis coincides with any one co-ordinate axis then the rotation of arbitrary axis will be same as that of matched co-ordinate axis. Here we are performing rotation about z-axis as we have matched arbitrary axis to z-axis. See Fig. 7.2.9

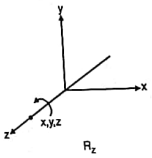


Fig. 7.2.9

Step 5 : After performing rotation about z-axis (i.e. rotation about arbitrary axis) we have to perform reverse transformations. Here we should not perform reverse rotation about z-axis, because it is rotation about arbitrary axis. So we have to perform reverse rotation about y-axis. i.e. if earlier we rotated y-axis, in anticlockwise direction, now we have to rotate it in clockwise direction. See Fig. 7.2.10.

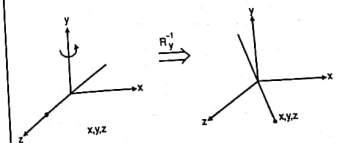


Fig. 7.2.10

Step 6 : Again we have to perform reverse rotation about x-axis. See Fig. 7.2.11.

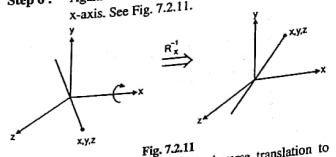


Fig. 7.2.11

Step 7 : Lastly we have to apply inverse translation to bring the arbitrary axis back to its original position. See Fig. 7.2.12

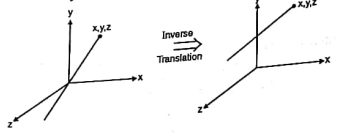


Fig. 7.2.12

- So we need above mentioned seven steps to perform rotation about arbitrary axis.
- Now we can determine the transformation matrix for rotation about arbitrary axis. Assume that arbitrary axis is drawn by using two points (x_1, y_1, z_1) and (x_2, y_2, z_2). Let us say point on this arbitrary axis is (P, Q, R). See Fig. 7.2.13.

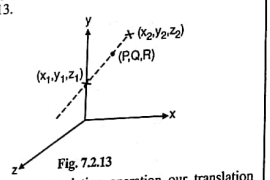


Fig. 7.2.13

- Step 1 indicates translation operation our translation matrix will be,

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -x_1 & -y_1 & -z_1 & 1 \end{vmatrix}$$

- After performing this translation, point (x_1, y_1, z_1) will be shifted to origin. See Fig. 7.2.14.

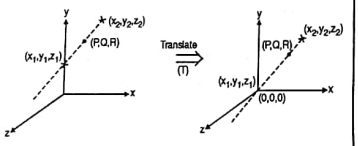


Fig. 7.2.14

- Immediately we will find inverse translation also. Its inverse translation matrix will be,

$$T^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ x_1 & y_1 & z_1 & 1 \end{vmatrix}$$

- Now the next step is rotation about x-axis, so that the arbitrary axis will lie in XZ plane. To form this transformation matrix refer Fig. 7.2.15.

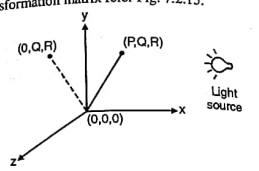


Fig. 7.2.15

- Here we are considering a line segment (0, 0, 0) to (P, Q, R) which is a part of rotation axis. When we are focusing some ray's in x direction, this line segments shadow will lie on YZ plane and the length of shadow will be (0, 0, 0) to (0, Q, R). See Fig. 7.2.15. Now if we perform rotation about x-axis until arbitrary axis is in XZ plane, the shadow of line (0, 0, 0) and (0, Q, R) will coincide with z-axis as shown in Fig. 7.2.16.

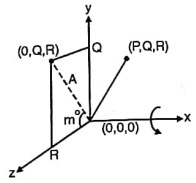


Fig. 7.2.16

- Now we have to decide by how much angle we have to rotate the shadow, so that it will match to the z-axis. Let's call that angle as 'M', and the length of shadow as 'A'.

$\therefore$ length of shadow $(A) = \sqrt{(Q)^2 + (R)^2}$

And $\sin M = \dfrac{\text{Opposite side}}{\text{hypotenuse}}$

$\sin M = \dfrac{Q}{A}$

$\cos M = \dfrac{\text{Adjacent side}}{\text{hypotenuse}}$

$\cos M = \dfrac{R}{A}$

- The matrix for rotation about x-axis, in anticlockwise direction is,

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- By replacing θ with angle M we will get rotation about x-axis as,

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos M & \sin M & 0 \\ 0 & -\sin M & \cos M & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Now placing values of Cos M and Sin M in above matrix we get,

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & R/A & Q/A & 0 \\ 0 & -Q/A & R/A & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Immediately we will find reverse rotation of this. If earlier we have performed anticlockwise rotation, now we have to perform clockwise rotation. Magnitude remains unchanged. The matrix for reverse rotation about x-axis will be,

$$R_x^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & R/A & -Q/A & 0 \\ 0 & Q/A & R/A & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- After performing rotation about x-axis, the shadow will coincide with z-axis and the actual line will lie in XZ plane as shown in Fig. 7.2.17.

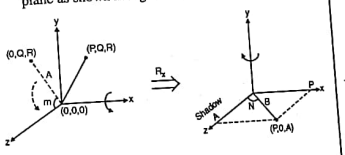


Fig. 7.2.17

- When we are performing rotation about x-axis, the values of x co-ordinates remains unchanged. But values of y co-ordinates becomes zero, because now our actual point lies on XZ plane and the value of z co-ordinate will be replaced by A instead of R. So now the actual points co-ordinates will be (P, 0, A). Now let's find the length of actual point from origin.

$$B = \sqrt{(P)^2 + (A)^2}$$

But $A^2 = Q^2 + R^2$

$\therefore$ $B = \sqrt{(P)^2 + (Q)^2 + (R)^2}$

- Now we have to perform rotation about y-axis in clockwise direction by angle N, so that the line of arbitrary axis aligns to z-axis.

$\therefore$ $\sin N = \dfrac{\text{Opposite side}}{\text{hypotenuse}}$

$\therefore$ $\sin N = \dfrac{P}{B}$

and $\cos N = \dfrac{\text{Adjacent side}}{\text{hypotenuse}}$

$\therefore$ $\cos N = \dfrac{A}{B}$

- We know that matrix for rotation about y-axis in clockwise direction is,

$$R_y = \begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Replacing θ by angle N, we will get,

$$R_y = \begin{vmatrix} \cos N & 0 & \sin N & 0 \\ 0 & 1 & 0 & 0 \\ -\sin N & 0 & \cos N & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Replacing the values of cos N and sin N in above matrix we will get,

$$R_y = \begin{vmatrix} A/B & 0 & P/B & 0 \\ 0 & 1 & 0 & 0 \\ -P/B & 0 & A/B & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Immediately we will find reverse rotation of this it will be,

$$R_y^{-1} = \begin{vmatrix} A/B & 0 & -P/B & 0 \\ 0 & 1 & 0 & 0 \\ P/B & 0 & A/B & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- Here at this stage we have matched an arbitrary axis to z-axis. Now we can use our normal rotation about z-axis which will be nothing but rotation about arbitrary axis by some angle θ.

$$\therefore \quad R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

- So the final transformation matrix, which will be a **3D transformation pipeline** for rotation about arbitrary axis, will be the product of above transformation.

$$\therefore \quad R_A = T \cdot R_x \cdot R_y \cdot R_z \cdot R_y^{-1} \cdot R_x^{-1} \cdot T^{-1}$$

7.2.4 Other Transformations

- In addition to normal scaling, translation and rotation there are other transformations also, which we can perform on 3D. There are two types of reflection transformations.

1. Reflection relative to co-ordinate axes
2. Reflection relative to planes.

– In general three dimensional reflection matrices are set up similar to that of two dimensional reflection matrices.

Reflection at y-axis

– In case of two dimension, when we are considering reflection at y-axis, we are keeping the values of x and y co-ordinates as it is and changing only the sign of x co-ordinate. It's transformation matrix will be,

$$\begin{vmatrix} -1 & 0 \\ 0 & 1 \end{vmatrix}$$

– But in case of 3D, when we are taking reflection about y-axis, we have to keep the magnitudes of x, y and z co-ordinates as it is. We have to change only signs of x and z co-ordinates. It's transformation matrix will be,

$$\begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{vmatrix}$$

– Let's take an example, suppose we are having a point (2, 1, 2) and we want to take its reflection about y-axis. Then

$$|2,\ 1,\ 2| * \begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{vmatrix} = |-2,\ 1,\ -2|$$

– ∴ new point will become as (–2, 1, –2). Fig. 7.2.18 shows reflection about y-axis.

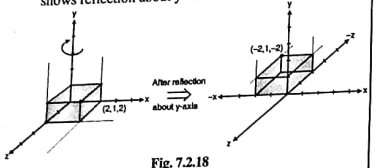


Fig. 7.2.18

– Reflection relative to given axis is equivalent to 180° rotation about that axis. Like this we can achieve reflection about x-axis and reflection about z-axis also. The following table gives transformation matrix and the co-ordinates of point (2, 1, 2) after r eflection.

Reflection About	Transformation matrix	Reflection of point (2, 1, 2)
y-axis	$\begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{vmatrix}$	= (–2, 1, –2)
x-axis	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{vmatrix}$	= (2, –1, –2)
z-axis	$\begin{vmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	= (–2, –1, 2)

– Similarly we can find reflection about planes also. In 3D we are having three different planes i.e. XY, YZ and XZ plane. Basically reflection means 180° rotation.

– The transformation matrix for reflection about XZ plane is

$$\begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

– Let us take an example. Suppose we are having a point (3, 3, 3), then after reflection about XZ plane the point will become $|3,\ 3,\ 3| * \begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = |3,\ -3,\ 3|$

Fig. 7.2.19 shows reflection about XZ plane

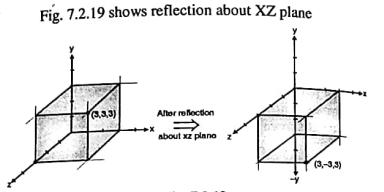


Fig. 7.2.19

– Like this we can achieve reflection about YZ and XY plane also. The following table gives transformation matrix and the co-ordinates of point (3, 3, 3) after reflection.

Reflection about plane	Transformation matrix	Reflection of point (3, 3, 3)
XZ plane	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	= (3, –3, 3)
XY plane	$\begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{vmatrix}$	= (3, 3, –3)
YZ plane	$\begin{vmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$	= (–3, 3, 3)

Program for 3D Transformations

```
/*************************************************
   Program Title : 3D Transformations.
   A C++ program to implement different 3D
transformations such as
            * Scaling
            * Translation
            * Rotation
*************************************************/
#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<graphics.h>
#include<alloc.h>
#include<stdlib.h>
#include<math.h>
#include<dos.h>

int gd=0, gm;

class axes
{
    private:   // Initialiasing data members for axes
    int xmx,ymx,yst,xst,zmx,zst;
    float xsc,ysc,xstep,ystep,zsc,zstep;
    public:
    axes();
    void display();
    int actx(int tmx,int tmy,int tmz)
    {
    return 320+((xsc*tmx)-(zsc*tmz/2))+tmy-tmy;
    }
    int acty(int tmx,int tmy,int tmz)
    {
    return 255-((ysc*tmy)-(zsc*tmz/2))+tmx-tmx;
    }
};
axes::axes()
{
    xmx=280;
    ymx=200;
    zmx=400;
    xsc=280.0/xmx;
    xstep=10.0*xmx/280.0;
    xst=ceil(xstep);

    ysc=200.0/ymx;
    ystep=10.0*ymx/200.0;
    yst=ceil(ystep);

    zsc=400.0/zmx;
    zstep=10.0*zmx/400.0;
    zst=ceil(zstep);
}
/*============================
   Member Name :  display()
   Purpose     :  To display the X,Y & Z-axis
-----------------------------*/

void axes::display()
{
    setcolor(CYAN);
    line(25,255,615,255);
    line(320,25,320,455);
    line(actx(0,0,zmx),acty(0,0,zmx),
    actx(0,0,-zmx),acty(0,0,-zmx));
    outtextxy(325,255,"0");
    settextstyle(3,0,1);
    outtextxy(620,240,"X");
    outtextxy(323,20,"Y");
    outtextxy(actx(0,0,zmx)+3,acty(0,0,zmx)+3,"Z");
    settextstyle(0,0,0);
}
class polygon
{
    private:  // Initialiasing data members
    float cube[8][4];
    public:  // Declaring all transformation functions
    void accept();
    void display(int);
    void scaling();
    void scale(float,float,float);
    void translation();
    void trans(int,int,int);
    void rotate();
    void x_rotat(float);
    void y_rotat(float);
    void z_rotat(float);
    int plotx(int i)
    {
        return cl.actx(cube[i][0],cube[i][1],cube[i][2]);
    }
    int ploty(int i)
```

```
    {
        return cl.acty(cube[i][0],cube[i][1],cube[i][2]);
    }
    polygon operator * (float [4][4]);
}p1;
/*============================
Member Name :  display(int)
Purpose     :  To display a 3D cube with specified color
----------------------------*/
void polygon::display(int color)
{
    setcolor(color);
    line(plotx(0),ploty(0),plotx(1),ploty(1));
    line(plotx(1),ploty(1),plotx(3),ploty(3));
    line(plotx(3),ploty(3),plotx(2),ploty(2));
    line(plotx(2),ploty(2),plotx(0),ploty(0));
    line(plotx(4),ploty(4),plotx(5),ploty(5));
    line(plotx(5),ploty(5),plotx(7),ploty(7));
    line(plotx(7),ploty(7),plotx(6),ploty(6));
    line(plotx(6),ploty(6),plotx(4),ploty(4));
    line(plotx(0),ploty(0),plotx(4),ploty(4));
    line(plotx(1),ploty(1),plotx(5),ploty(5));
    line(plotx(2),ploty(2),plotx(6),ploty(6));
    line(plotx(3),ploty(3),plotx(7),ploty(7));
}
/*============================
    Member Name  : accept()
    Purpose      : To accept the 3D only one corner
                   point of an object from the user
                   and generate all point of object to
                   display on the screen .
----------------------------*/
void  polygon::accept()
{
    int xc,yc,zc;
    cout<<"\n\t\t 3-D TRANSFORMATIONS";
cout<<"\n\tenter the co-ordinates of the corner of the
cube...";
    cout<<"\n\n\tx :     ";
    cin>>xc;
    cout<<"\n\ty : ";
    cin>>yc;
    cout<<"\n\tz : ";
    cin>>zc;
    int sd;
    cout<<"\n\tEnter the length of each side : ";
    cin>>sd;
    for(int i=0;i<8;i++)
      {
          cube[i][3] =1;
      }
    cube[0][0] = xc;
    cube[0][1] = yc;
    cube[0][2] = zc;

    cube[1][0] = xc;
    cube[1][1] = yc;
    cube[1][2] = zc+sd;

    cube[2][0] = xc;
    cube[2][1] = yc+sd;
    cube[2][2] = zc;
    cube[3][0] = xc;
    cube[3][1] = yc+sd;
    cube[3][2] = zc+sd;

    cube[4][0] = xc+sd;
    cube[4][1] = yc;
    cube[4][2] = zc;

    cube[5][0] = xc+sd;
    cube[5][1] = yc;
    cube[5][2] = zc+sd;
    cube[6][0] = xc+sd;
    cube[6][1] = yc+sd;
    cube[6][2] = zc;
    cube[7][0] = xc+sd;
    cube[7][1] = yc+sd;
    cube[7][2] = zc+sd;
}
/*============================
    Member Name  : operator* (float)
    Purpose      : To multiply two matrices by
                   using operator overloading
                   concept
----------------------------*/
polygon polygon::operator* (float b[4][4])
{
    polygon c;
    for(int i=0;i<8;i++)
        for(int j=0;j<4;j++)
        {
            c.cube[i][j]=0;
            for(int k=0;k<4;k++)
            c.cube[i][j]+=cube[i][k]*b[k][j];
            delay(1);
        }
        return c;
}
/*============================
    Member Name  : scaling().
```



```
    Purpose  : To accept the scaling factors to
               perform scaling transformation.
----------------------------*/
void polygon::scaling()
{
    float sx,sy,sz;
    gotoxy(1,1);
    cout<<"  "
          "
          ";
    gotoxy(1,1);
    cout<<" X scaling factor  :";
    cin>>sx;

    gotoxy(1,1);
    cout<<"  "
          "
          ";
    gotoxy(1,1);
    cout<<" Y scaling factor  :";
    cin>>sy;
    gotoxy(1,1);
    cout<<"          "
          "
          ";
    gotoxy(1,1);
    cout<<" Z scaling factor  :";
    cin>>sz;
    scale(sx,sy,sz);
    getch();
    gotoxy(1,3);
    cout<<" ";
    gotoxy(1,4);
    cout<<" ";
    gotoxy(1,5);
    cout<<" ";
}

/*============================
    Member Name  :    scale()
    Purpose      :    To create scaling
                      matrix and multiply it
                      with object to
                      perform scaling      transformation.
----------------------------*/
void polygon::scale(float sx,float sy, float sz)
{
    float sclmt[4][4];
    sclmt[0][0]=sx;
    sclmt[0][1]=0;
    sclmt[0][2]=0;
    sclmt[0][3]=0;

    sclmt[1][0]=0;
    sclmt[1][1]=sy;
    sclmt[1][2]=0;
    sclmt[1][3]=0;

    sclmt[2][0]=0;
    sclmt[2][1]=0;
    sclmt[2][2]=sz;
    sclmt[2][3]=0;

    sclmt[3][0]=0;
    sclmt[3][1]=0;
    sclmt[3][2]=0;
    sclmt[3][3]=1;
    cleardevice();
    gotoxy(1,1);
    cout<<"scaling results....";
    cl.display();

    p1=p1*sclmt;
    display(2);
    gotoxy(1,3);
    cout<<"sx =  "<<sx;
    gotoxy(1,4);
    cout<<"sy =  "<<sy;
    gotoxy(1,5);
    cout<<"sz =  "<<sz;
}
/*============================
    Member Name  :    translation()
    Purpose      :    To accept the translation factors
                      to perform translation.
----------------------------*/
void polygon::translation()
{
    float tx,ty,tz;
    gotoxy(1,1);
    cout<<"      "
          "
          ";
    gotoxy(1,1);
    cout<<" X Translation factor    :";
    cin>>tx;

    gotoxy(1,1);
    cout<<"      "
          "
          ";
```

```
    gotoxy(1,1);
    cout<<" Y translation factor      :";
    cin>>ty;

    gotoxy(1,1);
    cout<<"                           "
                                      "
                                      ";

    gotoxy(1,1);
    cout<<" Z Translation factor      :";
    cin>>tz;

    trans(tx,ty,tz);
    getch();
    gotoxy(1,3);
    cout<<"                           ";
    gotoxy(1,4);
    cout<<"                           ";
    gotoxy(1,5);
    cout<<"                           ";
}
/*============================
    Member Name  :   trans(int,int,int)
    Purpose      :   To create translation matrix and
                     multiply it  with object to perform
                     translation.
----------------------------*/

void polygon::trans(int tx,int ty, int tz)
{
    float tr[4][4];

    tr[0][0]=1;
    tr[0][1]=0;
    tr[0][2]=0;
    tr[0][3]=0;

    tr[1][0]=0;
    tr[1][1]=1;
    tr[1][2]=0;
    tr[1][3]=0;

    tr[2][0]=0;
    tr[2][1]=0;
    tr[2][2]=1;
    tr[2][3]=0;

    tr[3][0]=tx;
    tr[3][1]=ty;
    tr[3][2]=tz;
    tr[3][3]=1;
    cleardevice();
    gotoxy(1,1);
    cout<<"Translation results....";
    cl.display();
    p1=p1*tr;
    display(4);
    gotoxy(1,3);
    cout<<"tx =  "<<tx;
    gotoxy(1,4);
    cout<<"ty =  "<<ty;
    gotoxy(1,5);
    cout<<"tz =  "<<tz;
}
/*============================
    Member Name  :   rotate()
    Purpose      :   To select rotation about a either
                     X,Y or Z-axis with specified angle.
----------------------------*/

void polygon::rotate()
{
    int ch;
    gotoxy(1,1);
    cout<<"                           "
                                      "
                                      ";

    gotoxy(1,1);
    cout<<" 1=> X    2=> Y    3=> Z    :";
    cin>>ch;
    float angle;
    gotoxy(1,1);
    cout<<"                           "
                                      "
                                      ";

    gotoxy(1,1);
    cout<<" Rotation angle : ";
    cin>>angle;
    angle*=M_PI/180;
    switch(ch)
    {
        case 1 :    x_rotat(angle);
                                      break;

        case 2 :    y_rotat(angle);
                                      break;

        case 3 :    z_rotat(angle);
                                      break;

    }
}
/*============================
    Member Name  :   x_rotat(float)
    Purpose      :   To perform rotation about X-axis
                     with specified angle.
```



```
----------------------------*/
void polygon::x_rotat(float angle)
{
    float rt[4][4];
    rt[0][0]=1;
    rt[0][1]=0;
    rt[0][2]=0;
    rt[0][3]=0;

    rt[1][0]=0;
    rt[1][1]=cos(angle);
    rt[1][2]=-sin(angle);
    rt[1][3]=0;

    rt[2][0]=0;
    rt[2][1]=sin(angle);
    rt[2][2]=cos(angle);
    rt[2][3]=0;

    rt[3][0]=0;
    rt[3][1]=0;
    rt[3][2]=0;
    rt[3][3]=1;

    cleardevice();
    p1=p1*rt;
    cleardevice();
    gotoxy(1,1);
    cout<<"Rotating about X-axis....";
    cl.display();

    display(2);
    delay(5000);
}
/*============================
    Member Name  :   y_rotat(float)
    Purpose      :   To perform rotation about
                     Y-axis with specified angle.
----------------------------*/
void polygon::y_rotat(float angle)
{
    float rt[4][4];
    rt[0][0]=cos(angle);
    rt[0][1]=0;
    rt[0][2]=sin(angle);
    rt[0][3]=0;
    rt[1][0]=0;
    rt[1][1]=1;
    rt[1][2]=0;
    rt[1][3]=0;
    rt[2][0]=-sin(angle);
    rt[2][1]=0;
    rt[2][2]=cos(angle);
    rt[2][3]=0;

    rt[3][0]=0;
    rt[3][1]=0;
    rt[3][2]=0;
    rt[3][3]=1;

    cleardevice();
    gotoxy(1,1);
    cout<<"Rotating about Y-axis....";
    cl.display();
    p1=p1*rt;
    display(2);
    delay(5000);
}
/*============================
    Member Name  :   z_rotat(float)
    Purpose      :   To perform rotation about Z-axis
                     with specified angle.
----------------------------*/
void polygon::z_rotat(float angle)
{
    float rt[4][4];
    rt[0][0]=cos(angle);
    rt[0][1]=-sin(angle);
    rt[0][2]=0;
    rt[0][3]=0;

    rt[1][0]=sin(angle);
    rt[1][1]=cos(angle);
    rt[1][2]=0;
    rt[1][3]=0;

    rt[2][0]=0;
    rt[2][1]=0;
    rt[2][2]=1;
    rt[2][3]=0;

    rt[3][0]=0;
    rt[3][1]=0;
    rt[3][2]=0;
    rt[3][3]=1;
    cleardevice();
    gotoxy(1,1);
    cout<<"Rotating about Z-axis....";
    cl.display();
```

```
        p1=p1*rt;
    display(2);
    delay(5000);
}

main()
{
    initgraph(&gd,&gm,"c:\\tcplus\\bgi");
    p1.accept();
    cleardevice();
    cl.display();
    p1.display(2);
    int choice;
    while(1)
    {
        gotoxy(1,1);          cout<<"          "
                                  "          ";
        gotoxy(1,1);
cout<<"1.scaling  2.translation  3. Rotation  4.exit : ";
        cin>>choice;
        switch(choice)
        {
            case 1 : p1.scaling();
                                  break;
            case 2 : p1.translation();
                                  break;
            case 3 : p1.rotate();
                                  break;
            case 4 : getch();
                                  closegraph();
                                  exit (0);
        }
    }
}
/*======== End of program ==========*/
```

Example 7.2.1

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 the following transformations on the above cube :

(i) Translation ($t_x = 2,\ t_y = 4,\ t_z = 0$)

(ii) Scaling ($s_x = 0.5,\ s_y = 1,\ s_z = 1$)

(iii) Reflection about planes.

Solution :

Given : A cube having 8 vertices.

Let's represent the cube in matrix form and draw the initial state.

$$\begin{bmatrix} 0&0&0&1\\ 2&0&0&1\\ 2&2&0&1\\ 0&2&0&1\\ 0&0&2&1\\ 0&2&2&1\\ 2&0&2&1\\ 2&2&2&1 \end{bmatrix}$$

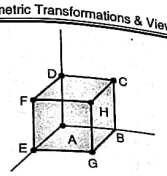


Fig. P. 7.2.1

Now let's perform different transformations on it.

(i) Translation ($t_x = 2,\ t_y = 4,\ t_z = 0$)

Here we have to represent a cube matrix in homogeneous co-ordinate form.

$$\begin{bmatrix} 0&0&0&1\\ 2&0&0&1\\ 2&2&0&1\\ 0&2&0&1\\ 0&0&2&1\\ 0&2&2&1\\ 2&0&2&1\\ 2&2&2&1 \end{bmatrix} \times \begin{bmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&0\\ 2&4&0&1 \end{bmatrix} = \begin{bmatrix} 2&4&0&1\\ 4&4&0&1\\ 4&6&0&1\\ 2&6&0&1\\ 2&4&2&1\\ 2&6&2&1\\ 4&4&2&1\\ 4&6&2&1 \end{bmatrix}$$

Translation Matrix

(ii) Scaling ($s_x = 0.5,\ s_y = 1,\ s_z = 1$)

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

Scaling Matrix

(iii) Reflection about planes

For xz plane

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

For xy plane Reflection about xz plane

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

For yz plane Reflection about xy plane

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

Reflection about yz plane

Example 7.2.2

A 3D cube of dimensions (length, breadth, and height) 2 units each is placed in a 3D anti-clockwise axis system such that one of its vertex "A" is at the origin (i.e. (0, 0, 0)) and vertex "F" in 3D space. Apply necessary transformations such that vertex F becomes the origin. Give complete mathematical formulation. Draw initial and final state of the cube.

Solution :

Given :

A cube having length, breadth and height as 2 units in such a way as point A is at origin and point F is in space. Let's draw the cube and represent it in matrix form as,

$$\begin{bmatrix} 0&0&0&1\\ 2&0&0&1\\ 2&2&0&1\\ 0&2&0&1\\ 0&2&2&1\\ 0&0&2&1\\ 2&0&2&1\\ 2&2&2&1 \end{bmatrix}$$

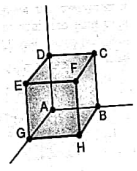


Fig. P. 7.2.2

We need to shift vertex 'F' which is in space to origin so we have to apply translation with $t_x = -2,\ t_y = -2$ and $t_z = -2$

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

Translation Matrix

So the final cube will become as

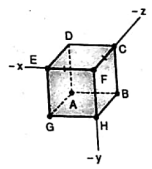


Fig. P. 7.2.2(a)

Syllabus Topic : Projections - Parallel , Perspective (Matrix Representation)

7.3 Projections

- We know that the objects available in nature are described in world-coordinate system. But all graphics display media are two dimensional, such as screen of monitor, printout of printer etc. when we want to draw a 3D object on a monitor, we have to convert the world coordinates into screen coordinates. For this we have to project a 3D object on a 2D plane, here 2D plane will be screen of monitor. Projection is nothing but a shadow of the object.
- Ideally an object is projected by projecting each of its points. As there are infinite points in an object, we cannot produce projections of all these points. Instead of that we will take projections of only corner points of an object on 2D plane and then we will join these projected points by straight line in 2D plane.

There are two basic projection methods :

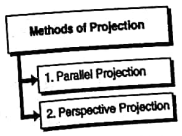


Fig. C7.1 : Methods of projection

7.3.1 Parallel Projection

(May 2013, May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, Dec 2016, May 2017)

Q. Explain Parallel projection. **MU - May 2013, May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, Dec. 2016, 5 Marks.**

Q. Differentiate between parallel and perspective projections. Explain with the help of examples. **MU - May 2017, 10 Marks**

- A parallel projection is formed by extending parallel lines from each vertex of the object until they intersect the plane of the screen. The point of intersection is the projection of the vertex. Then we connect the projected vertices by line segments which correspond to connections on the original object. See Fig. 7.3.1.

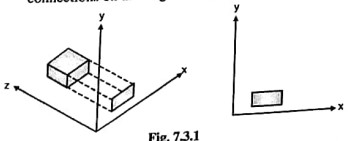


Fig. 7.3.1

- Here we are taking projection of each vertex of the object till it intersects to the plane of screen. The plane on which we are taking projection is called as view plane. If we want to represent a 3D object on 2D plane, the most simple way is to discard the z co-ordinate. We are discarding the z co-ordinates only when the screen or viewing surface is parallel to the XY plane and the lines of projection are parallel to the z-axis.
- A parallel projection preserves relative properties of objects, in x and y direction. Accurate views of the various sides of an object are obtained with a parallel projection, but this does not give us a realistic representation of the appearance of a 3D object.
- Let us take an example. Suppose there is a 3D cube in space and we want to draw it on screen i.e. on XY plane. See Fig. 7.3.2.
- We obtain the parallel projection of a point (x_1, y_1, z_1) by drawing a line through this point with a certain direction (x_p, y_p, z_p) in space. This line is called as projection vector. The point where this line intersects the plane $z = 0$ is the parallel projection of the given point; it has the coordinate $(x_2, y_2, 0)$

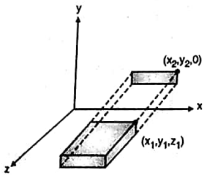


Fig. 7.3.2

- We will write the equation for a line passing through the point (x_1, y_1, z_1) and the direction of projection.

$$x = x_1 + x_p \cdot u$$
$$y = y_1 + y_p \cdot u$$
$$z = z_1 + z_p \cdot u$$

where $u = 0$ to 1

$$x_p = (x_2 - x_1)$$
$$y_p = (y_2 - y_1)$$
$$z_p = (z_2 - z_1)$$

- Now we have to find at which point this line intersects the XY plane. It means what will be x and y values at which z becomes zero.

If we place $z = 0$ in above equation we will get

$$z = z_1 + z_p \cdot u$$

put $z = 0$

$$0 = z_1 + z_p \cdot u$$

$$\therefore u = \frac{-z_1}{z_p}$$

- Substituting the value of 'u' in remaining two equations,

$$\therefore x = x_1 + \left(\frac{-z_1}{z_p}\right) \cdot u$$

$$y = y_1 + \left(\frac{-z_1}{z_p}\right) \cdot u$$

$$z = 0$$

- When the projection lines are normal (Perpendicular) to the projection plane, the projection is called orthographic projection.
- We obtain this projection by setting z co-ordinate to zero.
- This projection is a good approximation of the actual projections that the human eye makes in the real world.
- Orthographic projections do not change the length of line segments which are parallel to projection plane. Other lines are projected with reduced length.
- Generally we are using orthographic projections in engineering drawings to produce front, side and top view of an object. Fig. 7.3.3 shows different view of an object.

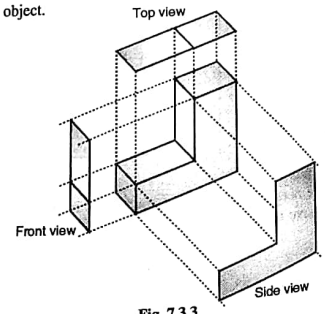


Fig. 7.3.3

- Here by using orthographic projections we can draw any sides view of an object. But we can also form orthographic projections in such a way that more than one face of an object can be displayed. Such views are called as **axonometric orthographic projections.**

Types of axonometric orthographic projections

- There are three types of axonometric orthographic projections. They are :

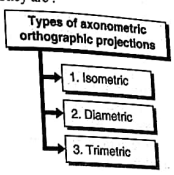


Fig. C7.2 : Types of orthographic projections

1. Isometric projection

- The most commonly used axonometric projections are isometric projections. In isometric projections, the projections are aligned in such a way that all the edges will appear shortened by same distance.

See Fig. 7.3.4 (a) and (b).

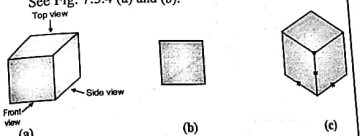


Fig. 7.3.4

- If we consider an object as cube then side view, front view and top view will look like square i.e. we are able to see only one face at a time. But by using isometric orthographic projection we can see more than one face. The object shown in Fig. 7.3.4(a) will look like as shown in Fig. 7.3.4(c).
- Here all three principal axes are shortened equally so that relative proportions are maintained.

2. Diametric Projection

- In diametric projections, the direction of projection is in such a way that the edges parallel to only two principle axes are equally shortened. In diametric projections, a cube will appear as shown in Fig. 7.3.5.

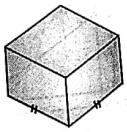


Fig. 7.3.5

3. Trimetric Projection

- In trimetric projections, the direction of projection is in such a way that none of the three edges are equally shortened. Fig. 7.3.6 shows appearance of cube in trimetric projection.

Fig. 7.3.6

- Until now we have seen orthographic projections, where project lines are normal to view plane. But if the projection lines are not normal to view plane, then it is called as **oblique** projections Fig. 7.3.7 shows difference between orthographic and oblique projections.

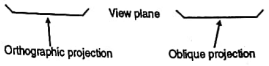


Fig. 7.3.7

- An oblique projection is obtained by projecting points along parallel lines that are not perpendicular to the view plane such a projection appears in nature when sunlight casts shadows on the ground. Types of oblique projections we have two types of oblique projections.

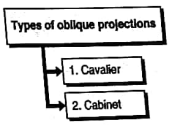


Fig. C7.3 : Types of oblique projections

1. Cavalier Projection

- Cavalier projection is a special case of oblique projection. This occurs when the projection vector forms an angle of 45° with the z-axis. The lines which are parallel to the z-axis are projected with no change in length.
- Consider an object whose edges are parallel to axes and front face of the object is parallel to view plane XY. See Fig. 7.3.8.

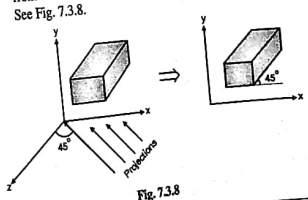


Fig. 7.3.8

- As the projections are not perpendicular to view plane, we will not get only one face of the object. As the angle of projection is 45°, we will get z-axis values as unchanged rather x and y-axis values gets reduced. So the object will look like an elongated object.

→ 2. Cabinet Projections

- In cabinet projections we are having only half the actual z distance along the projected axis. It uses a projection vector that forms an angle of approximately 26.6° with the z-axis. See Fig. 7.3.9.

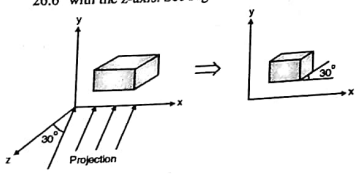


Fig. 7.3.9

- As the projections are making angle of 26.6° (= 30°), the z-axis values get half of its original values. Cabinet projections appear more realistic than cavalier projections because of this reduction in the length of edges which are perpendicular to view plane.

→ 7.3.2 Perspective Projection

→ (May 2013, May 2014, Dec. 2014, May 2015, Dec. 2015, May 2016, Dec 2016, May 2017)

Q. Explain perspective projection.
MU - May 2013, Dec. 2014. 5 Marks

Q. Explain perspective projections. Derive the matrix for perspective projection.
MU - May 2014, May 2015, Dec. 2016, Dec. 2015, May 2016. 5 Marks

Q. Write short note on : Perspective Projections.
(4 Marks)

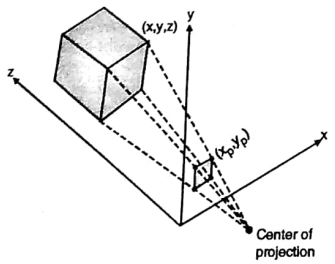


Fig. 7.3.10

- In real world, the objects which are away from the viewer, appear smaller. Perspective projection preserves this property. In perspective projection, the lines of projection are not parallel. Here all projection lines are ending at a single point called center of projection. In real world this center of projection is human eye. For perspective projection see Fig. 7.3.10.
- To draw the image of object on 2D plane, assume that the object is in space having some x, y, and z values. We have to draw the image on XY plane, so the center of projection should be beyond the XY plane. All projections which are originating from different vertices of object meet at center of projection point, intersecting XY plane. Assuming that the plane XY or screen is placed at z = 0.
- So the distance of the center of projection from the XY plane, i.e. plane whose z = 0 is called 'd'. With this assumption computations become simple. The projection of a point is the intersection of the straight line from that point to the center with the plane z = 0. Fig. 7.3.11, shows side view of the projection of the point (x, y, z) which gives projected point (x_p, y_p)

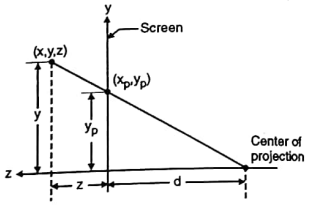


Fig. 7.3.11

- To draw the image of an object we have to find the values of (x_p, y_p), as (x_p, y_p) is intersection point of projection with screen. To obtain y_p, we have to say

$$\frac{y}{(z+d)} = \frac{y_p}{d}$$

$$\therefore \quad y_p = \left(\frac{y}{(z+d)}\right) \cdot d$$

- Similarly to obtain x_p, we have to consider top view of Fig. 7.3.10. It will be as shown in Fig. 7.3.12.

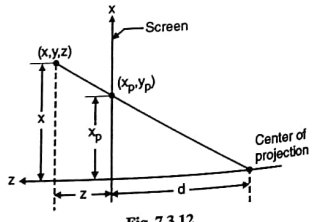


Fig. 7.3.12

Here, to find x_p, we have to say

$$\frac{x}{(z+d)} = \frac{x_p}{d}$$

$$\therefore \quad x_p = \left(\frac{x}{(z+d)}\right) \cdot d$$

∴ The co-ordinates of projected point on screen are

$$x_p = \frac{(d * x)}{(d+z)} \quad \text{and} \quad y_p = \frac{(d * y)}{(d+z)} \qquad z_p = 0$$

- It is not compulsory to take z = 0 and center on z-axis. We can have center on any other axis also, but then we have to use translation also and due to this computations will increase. Here we are using z-axis as it generates simple formula to find projected point on screen.

7.3.2(A) Types of Perspective Projections

- Vanishing point on the view plane is referred as a point where all the lines which are not parallel to view plane are appeared to meet. There are different types of perspective projections based on the number of vanishing points. We can have one point, two point and three point perspectives.

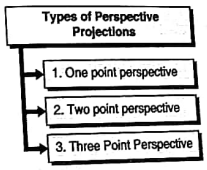


Fig. C7.4 : Types of Perspective projections

→ 1. One point perspective

- One point perspective occurs when the projection plane is parallel to two principal axes. Conversely, when the projection plane is perpendicular to one of the principal axis, one point perspective occurs. Receding lines along one of the principal axis converge to a vanishing point. Refer Fig. 7.3.13.

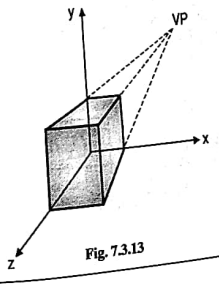


Fig. 7.3.13

→ 2. Two point perspective

- If the projection plane is parallel to one of the principal axes or if the projection plane intersects exactly two principal axes, a two-point perspective projection occurs. Refer Fig. 7.3.14.

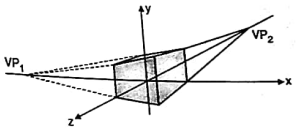


Fig. 7.3.14

→ 3. Three Point Perspective

- If the projection plane is not parallel to any principal axis, a three-point projection occurs. Refer Fig. 7.3.15.

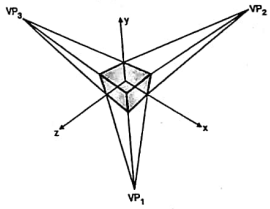
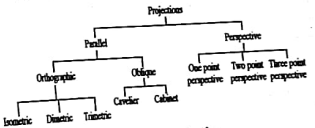


Fig. 7.3.15

- We can represent 3D projections in hierarchical format as follows :



☞ C program for Parallel Projection

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

void accept(int[],int[],int[],int);
void display(int[],int[],int);
void display_ortho(int[],int[],int);
void cavalierOblique(int [] , int [],int [] , int , float , int [],int
[]);

int main()
```

```
{
    int gd=DETECT,gm;
    int x[10],y[10],z[10],no,xp[10],yp[10],choice;
    float angle1;
    initgraph(&gd,&gm, "c://tc//bgi");

    printf(" Enter no of vertices for object\n");
    scanf("%d",&no);
    printf("Enter angle Phi\t");
    scanf("%f",&angle1);
     accept(x,y,z,no);

    printf("Orthographic Projenction \n");
    display_ortho(x,y,no);

    printf("Cavalier Oblique Projenction(RED)\n");
    cavalierOblique(x,y,z,no,angle1,xp,yp);
    setcolor(RED);

    display(xp,yp,no);
    return 0;
}
void  accept(int x[],int y[],int z[],int no)
    {
        int i;
        printf("Enter co-ordinate\n");
        for (i = 0; i < no; i++)
        {
            printf("P[%d]\t",i+1);
            scanf("%d",&x[i]);
            scanf("%d",&y[i]);
            scanf("%d",&z[i]);
        }
    }

void display_ortho(int x[],int y[],int no)
{
    int i;
    if(no==1)
     putpixel(x[0],y[0],YELLOW);
    else
    {
        for (i = 0; i < no-1; ++i)
        line(x[i],y[i],x[i+1],y[i+1]);
    }

if(no>2)
  line(x[0],y[0],x[no-1],y[no-1]);
}
void cavalierOblique(int x[],int y[],int z[],int no,float
angle1,int xp[],int yp[])
{
    int i;
    for (i = 0; i < no; ++i)
    {
        xp[i]=x[i]+(z[i]/tan(45))*cos(angle1);
        yp[i]=y[i]+(z[i]/tan(45))*sin(angle1);
    }
}

void display(int xp[],int yp[],int no)
    {
        int i;
        if(no==1)
         putpixel(xp[0],yp[0],RED);
else
        {
            for (i = 0; i < no-1; ++i)
            line(xp[i],yp[i],xp[i+1],yp[i+1]);
        }

if(no>2)
 line(xp[0],yp[0],xp[no-1],yp[no-1]);
```

7.4 3D Viewing Parameters

- We can form a 2D image from 3D object by using parallel and perspective projections. But if we want to look at the object from other side or from bottom side or from behind then we have to apply some rotation transformations on the object before projecting. Here we are having two options.
  - Keep view plane fixed and rotate the object.
  - Keep object fixed and view plane is repositioned.
- Generally we are using second option where object is fixed and the view plane, i.e. plane on which projections are taken is shifted. Let us take an example of photographer camera. When we are taking photograph of any object, an image of object is developed on paper. Here developing paper is acting as a view plane. If we change the position of camera, naturally we are getting another view of an object. See Fig. 7.4.1.

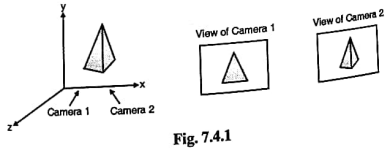


Fig. 7.4.1

- By setting the viewing parameters we can change the camera position. The first parameter is **view reference point** (X_{VR}, Y_{VR}, Z_{VR}). This point is a center of attention. Other parameters are expressed relative to this point. If we rotate the view it will be a rotation about the view reference point. We can consider as if a elastic thread is tied to view reference point and other end of that thread is tied to camera (view plane). By changing the other parameters we can change the position of the camera.
- Another parameter is **view plane normal vector** [DXN, DYN, DZN]. This is a vector which gives direction of elastic thread, which is perpendicular to view plane. It means the camera will always looks along the string toward the view reference point. See Fig. 7.4.1.

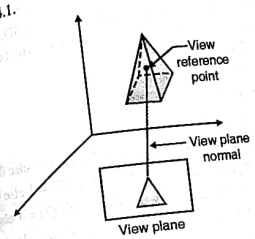


Fig. 7.4.2

- The third parameter, **view-distance parameter**, gives the length of the string from view reference point to view plane or camera. The view plane is placed at some distance away from the view reference point in the direction of view plane normal. These parameters allow the user to select how to view the object.

Syllabus Topic : 3D clipping

7.5 3D Clipping

Q. What is 3D clipping? Derive equations for all the planes.(left, right, top, bottom, front, back). **(10 Marks)**

- 3D clipping is conceptually same as 2D clipping. In 2D we are clipping the lines or objects by using clipping window. But in 3D we are using **Clipping Volume** or **View Volume**. In simple terms we can say it is a 3D box. The objects within this 3D box or view volume may be seen, while those outside are not displayed. It means we are clipping the portion of object which lie outside the 3D box. As view volume is nothing but a box, we are clipping the object with respect to six sides. i.e. in addition to left clip, right clip, bottom clip and top clip we have to use front clip and back clip also.
- In 3D clipping, left and right clipping deals with x co-ordinates; top and bottom clipping deals with y co-ordinates and front and back clipping deals with z co-ordinates. Here we require the front plane to be parallel to the view plane, which normally coincides with the screen. This view volumes contents will be projected onto the screen, which acts as a 2D window.
- Different types of projections create different shapes of view volumes. In case of **orthographic parallel projections**, the view volume is a rectangular box whose front plane is parallel to the view plane. Fig. 7.5.1 shows orthographic view volume.

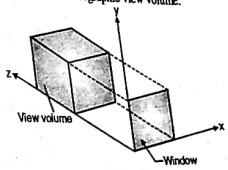


Fig. 7.5.1

- Fig. 7.5.1 shows view volume and its projections on XY plane. Here meaning of window is same as that of 2D. Remember that this window is on XY plane. So the portion of object which lies outside the window should be discarded. Similarly in case of perspective projections, the shape of view volume will be truncated pyramid. The pyramid's apex will be the center of projection. Fig. 7.5.2 shows perspective view volume.
- The front plane is parallel to the view plane, but it must lie in front of the center of projection. The objects which are inside the truncated pyramid are visible.

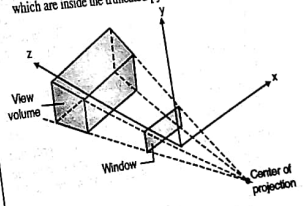


Fig. 7.5.2

7.5.1 3D Clipping

- The concept of 2D clipping is extended to achieve 3D clipping. IN 3D, depth clipping is the most important factor. Here we are having two options. First option will be, clip a 3D object in space only and then project that clipped object on XY plane, but clipping in space is conceptually more difficult and it require lot of computations so we will select second option. Then we are clipping only depth of the object in space. Then we will project that depth clipped object on view plane and apply our normal 2D clipping algorithm.

Parallel projections

- For parallel projections we are forming a rectangular box as view volume. This box will be in space. See Fig. 7.5.3.

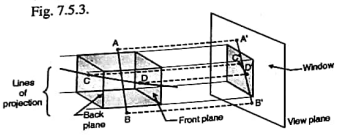


Fig. 7.5.3

- Before taking projections of an object first we are going to clip the depth i.e. we are comparing all z co-ordinates of object with back plane and front plane. The points or part of object which lies beyond back plane and in front of the front plane are ignored. The portion of object which is in between back and front plane will be considered for further processing. Remember that we are not clipping object with respect to top, bottom right or left boundary.
- If we keep the window parallel to z-axis with $z = 0$, then we will get image on XY plane, which is similar to 2D clipping.
- Refer Fig. 7.5.3, where we are forming a rectangular box and we are having two lines, AB and CD. When we are considering depth clipping, we are checking depth of end points of lines with back and front plane. As point C is beyond back plane and point D is in front of front plane, so we are clipping line CD with respect to back and front plane. Similarly we need to do it for line AB also.
- But AB is not beyond back plane and not in front of front plane, so there is no question of clipping line AB (Even though line AB is not completely inside view volume). Then we are taking projections of clipped line CD and line AB on view plane; at the same time we are taking projection of 3D box also on view plane which will act as window.
- Now we can apply our normal 2D clipping algorithm to clip the objects four sides.
- After taking projection of lines AB and CD on view plane, we will get Fig. 7.5.4.

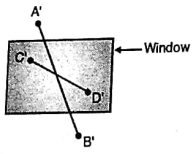


Fig. 7.5.4

- Now we can apply our normal Cohen-Sutherland line clipping algorithm. It means the whole clipping process will be as shown in Fig. 7.5.5.

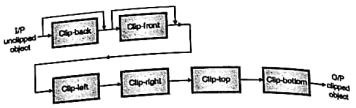


Fig. 7.5.5

- When we are using Cohen-Sutherland algorithm for line clipping in 3D, we have to slightly modify it. Normally in Cohen-Sutherland we are having four bit outcode [as we are having four boundaries i.e. Above, Below, Right and Left]. But in case of 3D, we need 6 bit outcode. The additional two bits will be for Front and Back plane.
- So now our outcode will become

"FBABRL"

where

if $x <$ left edge of window, set bit (1) = 1 else 0.

if $x >$ right edge of window, set bit (2) = 1 else 0.

if $y <$ bottom edge of window, set bit (3) = 1 else 0.

if $y >$ Top edge of window, set bit (4) = 1 else 0.

if $z >$ Back plane, set bit (5) = 1 else 0.

if $z <$ Front plane, set bit (6) = 1 else 0.

- We calculate outcodes of both end points of the line. If both outcodes are zero, then the line is totally visible. If both outcodes are not zero then we have to perform logical AND operation. If result of AND operation is nonzero, then line is totally invisible. And if the result of AND operation is zero, it means we have to clip the line i.e. the line is partially visible.

Example

- Suppose a line is formed by two points P_1 and P_2. See Fig. 7.5.6.

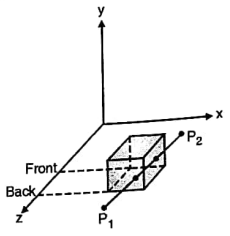


Fig. 7.5.6

Let us say $P_1 = x_1, y_1, z_1$ and

$P_2 = x_2, y_2, z_2$

- Here we are using equation of line in parametric form

$$x = x_1 + u(x_2 - x_1)$$
$$y = y_1 + u(y_2 - y_1)$$
$$z = z_1 + u(z_2 - z_1)$$

- To find intersection point with front plane

i.e. $z = f$

$$\therefore \quad z = z_1 + u(z_2 - z_1)$$
$$\therefore \quad f = z_1 + u(z_2 - z_1)$$
$$\therefore \quad u = \frac{f - z_1}{z_2 - z_1}$$

- Placing this value of 'u' in remaining equations, we get,

$$x = x_1 + \left(\frac{f - z_1}{z_2 - z_1}\right) \cdot (x_2 - x_1)$$
$$y = y_1 + \left(\frac{f - z_1}{z_2 - z_1}\right) \cdot (y_2 - y_1)$$
$$z = f$$

- Similarly we have to obtain intersection points for back plane and other planes also. We can use a mid point subdivision algorithm also to clip a line in 3D. Here we have to find mid point as (x_m, y_m, z_m) and compare z_m with front or back plane till z_m = Front or Back

Perspective Projections

- Like parallel projections we can create a box or view volume in perspective projections also. Here we are using center of projection which will be before the view plane and window. To develop the mathematics for perspective view volume we have to make following assumptions.
  - Center of projection is the middle of window at a distance 'd' from window
  - Window is in plane $z = 0$
  - View plane does not coincide with front plane
  - The window of view plane will be bounded by A, B, R, L. See Fig. 7.5.7.

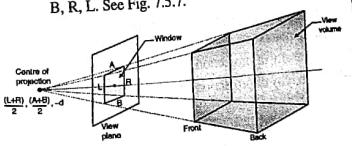


Fig. 7.5.7

- Here center of projection will be $\left(\frac{L+R}{2}, \frac{A+B}{2}, -d\right)$
- We need four constants to express the equations of the four side planes. These are the slopes of the planes in relation to the z-axis Fig. 7.5.8 shows vertical cross section of Fig. 7.5.7. to determine the slopes for the top and bottom planes.

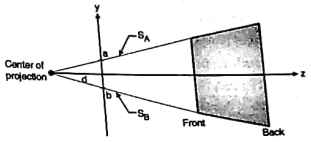


Fig. 7.5.8

- Slope of above plane $(S_A) = \frac{dy}{dx}$

$$\therefore S_A = \frac{(a - b) * (0.5)}{d}$$

Similarly

- Slope of bottom plane $(S_B) = \frac{dy}{dx}$

$$\therefore \quad S_B = \frac{(a - b) * (-0.5)}{d}$$

- On the same line we can find slopes of left and right planes also. See Fig. 7.5.9.

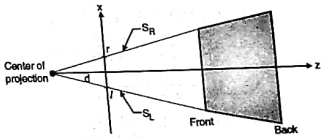


Fig. 7.5.9

Slope of right plane $(S_R) = \frac{dy}{dx}$

$$\therefore \quad S_R = \frac{(r - l) * (0.5)}{d}$$

Similarly

Slope of left plane $(S_L) = \frac{dy}{dx}$

$$\therefore \quad S_L = \frac{(r - l) * (-0.5)}{d}$$

- Now with these constants we have to form equation of planes. Let us consider top/above plane. But this plane is just like a line. See Fig. 7.5.9. So we will use equation of line as $y = mx + c$.

$\therefore$ for above plane

$y = S_A \cdot z + a$

Where S_A = slope

a = y intercept

Similarly,

We have to form other equations also

for bottom plane $y = S_B \cdot z + b$

for left plane $x = S_L \cdot z + l$

for right plane $x = S_R \cdot z + r$

Similarly for front (near) plane $z = n$

And for back (far) plane $z = f$

- Now to find the intersection of a line from (x_1, y_1, z_1) to (x_2, y_2, z_2) with six planes are as follows.

$$x = x_1 + u(x_2 - x_1)$$
$$y = y_1 + u(y_2 - y_1)$$
$$z = z_1 + u(z_2 - z_1)$$

Here for each edge we have to find separate 'u' value.

Let us take an example of above plane.

$$y = y_1 + u(y_2 - y_1)$$
$$\therefore \quad u = \frac{y_1 - y}{y_1 - y_2}$$

Now putting $y = S_A \cdot z + a$ in this equation

We get $u_A = \dfrac{y_1 - a - S_A \cdot z_1}{y_1 - y_2 + S_A (z_2 - z_1)}$

Similarly we have to find U_L, U_R, U_B, U_F, U_N

$$\therefore \quad U_L = \frac{x_1 - l - S_L \cdot z_1}{x_1 - x_2 + S_L (z_2 - z_1)}$$

$$U_R = \frac{x_1 - r - S_R \cdot z_1}{x_1 - x_2 + S_R (z_2 - z_1)}$$

$$U_B = \frac{y_1 - b - S_B \cdot z_1}{y_1 - y_2 + S_B (z_2 - z_1)}$$

$$U_N = \frac{z_1 - n}{z_1 - z_2}$$

$$U_F = \frac{z_1 - f}{z_1 - z_2}$$

- By placing appropriate u-value in parameterized line equation, we can find intersection point (x, y, z) with any of the planes. The point (x, y, z) will be inside the view volume if

for left plane $x \geq l + S_L \cdot z$

for right plane $x \leq r + S_R \cdot z$

for above plane $y \leq t + S_A \cdot z$

for bottom plane $y \geq b + S_B \cdot z$

for near plane $z \geq n$

for far plane $z \leq f$

- Like this we are going to determine that whether a particular point is inside the view volume or not.

Important Questions

Here we are mentioning some important questions from the topic which we have seen and how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 Explain how rotation about z axis in 3D, is same as rotation of 2D.

Ans. :

The answer for this question should include following points.

- Explain anticlockwise / clockwise rotation matrix for 2D.
- Draw diagram and find x_{new} and y_{new} for 2D.
- explain rotation about z-axis of 3D by using diagram.
- Derive the rotation matrix from equations
- Show that rotation about z-axis in 3D is same as rotation of 2D.

Q. 2 Explain various steps to perform rotation about an arbitrary axis in 3D.

Ans. :

The answer for this question should include following points.

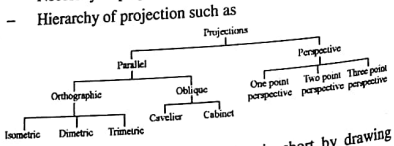
- All the seven step's in detail with diagram.
- Take a sample point on arbitrary line and show how it is performing rotation.
- Draw diagrams wherever necessary.
- Generate different equations and from that transformation matrices.

Q. 3 Explain 3D projections and its types.

Ans. :

For the answer of such question we have to include following points.

- Necessity of projection
- Hierarchy of projection such as



- Explain each type of projection in short by drawing appropriate diagrams.

Q. 4 Write a short note on parallel / perspective projection.

Ans. :

The answer for this question should include

- Necessity of projection.
- Draw an object and take its parallel / perspective projection on view plane
- Draw the 3D objects image on 2D plane
- Explain advantages and disadvantages of parallel / perspective projections.

Important Assignments

This chapter is important from practical examination point of view. We have already explained and given enough number of programs at the time of explaining algorithms. In addition to these programs, in final practical examination, few modifications are also expected. The list of such programs is given below :

(1) Write a program to scale a 3D cube Accept only one point of cube from user and generate the cube.

(2) Write a program to perform rotation about x, y, z-axis. Accept 3D cube from user.

(3) Write a program to perform reflection about xy, yz and xz planes.

(4) Write a program to perform rotation about arbitrary axis in 3D. Accept the axis and an object from user. Display the object after every step.

(5) Write a program to perform reflection about co-ordinate axes.

7.6 Important Questions and Answers

Q. 1 Write the matrices and draw the diagrams for scaling, rotation of a 3D object.

Ans. :

Scaling

Scaling a 3D point or object is much identical to scaling a 2D point or object. The scaling transformation relocates a point with relation to the origin. The formulas for 2D and 3D scaling are the same with addition of z co-ordinate. Basically scaling is nothing but changing the size of the object. The normal scaling matrix in 3D will be,

$$\begin{vmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{vmatrix}$$

where s_x, s_y and s_z represents the scaling factors for the three dimensions. 3D scaling matrix with homogeneous co-ordinate system will be,

$$\begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

In normal 2D, scaling matrix with homogenous co-ordinates is having size 3×3, whereas in 3D, it is of size 4×4. Here we are having additional factor s_z. Scaling a point is done by multiplication of scaling matrix with a point.

$$|x_{old}, y_{old}, z_{old}, 1| * \begin{vmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = |x_{new}, y_{new}, z_{new}, 1|$$

where, $x_{new} = x_{old} * s_x$

$y_{new} = y_{old} * s_y$

$z_{new} = z_{old} * s_z$

By properly selecting the values of s_x, s_y and s_z we can either shrink or elongate the object. For diagram of 3D Scaling refer Fig. 1.

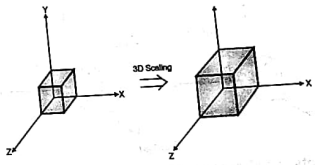


Fig. 1

Rotation of 3D object :

There are three types of rotations in 3D transformations.

Rotation about z-axis : When we are saying that we want to rotate a point, in anticlockwise direction, about z-axis it is very similar to performing a 2D rotation in anticlockwise direction.

If we perform rotation about z-axis (Clockwise/Anticlockwise), z co-ordinates remains unchanged and x and y co-ordinates changes as in case of 2D. So after performing rotation about z-axis, in anticlockwise direction, x_{new}, y_{new} and z_{new} will become,

$$\left.\begin{aligned} z_{new} &= z_{old} \text{ i.e. unchanged} \\ x_{new} &= x_{old} \cdot \cos\theta - y_{old} \cdot \sin\theta \\ y_{new} &= x_{old} \cdot \sin\theta + y_{old} \cdot \cos\theta \end{aligned}\right\} \text{ Same as 2D}$$

Now let's derive a transformation matrix for rotation about z-axis, in anticlockwise direction. In homogeneous co-ordinates it will become,

$$[x_{old}, y_{old}, z_{old}, 1] \cdot \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} = [x_{new}, y_{new}, z_{new}, 1]$$

where, $x_{new} = x_{old} \cdot \cos\theta - y_{old} \cdot \sin\theta$

$y_{new} = x_{old} \cdot \sin\theta + y_{old} \cdot \cos\theta$

$z_{new} = z_{old}$

Therefore the transformation matrix will be,

$$R_z = \begin{vmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For anticlockwise)

The sign will change according to the angle of θ. If θ is negative, i.e. for clockwise, it will be

$$R_z = \begin{vmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For clockwise)

Where R_z stands for rotation about z-axis.

For diagram of rotation about Z axis, refer Fig. 1(a)

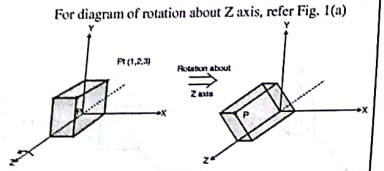
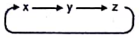


Fig. 1(a)

Rotation about x-axis :

The rotation about x-axis can be obtained from transformation matrix of rotation about z-axis with a cyclic permutation of the co-ordinate parameters x, y and z. It means we have to replace y co-ordinates by x co-ordinate; z by y and x by z co-ordinates.



...(1)

The equations of rotation about z-axis is,

$$x_{new} = x_{old} \cdot \cos\theta - y_{old} \cdot \sin\theta$$
$$y_{new} = x_{old} \cdot \sin\theta + y_{old} \cdot \cos\theta$$
$$z_{new} = z_{old}$$

By making use of Equation (1) in above equations we get,

$$y_{new} = y_{old} \cdot \cos\theta - z_{old} \cdot \sin\theta$$
$$z_{new} = y_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$
$$x_{new} = x_{old}$$

Here x-co-ordinate remains unchanged and y and z values gets changed.

Therefore the transformation will be,

$$R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For Anticlockwise)

and $R_x = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$ (For clockwise)

Where R_x stands for rotation about x-axis.

For diagram of rotation about X axis, refer Fig. 1(b).

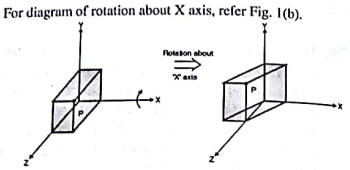


Fig. 1(b)

Rotation about y-axis :

To form a transformation matrix for rotation about y-axis, we are making use of transformation matrix of rotation about x-axis with a cyclic permutation of the co-ordinate parameters x, y and z.

The equations of rotation about x-axis is,

$$x_{new} = x_{old}$$
$$y_{new} = y_{old} \cdot \cos\theta - z_{old} \cdot \sin\theta$$
$$z_{new} = y_{old} \cdot \sin\theta + z_{old} \cdot \cos\theta$$

By making use of cyclic permutations of x, y and z in above equations we get,

$$y_{new} = y_{old}$$
$$z_{new} = z_{old} \cdot \cos\theta - x_{old} \cdot \sin\theta$$
$$x_{new} = z_{old} \cdot \sin\theta + x_{old} \cdot \cos\theta$$

Here y co-ordinate remains unchanged and x and z values gets changed

Therefore the transformation matrix will be,

$$R_y = \begin{vmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$ (For Anticlockwise)

And $R_y = \begin{vmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$ (For clockwise)

For diagram of rotation about Y axis, refer Fig. 1(c).

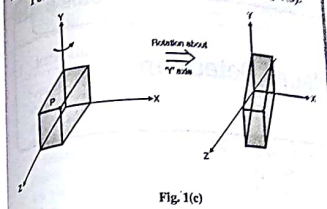


Fig. 1(c)

7.7 Exam Pack (University and Review Questions)

Syllabus Topic : 3D Transformations - Translation, Rotation , Scaling and Reflection. Composite transformations - Rotation about an arbitrary axis 3D transformation pipeline

Q. Write a short note on 3D rotation.
(Refer Section 7.2.3) **(May 2017, 10 Marks)**

Q. Derive all the necessary matrices required to perform 3D Rotation about arbitrary axis.
(Refer Section 7.2.3(D)) **(10 Marks)**

Syllabus Topic : Projections – Parallel , Perspective (Matrix Representation)

Q. Explain Parallel projection.
(Refer Section 7.3.1)
(May 2013, May 2014,Dec. 2014, May 2015, Dec. 2015, May 2016, Dec. 2016, 5 Marks)

Q. Differentiate between parallel and perspective projections. Explain with the help of examples.
(Refer Section 7.3.1) **(May 2017, 10 Marks)**

Q. Differentiate parallel and perspective projection.
(Refer Section 7.3.1) **(5 Marks)**

Q. Explain perspective projection.
(Refer Section 7.3.2)
(May 2013, Dec. 2014, 5 Marks)

Q. Write short note on : Perspective Projections.
(Refer Section 7.3.2) **(May 2011, 20 Marks)**

Q. Explain perspective projections. Derive the matrix for perspective projection.
(Refer Section 7.3.2)
(Dec. 2011, May 2014, May 2015, Dec. 2016, Dec. 2015, May 2016, 5 Marks)

Syllabus Topic : 3D clipping

Q. What is 3D clipping? Derive equations for all the planes.(left, right, top, bottom, front, back).
(Refer Section 7.5) **(10 Marks)**

□□□

Module 6

CHAPTER 8

Visible Surfaces Detection

Section Nos.	Name of the Topic
8.1	Introduction
8.2	Back Surface Detection
8.3	Depth Sorting Methods
8.4	Painter's Algorithm
8.5	Area Subdivision Method
8.6	Binary Space Partition Method

Chapter 8 discusses different ways to recognize the parts of scene that are behind other objects or getting hidden. As we are not able to see those parts, we can't draw them. This chapter has six sections.

8.1 **Introduction** shows why removing hidden lines and surfaces is important and gives some background information.

8.2 **Back Surface Detection** needs that we should identify those faces of an object which are away from the viewer.

8.3 **Depth Sorting Methods** explains the easy and simple way to detect the hidden portion of the object. Algorithm's like Z Buffer is explained with its advantages and disadvantages.

8.4 **Painter's Algorithm** explains how to remove the hidden faces or portion of the scene. This section also explains Warnock's algorithm.

8.5 **Area Subdivision Method** explains the use of algorithm for detecting and removing the hidden surfaces.

8.6 **BSP Method** explains the two step recursive way to display the polygons which are at far distance first and the nearest polygons at last without taking help of z co-ordinates.

Syllabus Topic : Classification of Visible Surface Detection Algorithm

8.1 Introduction

- Suppose we are looking at some object 'A' and then some other object say 'B' comes in between our eye and object 'A', then in that case we are not able to see the object 'A' fully or partially.
- It means object 'B' interrupts the projected path of object 'A'. In fact some surfaces of this new object 'B' are also not visible because they are eclipsed by object B's visible parts. See Fig. 8.1.1.
- The surfaces that are blocked or hidden from view must be removed in order to construct a realistic view of 3D scene. So the object 'B' should be displayed with only three of the six faces as shown in Fig. 8.1.1(c).
- The identification and removing of these surfaces is called the hidden-surface problem or **visible surface detection**. And that we have to determine the closest visible surface along each projection line.

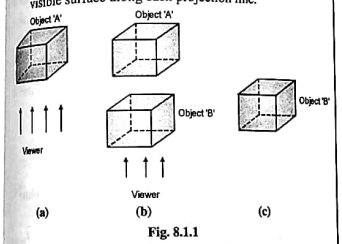


Fig. 8.1.1

- There are many visible surface detection algorithms. Each of them can be characterized as either an image-space method or an object-space method. In image space method pixel grid is used to determine the visibility at the pixel level. While in object-space method, the surface visibility is determined using continuous models in the object-space.

Syllabus Topic : Back Surface Detection Method

8.2 Back Surface Detection

→ **(Dec. 2016, May 2017)**

Q. Explain the Back face detection method with an example. **MU - Dec. 2016, May 2017. 10 Marks**

Q. Write short note on Back face removal technique. **(5 Marks)**

- Object surfaces that are oriented away from the viewer are called back surfaces or faces. The back surfaces of a cube are completely blocked by the cube itself and hidden from view. Therefore, we can identify and remove these back surfaces. We know that equation of plane is given by,

$$A_x + B_y + C_z + D = 0 \quad \ldots(8.2.1)$$

- In object space method, the identification of back surfaces is based on above equation. From the above equation, we can say, if a point (x, y, z) satisfies the equation then the point (x, y, z) is lying on the plane. But,

if $A_x + B_y + C_z + D < 0$ ...(8.2.2)

It means (x, y, z) lies on negative side.

And if $A_x + B_y + C_z + D > 0$ ...(8.2.3)

It means (x, y, z) lies on positive side.

- If we consider any point (x, y, z) as viewing point, then any plane which satisfies the Equation (8.2.2) must be a back face. After finalizing the back face we have to remove it from the further visibility.
- Let N = (A, B, C) be the normal vector. In right handed system with viewing direction along the negative z axis, the polygon is a back face if C < 0. See Fig. 8.2.1.

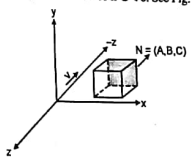


Fig. 8.2.1

- Also we cannot see any face whose normal has z component C = 0. Thus we can say any polygon is classified as a back face when its normal vector is negative or equal to zero i.e. C ≤ 0.

Syllabus Topic : Depth Buffer Method, Depth Sorting Method, Scan Line Method

8.3 Depth Sorting Methods (Algorithms)

- We now look at the few algorithms for hidden-surfaces. But many algorithms are based on the concept of depth comparison. So let us first try to understand the concept of depth comparison. The hidden-surface algorithm must be capable of finding the edges or surfaces which are visible.
- Suppose there are two points A (x_1, y_1, z_1) and B (x_2, y_2, z_2). See Fig. 8.3.1. Now we want to know which point is getting hidden out of A and B. In other words, whether point A is hiding point B or vice versa. For this we have to apply two steps process.

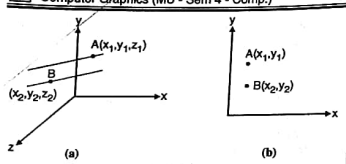


Fig. 8.3.1

(I) If both points A and B are not lying the on same projection line then no point hides another. See Fig. 8.3.1(a). It means that if both points are not co-linear then both will be visible. And therefore we have to take projection of both A and B points on XY plane for display. The points A and B will be displayed as shown in Fig. 8.3.1(b).

(II) But if both points A and B are lying on same projection line, i.e. both points are co-linear, then either one of these two points get hidden by other. See Fig. 8.3.2.

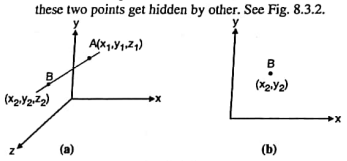


Fig. 8.3.2

- As both points A and B are co-linear, the projection of both points in 2D plane i.e. XY will be only single point. This case will happen when $x_1 = x_2$ and $y_1 = y_2$. But in 2D we are giving which name to this point ? For that we have to use third parameter of a point i.e. z co-ordinate. This z co-ordinate is nothing but a depth. If z co-ordinate of point A i.e. z_1 is smaller than z co-ordinate of point B i.e. z_2 then z_2 is in front of z_1. It means point B hides point A. So in 2D we have to plot point B only see Fig. 8.3.2(b). If z_1 is greater than z_2 i.e. point A is in front of B then we have to display point A.

8.3.1 Z - Buffer (Depth Buffer)

Q. Write short notes on: Depth Buffer algorithm (5 Marks)
Q. What is Z-buffer algorithm? (5 Marks)

- Another way to handle hidden lines and surfaces is z buffers. It is also called as depth buffer algorithm. Here we are sorting the polygons according to their position in space. And then in frame buffer itself. We are storing polygons which are closer to viewer. We know that frame buffer is used to store the images which we want to display on monitor. Here for visibility test we are making use of z buffer along with frame buffer.
- The z buffer is a large array to hold all the pixels of display. z buffer is somewhat similar to frame buffer. In frame buffer we are having arrays to store x and y co-ordinates of an image. Similarly z buffer contains z co-ordinates of pixels which we want to display. It helps to sort the polygons by comparing their z position. In simple terms it keeps track of nearest z coordinate of those points which are seen from the pixel (x, y).
- When there is nothing to display on monitor i.e. frame buffer is empty, at that time we have to initialize z buffer elements to a very large negative value. A large negative value on z axis represents a point beyond which there is nothing i.e. setting background color.

$\therefore \quad z_{buffer}(x, y) = z_{initial}$

- Polygons will be entered one by one into frame buffer by the display file interpreter. During this process, for each pixel (x, y) that lies inside the polygon we have to calculate z (x, y) for that pixel (x, y). Then we have to compare the z position of the polygon point with the z_{buffer} (x, y) value and decide whether the new surface is in front of or behind the current contents of frame buffer.
- If the new surface has z value greater than z_{buffer} value, then it lies in front. So we have to modify the contents of z_{buffer} (x, y) by new z value and set the pixel value at (x, y) to the color of the polygon at (x, y).

```
i.e. if (z (x, y) > z_buffer (x, y))
     {
          z_buffer (x, y) = z (x, y)
          put pixel (x, y, polygon-color)
     }
```

- If the z value of new surface is smaller than z_{buffer} (x, y), then it lies behind some polygon which was previously entered. So the new surface should be hidden and should not be displayed. The frame buffer and z_{buffer} should not be modified here. Here the comparison should be carried out by using pixel by pixel method.

Advantages

(1) It is easy to implement.
(2) As z buffer algorithm processes one object at a time, total number of objects can be large.
(3) It does not require any additional technique.

Disadvantages

(1) It requires lots of memory as we are storing each pixel's z value.
(2) It is a time consuming process as we are comparing each and every pixel.

- To understand z buffer algorithm let us take an example. Suppose there are two points, A (x_1, y_1, z_1) and B (x_2, y_2, z_2) as shown in Fig. 8.3.3(a).

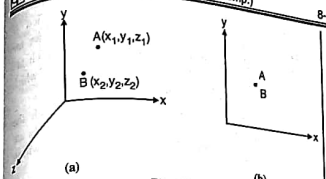


Fig. 8.3.3

- If $x_1 = x_2$ and $y_1 = y_2$ then in 2D we can represent these points by single location as shown in Fig. 8.3.3(b). Now we can use z buffer algorithm for visibility test. Let's first check point 'B'.

$z(B) = z_2$

i.e. $z(x_2, y_2) = z_2$

Now compare,

$z(x_2, y_2)$ with $z_{buffer}(x_2, y_2)$

But $z_{buffer}(x_2, y_2)$ = large negative value

So $z(x_2, y_2)$ is greater than $z_{buffer}(x_2, y_2)$.

$\therefore z_{buffer}(x_2, y_2) = z_2$ ...as $z(x_2, y_2) = z_2$

Now check for point 'A'.

$\therefore \quad z(A) = z_1$

i.e. $z(x_1, y_1) = z_1$

Now compare,

$z(x_1, y_1)$ with $z_{buffer}(x_1, y_1)$.

But we assumed that

$x_1 = x_2$ and $y_1 = y_2$.

$\therefore \quad z(x_1, y_1) = z(x_2, y_2)$ and $z_{buffer}(x_1, y_1) = z_{buffer}(x_2, y_2)$

$\therefore$ We have to compare

$z_{buffer}(x_1, y_1) = z_{buffer}(x_2, y_2)$ with z_1

But we know that,

$z_{buffer}(x_2, y_2)$ is set to z_2 at the time of point B.

So, we have to compare z_2 with z_1.

As z_2 is greater than z_1 so no need to modify $z_{buffer}(x_2, y_2)$.

- It means point B (x_2, y_2, z_2) is in front of point A (x_1, y_1, z_1). So there is no need to draw point A as it gets overlapped with B.

8.3.2 Scan Line → (May 2015)

Q. What is the use of Scan line method and explain all steps. MU - May 2015, 10 Marks

- A scan line method of hidden surface removal is an another approach of image space method. It is an extension of the scan line algorithm for filling polygon interiors. Here, the algorithm deals with more than one surfaces. As each scan line is processed, it examines all polygon surfaces interesting that line to determine which are visible. It then does the depth calculation and finds which polygon is nearest to the view plane. Finally, it enters the intensity value of the nearest polygon at that position into the frame buffer.
- We know that scan line algorithm maintains the edge list in the edge table (ET). The AET (Active Edge Table) contains only edges that cross the current scan line, sorted in order of increasing x. The scan line method of hidden surface removal also stores a flag for each surface that is set ON or OFF to indicate whether a position along a scan line is inside or outside of the surface. Scan lines are processed from left to right. At the leftmost boundary of a surface, the surface flag is turned ON; and at the right most boundary, it is turned OFF.
- The Fig. 8.3.4(a) illustrates the scan line method for hidden surface removal. As shown in the Fig. 8.3.4(a), the active edge list for scan line 1 contains the information for edges AD, BC, EH. And FG. For the positions along this scan line between edges AD and BC, only the flag for surface S_1 is ON. Therefore, no depth calculations are necessary and intensity information for surface S_1 is entered into the frame buffer. Similarly, between edges EH and FG, only the flag for surface S_2 is ON and during that portion of scan line the intensity information for surface S_2 is entered into the frame 'buffer'.

ET entry	X	Y max	Δ x	ID	→

PT entry	ID	Plane eq.	Shading info.	In-out

Fig. 8.3.4(a) : Illustration of scan line method of hidden surface removal

AET Contents

Scan line	Entries
Scan line 1	AD BC EH FG
Scan line 2	AD EH BC FG

Fig. 8.3.4(b) : ET, PT, AET for the scan line Algorithm

- For scan line 2 In Fig. 8.3.4(a), the active edge list contains edges AD, EH, BC and FG. Along the scan line 2 from edge AD to edge EH, only the flag for surface S_1 is ON, However, between edges EH and BC, the flags for both surfaces are ON. In this portion of scan line 2, the depth calculations are necessary. Here we have assumed that the depth of S_1 is less than the

depth of S_2 and hence the intensities of surface S_1 are loaded into the frame buffer. Then, for edge BC to edge FG portion of scan line 2 intensities of surface S_2 are entered into the frame buffer because during that portion only flag for S_2 is ON.

- To implement this algorithm along with AET we are required to maintain a polygon table (PT) that contains at least the following information for each polygon, in addition to ID.
  1. The coefficient of the plane equation.
  2. Shading or color information for the polygon.
  3. A in-out Boolean flag, initialized to false and used during scan line processing.
- The Fig. 8.3.4(b) shows the ET, PT and AET for the scan line algorithm.

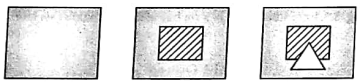
8.4 Painter's Algorithm

→ (May 2013)

Q. Explain Painter's Algorithm.

MU - May 2013. 10 Marks

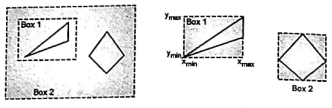
- The object surfaces that are away from the viewers are called as back faces. So we have to identify and remove those faces. For this there are many algorithms. **Painter's algorithm** is one of them. The painter's algorithm is also called as depth **sorting algorithm** or **priority algorithm**.
- The painter's algorithm processes polygons as if they are painted onto the view plane, in order of their distance from the viewer.
- It is similar to a work of painter. Painter first paints whole canvas by some color, which will be background color of picture. Then on top of this he paints with other colors or objects. This will be his foreground picture. See Fig. 8.4.1.



(a) (b) (c)

Fig. 8.4.1

- In Fig. 8.4.1(a) whole canvas is painted first then Fig. 8.4.1(b) shows on top of this canvas a rectangle is painted and Fig. 8.4.1(c) shows on top of this a triangle is painted. It gives appearance as the triangle is nearer to viewer than rectangle.
- It means more distance polygons are painted first and nearer distance polygons are painted over more distance polygons, partially or totally hiding them from view.
- The main factor over here is to decide the priority of the polygons to determine which polygons are to be painted first. It means when two polygons overlap, how to decide which polygon hides the other.
- For painters algorithm uses a technique called as **minimax test** or boxing test. From this test we are immediately sorting or isolating polygons which are overlapping (i.e. hiding by other polygons) and which polygons are not overlapping.
- The minimax test says that, if we place both the polygons in two different boxes and if these boxes are not overlapping, then it means the polygons within these boxes are not overlapping.
- To make efficient use of this test the bounding box must be as small as possible and still contain the polygon. It should not be as shown in Fig. 8.4.2(a). But should be as shown in Fig. 8.4.2(b).



(a) (b)

Fig. 8.4.2

- It means we have to form a box with boundaries as x_{max}, x_{min}, y_{max} and y_{min} of that polygon. See box 1 of Fig. 8.4.2(b).
- Now we will apply the test. If the two boxes are one over the another with same x_{min} and x_{max} values but different y_{min} and y_{max} values, as shown in Fig. 8.4.3, then two polygons which are inside these boxes are not overlapping.

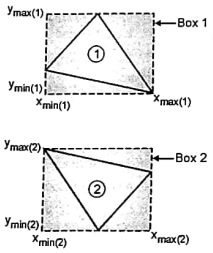


Fig. 8.4.3

Here, x_{min} and x_{max} are same for both boxes.

But $y_{min\,(1)} > y_{max\,(2)}$

- It means both polygons are visible, as there is no overlapping.
- Similarly, if the two boxes are one next to another, with same y_{min} and y_{max} values but different x_{min} and x_{max} values, as shown in Fig. 8.4.4, then both polygons are visible.

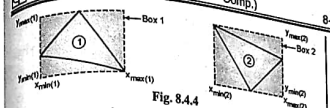
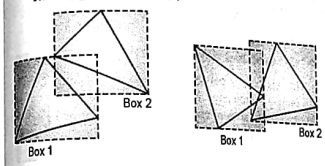


Fig. 8.4.4

Here, y_{min} and y_{max} are same for both boxes.

But $x_{min\,(2)} > x_{max\,(1)}$

- It means both polygons are not overlapping. So both are visible.
- Similarly we can check whether polygon one is to the right of polygon two and whether polygon one is below polygon two. If this Minimax test is true, then it means both polygons are visible, no one is hiding another. But if the boxes are overlapping then also we are not in a position to say firmly that the polygon is overlapping. Refer Fig. 8.4.5(a) and (b).



(a) No overlapping **(b) Overlapping**

Fig. 8.4.5

- So we have to perform some other test. This new test should be applied to z co-ordinates of the polygons. If the smallest z value for one polygon is larger than the largest z value of another polygon, then the first polygon lies in front of the other. Fig. 8.4.6 shows right handed system in which two polygons are there p_1 and p_2.
- Here both p_1 and p_2's boxes are overlapping, so are performing this test. Here polygon p_1 is in front of p_2, because z_{min} $(p_1) > z_{max}$ (p_2). But if z_{max} $(p_1) < z_{min}$ (p_2) then p_1 is behind p_2. So we have to store these polygons into a polygon list according to z_{min} of each polygon (in right handed system). It means we are storing smallest z co-ordinates of each polygon in an array. And first display the polygons whose z_{min} is smallest i.e. the last polygon, then next polygon and so on. At last we have to display a near polygon whose z_{min} is the maximum.

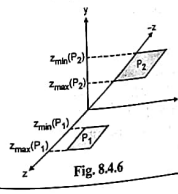


Fig. 8.4.6

Disadvantage of Painter's algorithm

- The hidden-surface techniques such as the painter's algorithm, which rely on the properties of the frame buffer, are not sufficient for calligraphic displays. On such displays we must not draw the hidden portions of lines.
- The first solution compared lines to objects. For each objects, it considered relevant edges to see if the object hid them. The object might not hide an edge at all or might hide it entirely. It might hide an end, making the visible portions of the edge smaller, or hide the middle, making two smaller visible line segments. After comparison of the line and object, the resulting visible line segments were compared in turn to the remaining objects.

Contour edges : We do not have to compare the line against all of the polygon edges in an object in order to tell if the object hides the line. The only edges which can change whether the line is visible or not are those on the boundary where a front face meets a back face. These are called *contour edges*.

Interior edges : For a solid object, each edge has two polygons adjacent to it. If the polygons which meet at the edge are both front faces and both back faces, then we have an *interior edge*; but if one is a front face and the other a back face, then it is a contour edge. (See Fig. 8.4.7).

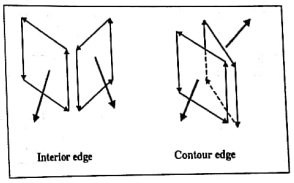


Fig. 8.4.7 : Interior and Contour Edges

- Compare contour edges of all objects to each line. For each intersection of a line with a contour edge, the line either passes behind an object or emerges from it. So with each such intersection, the number of faces hiding the line either increases or decreases by 1.
- Apple has termed the number of faces hiding a line is *quantitative invisibility*. His method for hidden-line removal is to find the quantitative invisibility for an initial point on a line, and then to follow along the connected lines, finding intersections with the contour edges and maintaining the quantitative invisibility.

Syllabus Topic : Area Subdivision Method

8.5 Area Subdivision Method

→ **(May 2015)**

Q. Write short note on : Area subdivision method. **MU - May 2015, 5 Marks**

Q. Explain Warnock's algorithm used to remove Hidden surfaces with example. **(10 Marks)**

- Area subdivision method is also called as **area based method** or **Warnock's algorithm**. In painter's algorithm, we have seen procedure to remove hidden-surfaces. It means painters algorithm deals with the procedure. But area subdivision method does not deal with the procedure to remove hidden surfaces. It just tries to display the final picture on screen. The basic idea is as the resolution of the display increases the size of the picture also increases and hence its correctness also increases.
- The algorithm is a recursive procedure based on a 2-step strategy.

Step 1 : Decide which polygons overlap the given area on the screen.

Step 2 : Which polygons are visible in that area.

- So, the area subdivision method checks whether the polygon is partially or fully visible in the given area. Let's call this area as 'A'. Then, each area is further subdivided into sample sub-areas and again the test is carried out for the each sub-area. It means whether the polygon is visible in that area or not till the visibility decision is made or until the screen area becomes a pixel. The algorithm starts with initial area as 'A' and divides that area in four sub-areas. See Fig. 8.5.1.

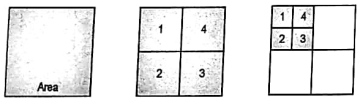


Fig. 8.5.1

Categories of polygons

- Here according to the screen area, we are classifying the polygons into four basic categories. These categories are as follows :

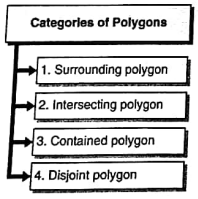


Fig. C8.1 : Categories of Polygon

→ (1) Surrounding polygon

- In this case a polygon completely covers the given screen area as shown in Fig. 8.5.2.

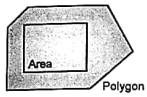


Fig. 8.5.2

→ (2) Intersecting polygon

- In this case a polygon is getting intersected with the screen area i.e. the polygon is partially inside the screen area as shown in Fig. 8.5.3.

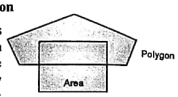


Fig. 8.5.3

→ (3) Contained polygon

- In this case a polygon is lying completely inside the given screen area as shown in Fig. 8.5.4.

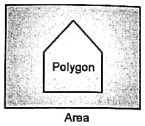


Fig. 8.5.4

→ (4) Disjoint polygon

- Whenever the polygon is lying completely outside the given screen area, it is called disjoint polygon. See Fig. 8.5.5.

Fig. 8.5.5

- Thus, the polygons which are coming under category four i.e. disjoint polygon, are totally invisible. So we are not considering them at all for visibility. Now we have to concentrate on, only first three categories. We will keep these polygons in a separate polygon list, which is called as Potentially Visible Polygon List (PVPL).
- If the polygon is of category one i.e. surrounding polygon, then whole area of the screen will get the color of surrounding polygon. For category two i.e. intersecting polygon, we will apply clipping algorithm to convert category two into category three and four. Fig. 8.5.6 shows that conversion.

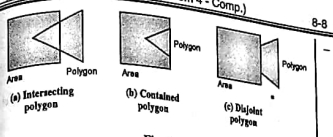


Fig. 8.5.6

- For category three i.e. contained polygon, we are going to subdivide the area into four sub-areas and again apply the same logic. There is no point in applying subdivision for case one and case four i.e. surrounding polygon and disjoint polygon case. Because as in surrounding case whole area A is visible so even if you further subdivide it, that area will be visible only. The same thing is for disjoint category. It means the subdivision procedure is to be applied to contained and intersection category only.
- Coming back to our Warnock algorithm or area subdivision method, if there are two polygons then the polygon is not visible, if it is in back of a surrounding polygon. Refer Fig. 8.5.7.
- If there are two polygons p_1 and p_2 : and p_2 surrounds p_1 then it means p_1 should not be displayed i.e. p_1 should be removed from PVPL.

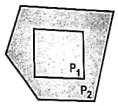


Fig. 8.5.7

- PVPL could be the list of smallest z co-ordinates of the polygons which are within the area. i.e. z_{min}. And let z_{smax} be the maximum z value of surrounding polygon. Fig. 8.5.7 is shown with different look in Fig. 8.5.8.
- From the Fig. 8.5.8 we can easily say that if $z_{max}(p_1) < z_{min}(p_2)$, then polygon p_1 is hidden. Here polygon p_2 is surrounding polygon. So we can say $z_{max}(p_1) < z_{min}$ then p_1 is hidden. In fact, all other polygons after p_1, i.e. whose z_{max} is smaller than $z_{max}(p_1)$, on the list will be hidden by p_2. So remove all them from PVPL.

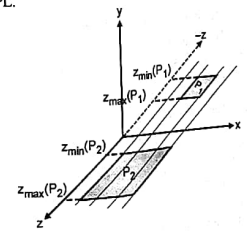


Fig. 8.5.8

- But if polygon p_1 is completely inside the polygon p_2 i.e. contained category, as shown in Fig. 8.5.9, then it will be exactly opposite.

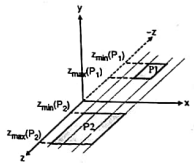


Fig. 8.5.9

- If p_1 is in completely inside p_2 then our conditions will be, If $z_{min}(p_1) > z_{max}(p_2)$ then p_1 is in front of p_2
- In this case, we have to display polygon p_1 and fill it with the color of p_1. Then we have to fill the remaining area by the color of polygon p_2.

Now let us summarize the algorithm.

Step 1 : Initialize the complete screen as area.

Step 2 : Create PVPL with respect to the area and sort on z_{min}.

Step 3 : Categorize the polygons.

Step 4 : Remove the disjoint polygons.

Step 5 : Now perform visibility test.

(i) If all the polygons are of disjoint category set pixel color as background.

(ii) If polygon is of contained category then color the polygon and remaining area as background color.

(iii) If we have single polygon in PVPL with surrounding category then set the whole area with color of surrounding polygon.

(iv) If area is a pixel (x, y), then find the z value of all the polygons at the pixel (x, y). We then choose the color of that polygon which has smallest z value and set pixel to this color.

Step 6 : If none of the above cases are true then subdivide the area further and goto step 2.

Step 7 : If we reached to a stage where we can take some decision on visibility test, then stop this process.

8.6 Binary Space Partition (BSP) Method

- A binary space partition tree is an efficient method for determining objects visibility by painting surfaces on to the screen from back to front, as in the painter's algorithm. The BSP tree method is a two step procedure:

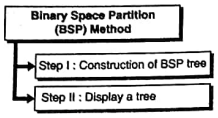


Fig. C8.2 : Binary Space Partition (BSP) Method

➜ **Step I :Construction of BSP tree**

- The BSP algorithm recursively subdivides the space into two half spaces. But the dividing plane is one of the polygon in the picture or scene.
- The other polygons in the scene are placed in appropriate half space.
- Referring Fig. 8.6.1, if we consider polygon A as a plane then on front side of A, the polygons will be B and C, whereas on backside of A the polygons will be D and E. But polygon F is getting intersected by plane A into two polygons F_a and F_b. Then each of this subdivided polygon is placed in appropriate half space. Here as we are considering polygon A as dividing plane, the root of BSP tree will be A only. The BSP tree at this level will be as shown in Fig. 8.6.2.

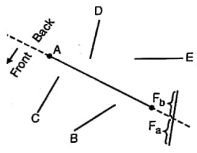


Fig. 8.6.1

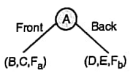


Fig. 8.6.2

- Each half space is then recursively subdivided, using one of the polygon in the half space as separating plane.
- This process of space subdivision continues till there is only one polygon in each half space. Because this is a binary tree.
- The subdivided space is then represented by a binary tree with the original polygon as the root.
- **Example :** Let us proceed with our example, we have divided the space into two parts, front and back. Let's concentrate on front part only. In front part we are having B, C and F_a polygons. So now we have to select one polygon out of these as a dividing plane. Let us select polygon B. See the Fig. 8.6.3(a).

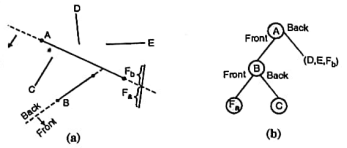


Fig. 8.6.3

- After selecting polygon B as a dividing plane, it places polygons C and F_a in back and front parts respectively. The BSP tree is also shown in Fig. 8.6.3(b). Now further we cannot divide F_a or C polygons, as there is nothing to divide. So we have to select another path. i.e. Back path of root, which is having D, E and F_b. Again divide this into two parts by selecting one of the polygon as a dividing plane. Let us select polygon E. See the Fig. 8.6.4(a).

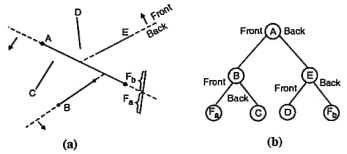


Fig. 8.6.4

- After selecting polygon E as a dividing plane, it places polygons D and F_b in front and back parts respectively. Now the BSP tree will become same as shown in Fig. 8.6.4(b). Further we cannot divide. So this is a final BSP tree.
- Remember that there is no unique BSP tree representation for a given scene, that will depend on selection of planes. For the given scene there could be more than one BSP tree representations.
- To reduce the number of polygons and so the size of BSP tree, it is advantageous if the separating planes create the minimum number of polygons to split.

➜ **Step II : Display a tree**

- Remember that our aim is to display the polygons which are closer to the view point or in other words, first display the polygons which are away from user and then display the polygons which are nearer. So there should be some relation between the view point and the root of the tree. The relation is, traverse the BSP tree inorder way i.e. first visit left node, then root and then right node.
- If the view point is in front of the root polygon, then BSP tree is traversed in the order of back branch, root, front branch i.e. reverse of inorder traversal.

- For our example, if the viewpoint is in front of the root polygon, as shown in Fig. 8.6.5(a) then the sequence of polygons for display will be F_b, E, D, A, C, B, F_a.
- If the view point is in back of the root polygon then BSP tree is traversed in the order of front branch, root, back branch i.e. normal inorder traversal.

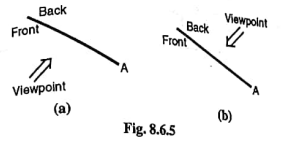


Fig. 8.6.5

For our example, if the view point is in back of the root polygon, as shown in Fig. 8.6.5(b), then the sequence of polygons for display will be F_a, B, C, A, D, E, F_b.

☞ **Advantages of BSP**

The main advantages of BSP method is :

- We are not dealing with z co-ordinates at all to decide the priority of the polygons for display.

☞ **Disadvantage of BSP**

- Waste of computational time as we are performing calculations for those objects also, which are hidden.

Important Questions

Here we are mentioning some important questions from the topic which we have seen and how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 Explain the depth comparison technique to remove back faces.

Ans. : The answer for this question should include following points.

- Explain the necessity to identify and removing of hidden points or faces.
- Take a suitable example and show how to remove hidden point.
- Explain how z co-ordinate plays important role in this matter.

Q. 2 Write a short note on z buffer.

Ans. : To write a short note for this we have to include following points :

- What is z buffer?
- Initial values of z buffer.
- How to modify the z buffer contents ?
- State advantages and disadvantages.
- Explain all this with the help of an example and suitable diagram.

Q. 3 Explain Painters algorithm.

Ans. : The answer for this question should include following points.

- The concept of painters algorithm.
- Minimax test.
- If the boxes of polygons are overlapping then what to do.
- Z co-ordinate test.
- Explain all this with the help of diagrams and suitable example.

Q. 4 What is a Warnock's algorithm ?

Ans. : To write the answer for this question we have to include following points.

- The difference between painters and Warnock algorithm.
- Different categories of polygons with neat diagrams.
- How to update PVPL ?
- Different conditions for surrounding and contained polygons.
- Take suitable example and explain the algorithm properly.

8.7 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Back Surface Detection Method**

Q. Explain the Back face detection method with an example. *(Refer Section 8.2)* **(Dec. 2016, May 2017, 10 Marks)**

Q. Write short note on Back face removal technique. *(Refer Section 8.2)* **(5 Marks)**

☞ **Syllabus Topic : Depth Buffer Method, Depth Sorting Method, Scan Line Method**

Q. Write short notes on: Depth Buffer algorithm. *(Refer Section 8.3.1)* **(5 Marks)**

Q. What is z-buffer algorithm? *(Refer Section 8.3.1)* **(5 Marks)**

Q. What is the use of Scan line method and explain all steps. *(Refer Section 8.3.2)* **(May 2015, 10 Marks)**

Q. Explain Painter's Algorithm. *(Refer Section 8.4)* **(May 2013, 10 Marks)**

☞ **Syllabus Topic : Area Subdivision Method**

Q. Write short note on : Area subdivision method. *(Refer Section 8.5)* **(May 2015, 5 Marks)**

Q. Explain Warnock's algorithm used to remove Hidden surfaces with example. *(Refer Section 8.5)* **(10 Marks)**

□□□

Module 7

CHAPTER 9

Illumination Models and Surface Rendering

Section Nos.	Name of the Topic
9.1	Introduction
9.2	Illumination Models
9.3	Colors
9.4	Color Models
9.5	Shading Methods
9.6	Halftone
9.7	Polygon Rendering

This chapter explains how we perceive light and colors, how we can describe color precisely, and how we compute colors. This chapter also explains different shading processes. This chapter has seven sections

9.1 **Introduction** points out how the use of color is important in computer graphics.

9.2 **Illumination models** consider both the sources of light and lights interaction with the surfaces of the objects that we display on the screen.

9.3 **Colors** explain how we perceive colors and how color primaries or their complements can be mixed to produce other colors.

9.4 **Color models** explains four systems for describing colors quantitatively, how these relate to color displays and how they are interrelated.

9.5 **Shading methods** explains different techniques for rendering the surfaces of displayed objects, making them appear smoothly rounded as desired.

9.6 **Halftone** explains tone of the image and Threshold and Dithering techniques.

9.7 **Polygon rendering** explains the rendering procedure and equation of rendering..

9.1 Introduction

- Until now, whenever we want to display anything on monitor, we are dealing with the intensity of pixels, as if we know only two colors, black and white. But we cannot neglect colors, which is the important property of light. Now we will try to understand colors, which are taken from the real world around us.
- In this chapter we will consider the shading of 3D objects also. We will try to develop a model for the manner in which light sources illuminate objects. We will also give stress on use of these models. We will discuss how colors are described and how the models are used by colored objects. We will also explain different shading algorithms.

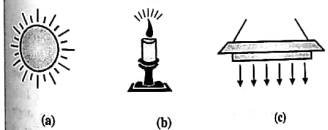
Syllabus Topic : Basic Illumination Models- Diffused Reflection, Phong Specular Relfection

9.2 Illumination Models

- Fundamentally lighting effects are described with models that consider the interaction with object surfaces. This interaction modifies the light in several ways. Once light reaches our eyes it gives signals to our perception process to decide what we have seen in actual scene. But visual perception is a complicated process. So we will not go in detail of that.
- Here we concentrate on the location and qualities of the light that falls on the objects and the way in which object interacts with it. A model for the interaction of light with a surface is called an **illumination model**.

9.2.1 Sources of Light

- Many objects which are available in nature may produce or emit light. These objects are called as light emitting sources. The example for this could be sun, lamp stars etc. See Fig. 9.2.1.



(a) (b) (c)

Fig. 9.2.1

- These light emitting sources are further categorized as **point source** and **distributed light source.** When the surface of the object which we want to illuminate is bigger than the surface of light emitting source, then we are calling that light emitting source as point source. See Fig. 9.2.1(b). But when the surface of light emitting source is greater than the surface of object then we are referring it as distributed light source. See Fig. 9.2.1(c).
- All objects which are not emitting light are grouped under separate category, which is called as **light reflecting sources**. See Fig. 9.2.2.
- When we try to produce realistic displays, we must consider this reflected light which is coming from all the directions. Such as light which is reflected from walls, floor, ceiling, objects in a room etc. It means the surface which is not directly exposed to a light emitting source will still be visible because of light reflecting sources. This light which is reflected from somewhere is called as **background light or ambient light.**

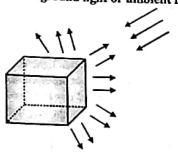


Fig. 9.2.2 : Light reflecting source

9.2.2 Diffuse Illumination

Q. What do you understand by Diffuse Illumination ? (5 Marks)

- When a light falls on any surface, it can be absorbed by the surface, while the rest will be reflected or retransmitted. In diffuse illumination, incoming light is not reflected in a single direction but is scattered almost in all directions. The part of incoming light will be absorbed by the surface. The light which is not absorbed will be reflected randomly in all directions. See Fig. 9.2.3.

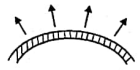


Fig. 9.2.3

- The ratio of light reflected from the surface to the total incoming light is called the **coefficient of reflection or reflectivity (K_d).** The white surface is having its reflectivity close to 1, hence it reflects almost all incoming light. Whereas the black surface absorbs most of the incoming light since its reflectivity is near to 0. So the value of reflectivity (K_d) is assigned in the range of 0 to 1, depending on the nature of surface. Intensity of the diffuse reflection at any point when surface is exposed to ambient light is given as,

$$I = K_d \cdot I_a$$

Where I_a = Intensity of ambient light

K_d = Coefficient of reflection.

- Shading model calculating intensities according to above equation will shade all visible surfaces of an object with same intensity. Realistic shading is obtained by including the effect of point sources in the shading model.

9.2.3 Point - Source Illumination

Q. What do you understand by Point Source Illumination ? (5 Marks)

- Light from point source differs from ambient light. The light originate from a fixed place, and it comes from a particular direction. Here the position and orientation of the objects surface relative to the light source will determine how much light the surface will receive.
- Surfaces which are near the light source and facing towards light will receive more light. The illumination is decreased by a factor of cos θ, where θ is the angle between the direction of light (L) and the direction normal (N) to the plane. This **angle θ** is called as **angle of incidence** which is shown in Fig. 9.2.4.

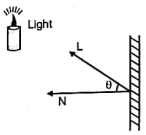
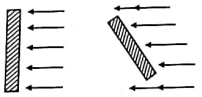


Fig. 9.2.4

- As the angle between direction of light and direction normal i.e. incidence angle increases, the incident light which falls on the surface will be less. Fig. 9.2.5 shows the difference between more illuminated and less illuminated objects because of angle of incidence.



(a) More illuminated **(b) Less illuminated**

Fig. 9.2.5

- If we denote angle of incidence as θ, the amount of illumination depends on cos θ. If I_P is the intensity of the point source light, then the diffuse reflection will be.

$$I = K_d \cdot I_p \cdot \cos\theta$$

- If the angle of incidence i.e. θ is in the range of 0 to 1 then only the surface will be illuminated by the point source. But if we place the value of cos θ in above equation then we will get,

$$\cos\theta = N \cdot L \quad \text{...(from Fig. 9.2.4)}$$

$$\therefore I = K_d \cdot I_p \cdot (N \cdot L)$$

- Now we can combine the equations of ambient (diffuse illumination) and point source to obtain an expression for total diffuse reflection.

$$\therefore I = K_a \cdot I_a + K_d \cdot I_p (N \cdot L)$$

where both K_a and K_d will depend on surface material and will be in the range of 0 to 1.

9.2.4 Specular Reflection

Q. Write short note on : Specular Reflection. (5 Marks)
Q. What do you mean by specular Reflection? (5 Marks)

- Specular reflection is a type of reflection which occurs at the surface of a mirror.
- Here, the light is reflected in a single direction. Many metals, glass and plastics have a specular reflection which is independent of color. In specular reflection, light comes in, strikes the surface and then bounces back. The angle which the reflected beam makes with the surface normal is called the angle of reflection. It is same in magnitude as that of angle of incidence. See Fig. 9.2.6.

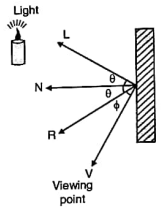


Fig. 9.2.6

- In Fig. 9.2.6, the vector 'L' points in the direction of light source. 'N' is the surface normal. The vector 'R' points in the direction of the reflected light. 'V' indicates unit vector pointing to the viewer from the surface position. Angle ϕ is the viewing angle relative to the specular reflection direction R. For an ideal reflector this angle ϕ should be zero. In this situation we could see only reflected light when vector V and R coincides.
- A shiny surface will reflect almost all the incoming light precisely in the direction of the reflection vector. So it is having narrow reflection range whereas a rough surface has wider reflection range as shown in Fig. 9.2.7.

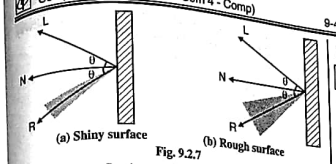


Fig. 9.2.7

- Specular reflection models must produce the highest intensity in the direction of R, with intensity decreasing rapidly as the viewing angle ϕ increases. The phong model sets the intensity of specular reflection proportional to $\cos^n\phi$ where angle ϕ, can be in the range of 0° to 90° i.e. cos ϕ in range of 0 to 1. And 'n' determines the type of surface that is to be viewed. A large value is assigned to 'n' if the surface is very shiny and 'n' will be small for rough surfaces. A perfect reflector, like mirror has 'n' as infinite.
- We can approximately model specular intensity variations using specular reflection coefficient ω(θ) for each surface. Generally as the angle of incidence increases ω(θ) also increases when θ = 90, at that time ω(θ) = 1 and all the incident light is reflected. So we can write specular reflection model as

$$I_{spec} = \omega(\theta) \cdot I \cdot \cos^n\phi$$

Where I = Intensity of light source
ϕ = angle between R and V

- But as V and R are unit vectors in the viewing and specular reflection directions, we can set

$$\cos\phi = V \cdot R$$

- We can further simplify the equation by assigning ω(θ) as a constant value (K_s) for the surface.

$$\therefore I_{spec} = K_s \cdot I \cdot (V \cdot R)^n$$

Where R can be represented in terms of L and N.

9.2.5 Reflections

- We should be able to see the specular reflections of objects, not just of lights. We look at the point p1 on an object and ask what light is coming from that point and from where it originates. We have seen how to find the light coming from diffuse illumination and directly from point sources. We can also follow a reflected ray back from our eye to the point and determine the incident ray. If we move back along the incident ray and reach a point P2 on some other surface, then we know that the light(I) from this second point will be reflected at P1. (See Fig. 9.2.8)

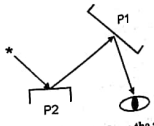


Fig. 9.2.8 : Light reflection from the another object

9.2.6 Ray-tracing

Q. Write short note on : Ray-tracing. (5 Marks)

- If we consider line of sight from a pixel position on the view plane through a scene, as in Fig. 9.2.9, we can determine which objects in the scene (if any) intersect this line. From the intersection points with different objects, we can identify the visible surface as the one whose intersection points is closest to the pixel. 'Ray Tracing' is an extension of this basic idea. Here, instead of identifying for the visible surface for each pixel, we continue to bounce the ray around the picture. This is illustrated in Fig. 9.2.10 when the ray is bouncing from one surface to another surface it contributes the intensity for that surfaces. This is simple and powerful rendering technique for obtaining global reflection and transmission effects.

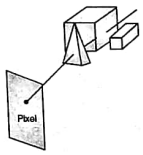


Fig. 9.2.9 : A ray along the line of sight from a pixel position through a scene

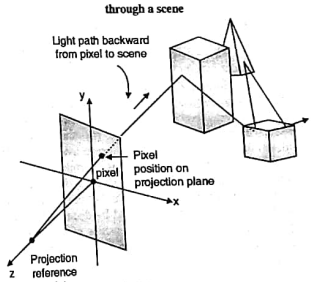


Fig. 9.2.10 : Bouncing of ray around the scene

- As shown in Fig. 9.2.10, usually pixel positions are designated in the xy plane and projection reference point lie on the z axis, i.e. the pixel screen area is centered on viewing co-ordinate origin. With this co-ordinate system the contributions to a pixel is determined by tracing a light path backward from the pixel to the picture.

- For each pixel ray, each surface is tested in the picture to determine if it is intersected by the ray. If surface is intersected, the distance from the pixel to the surface intersection point is calculated. The smallest calculated intersection distance identifies the visible surface for that pixel. Once the visible surface is identified the ray is reflected off the visible surface along a specular path where the angle reflection equals angle of incidence. If the surface is transparent, the ray is passed through the surface in the refraction direction. The ray reflected from the visible surface or passed through the transparent surface in the refraction direction is called 'secondary ray', The ray after reflection or reflection strikes another visible surface. This process is repeated recursively to produce the next generations of reflection and refraction paths. These paths are represented by 'ray tracing tree' as shown in Fig. 9.2.11.

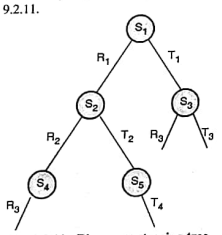


Fig. 9.2.11 : Binary ray tracing tree

- As shown in Fig. 9.2.11, the left branches in the binary ray tracing tree are used to represent reflection paths, and right branches are used to represent transmission path. The recursion depth for ray tracing tree is determined by the amount of storage available, or by the user. The ray path is terminated when predetermined depth is reached or if ray strikes a light source. As we go from top to bottom of the tree, surface intensities are attenuated by distance from the parent surface. The surface intensities of all the nodes are added traversing the ray tree from bottom to top to determine the intensity of the pixel.
- If pixel ray does not intersect to any surface then the intensity value of the background is assigned the intensity of the source, although light the pixel. If a pixel ray intersects a non reflecting light source, the pixel can be assigned the intensity of the source, although light sources are usually placed beyond the path of the initial rays.
- The Fig. 9.2.12 shows a surface intersected by a ray and unit vectors needed for the reflected light intensity calculations. Here, u is the unit vector in the direction of the ray path, N is the unit surface normal, R is the unit reflection vector, L is the unit vector pointing to the light source, and H is the unit vector halfway between V (viewer) and L (light source).

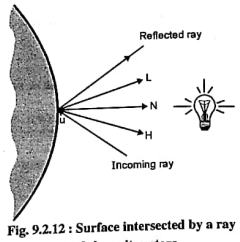


Fig. 9.2.12 : Surface intersected by a ray and the unit vectors

- If any object intersects the path along L between the surface and the point light sources the surface is in shadow with respect to that source. Hence a path along L is referred to as 'shadow ray'. Ambient light at the surface is given as $K_a I_a$, diffuse reflection due to the surface is proportional to K_d (N.L), and the specular-reflection component is proportional to K_s (H. N)^{ns}. We know that the specular reflection direction for R depends on the surface normal and the incoming ray direction. It is given as,

$$R = u - (2u \cdot N)\, N$$

- In a transparent material light passes through the material and we have to calculate intensity contributions from light transmitted through the material. Referring the Fig. 9.2.12, we can obtain the unit transmission vector from vectors u and N as

$$T = \frac{\eta_i}{\eta_r} u - (\cos\theta_r - \frac{\eta_i}{\eta_r} \cos\theta_i)\, N$$

- Where η_i and η_r, are the indices of reflection in the incident material and the refracting material, respectively then the angle of refraction is θ_r given by Snell's Law

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r}\right)^2 (1 - \cos^2\theta_i)}$$

9.3 Colors

- **Light** is a narrow frequency band within the electromagnetic spectrum, and each frequency value within the visible band corresponds to a separate color. As light is an electromagnetic wave, we can use terms frequency (V) and wavelength (λ) to explain different colors. Wavelength (λ) is the distance between crests of the wave. See Fig. 9.3.1.

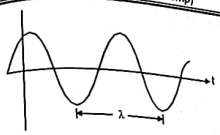


Fig. 9.3.1

- The wavelength and frequency are related properties as light moves at constant speed C.

$$\therefore\ C = \lambda \cdot V$$

- Frequency is constant for all materials, but the speed of light and the wavelength are material dependent. Light at the red end of the spectrum has a wavelength of 700 nanometers (nm) and for violet it is 400 nm. There are various characteristics of light other than frequency. Our eyes respond to the color and two more characteristics, when we view a source of light. These characteristics are brightness and purity. Brightness is the perceived intensity of the light and purity describes how 'pure' the color of light appears.
- With the help of two different color light sources, with appropriate intensities, we can generate wide range of other colors. If the two color sources combine to produce white light, they are referred as complementary colors. Complementary color pairs could be red and cyan, blue and yellow or green and magenta. When the two or three colors are used to produce other colors then they are called as primary colors in that color model. A diagram was developed by the Commission International L'Eclairage (CIE) which graphically points the eyes response to colors. See Fig. 9.3.2.

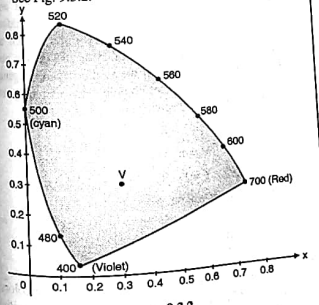
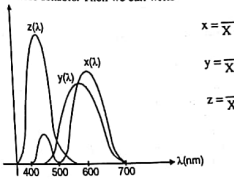


Fig. 9.3.2

- In the Fig. 9.3.2 every point represents some color. The diagram is designed in such a way that all colors on a line between two color points may be produced by mixing the light of the end point colors. Points along the curve are the pure colors in the electromagnetic spectrum, labeled according to wavelength in nanometers from the red end to the violet end of the spectrum. White light position in the diagram is represented by point V. Imagine that we have sensors for measuring light and those sensors have response curves with peaks at three specially chosen primary colors as shown in Fig. 9.3.3.
- Let X, Y and Z measure the absolute response from these sensors. Then we can write



$$x = \frac{X}{X+Y+Z},\quad y = \frac{Y}{X+Y+Z},\quad z = \frac{Z}{X+Y+Z}$$

Fig. 9.3.3

- But in CIE diagram we are showing only x and y values. It means for z value we have to say

$$z = 1 - x - y$$

- Here parameters x and y are called chromaticity values because they depend only on hue and purity. Also, if we specify colors only with x and y values, we cannot obtain the amounts x, y and z. So to convert from chromaticity co-ordinates (x, y) back to a specific color in the XYZ color space. We need an additional piece of information.

$$X = \frac{x}{y} \cdot Y;\qquad Y = Y;\qquad Z = \frac{z}{y} \cdot Y$$

where $z = 1 - x - y$.

- Therefore by using chromatically co-ordinates (x, y), we can represent all colors on a 2D diagram.

9.4 Color Models

→ (Dec. 2015)

Q. Explain the various color models in detail. MU - Dec. 2015. 5 Marks

Q. Write short note on : Color Models. (5 Marks)

- A color model is a method for explaining the properties or behavior of color within some particular context. No single color model can explain all aspects of color, so we make use of different models to describe the different perceived characteristics of color.

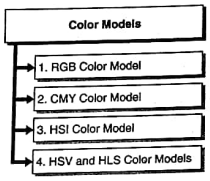


Fig. C9.1 : Color models

→ 9.4.1 RGB Color Model → (May 2013)

Q. Describe RGB color model. **MU - May 2013, 4 Marks**

- The **Red-Green-Blue (RGB)** model is generally used in computer graphics. It corresponds to Red, Green and Blue intensity settings of a color monitor. We can represent this model with the unit cube defined on R, G and B axes. The origin represents black and the vertex with co-ordinates (1, 1, 1) is white. See Fig. 9.4.1.

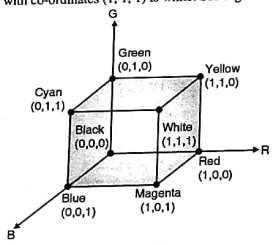


Fig. 9.4.1

- RGB color model is just an extension of XYZ model. Other colors are generated by adding intensities of primary colors, such as yellow (1,1,0) is a combination of Red and Green. Each color is represented by a triple (R, G, B). Chromacity co-ordinates for standard color television i.e. NTSC standard and CIE RGB color model are represented in tabular form.

	CIE model	NTSC/Stand Television
R	(0.734, 0.265)	(0.670, 0.330)
G	(0.273, 0.717)	(0.210, 0.710)
B	(0.166, 0.008)	(0.140, 0.080)

→ 9.4.2 CMY Color Model → (May 2013)

Q. Explain CMY color models. **MU - May 2013, 5 Marks**

- Whenever the primary colors are **Cyan, Magenta and Yellow (CMY)** in a color model, then it will be useful for describing color output on printer. These three colors are placed on white paper, when we want to draw any object. As we know that cyan can be formed by adding green and blue color. It means there is no red component, i.e. Red color is absorbed or subtracted by the ink. Similarly magenta absorbs green and yellow removes the blue colors. So by using this complementary relationship property between Red, Green, Blue i.e. (RGB) and Cyan, Magenta, Yellow i.e. (CMY) makes it easy to convert from one color model to the other.

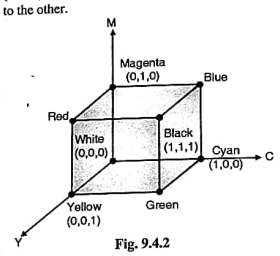


Fig. 9.4.2

- Above Fig. 9.4.2 shows a unit cube representation for CMY model. Here white color is represented by origin and black is represented by (1, 1, 1). Here principle axes (x, y, z) are mapped with cyan, magenta and yellow colors. Other colors are generated, by adding intensities of these (CMY) primary colors, such as green will be the combination of cyan and yellow and is represented as (1, 0, 1). If we assume cyan, magenta and yellow colors as exact complements of red, green and blue then we can represent this relationship as

$$C = 1 - R,$$
$$M = 1 - G,$$
$$Y = 1 - B$$

In matrix form it will be

$$\begin{vmatrix} C \\ M \\ Y \end{vmatrix} = \begin{vmatrix} 1 \\ 1 \\ 1 \end{vmatrix} - \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

- Usually printers with CMY model are using four colors i.e. Cyan, Magenta, Yellow and Black. Theoretically mixing all three i.e. cyan, magenta, and yellow should produce black, but it produces dark gray instead of black. So the fourth color is used separately as black.

→ 9.4.3 HSI Color Model

- Another color model is called **HSI.**
- **Where H = Hue, S = Saturation and I = Intensity.**
- Here hue indicates pure color of light. Saturation indicates how much white light is mixed with color and the intensity tells about the shade. Hue, saturation and intensity are often plotted in cylindrical co-ordinates with hue by angle, saturation by radius and intensity by the axis.
- We have to perform a two step procedure for conversion of RGB to HSI. In the first step we have to rotate the co-ordinates, so that the third axis is along G = R, B = R line. In second step we have to change to the cylindrical co-ordinates and proper scaling of intensity. The co-ordinates of the rectangular HSI space will be labeled as P_1, P_2 and I_1 where P_1 and P_2 indicates hue - saturation plane and I_1 indicates the intensity. So the transformation will be

$$\begin{vmatrix} P_1 \\ P_2 \\ I_1 \end{vmatrix} = \begin{vmatrix} 2/\sqrt{6} & -1/\sqrt{6} & -1/\sqrt{6} \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix}$$

→ 9.4.4 HSV and HLS Color Models

Q. Describe HSV color model. **(4 Marks)**

- Whenever **Hue (H), Saturation (S) and Value (V)** are used to indicate colors at that time we are calling that model as HSV color model. Here the meaning of H and S are same as that of HSI model. Here new term value is introduced. Value is the intensity of the maximum of the red, blue and green components of the color. Value is easier for calculation. For that the hue can be approximated by the distance along the edge of hexagon.
- The HSV color model representation is derived from the RGB color model and is shown in Fig. 9.4.3.

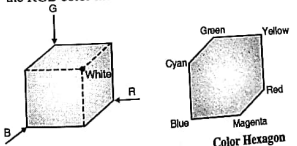


Fig. 9.4.3

- View the cube along its sides, giving as a shape of hexagon. This boundary represents the various hues and used as a top of HSV hexagon. See Fig. 9.4.4.
- In the Hexagon the saturation is measured along the horizontal axis and value is along the vertical axis through the centre of hexagon. Hue is represented as an angle about the vertical axis from 0° at red through 360°. As it is hexagon the vertices are separated at 60° intervals. Saturation (S) varies from 0 to 1. S for this model is the ratio of purity of a selected hue to its maximum purity at 1. The value (V) also varies from 0 to 1, when v = 0 it indicates black and white when, V = 1. At the top of hexagon, the colors have the maximum intensity.

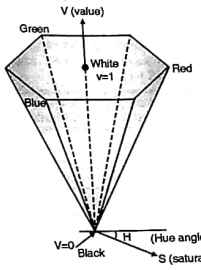


Fig. 9.4.4

- Hue selection is done by hue angle H with V = S = 1. The user describing the color desired in terms of adding either white or black to the pure hue.
- A variation of the HSV model is called as **HLS model** where L stands for lightness and H, S are representing hue and saturation respectively. The HLS model is generated by deforming the hexcone of HSV model by stretching the white point along the value axis. It gives a look as if two hexcones are placed back to back; with black at one apex and white at other. See Fig. 9.4.5.

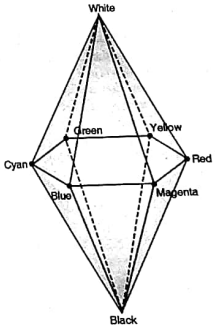


Fig. 9.4.5

- In HLS model, white has a lightness of 1 and black has a lightness of 0, but fully saturated colors have lightness value of 0.5 only. The average of the maximum and minimum RGB values is called as **lightness**.
- The HSI, HSV and HLS models are useful in implementing programs where the user must select colors. It is not always easy to tell which RGB values are needed to get a desired color. We might allow the user to specify a color using HSV model and then convert it to RGB for display.

Syllabus Topic : Polygon rendering – Constant Shading, Gouraud Shading, Phong Shading

9.5 Shading Methods

Q. Write short note on : Shading algorithms. **(5 Marks)**

- By various ways one can apply intensity models to the surface shading, depending on object surface and application requirement. Each polygon can be rendered with single intensity or by using interpolation scheme, the intensity can be obtained at each point of the surface.

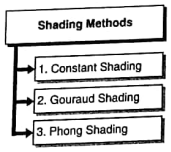
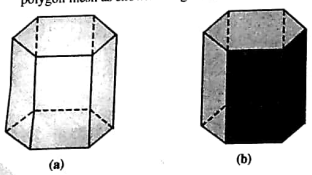


Fig. C9.2 : Shading Methods

➔ 9.5.1 Constant Shading

- **Constant shading** is also called as **Flat shading**. In this type of shading method, for each polygon, a single intensity is calculated. Then with the same intensity value all points on that surface of the polygon are displayed. This method is useful when we want to display the appearance of curved surfaces quickly. For example, a cylindrical object may be modeled by a polygon mesh as shown in Fig. 9.5.1.



(a) (b)

Fig. 9.5.1

- Constant shading can produce good results for dull polyhedrons lit by light sources that are relatively far away. The main disadvantage of this method is that false contours appear between adjacent polygons that approximate a curved surface. Refer Fig. 9.5.1(b). Also, the specular reflection of the light source tends to get lost. On the other hand, if a selected surface point happens to be at the location of the strongest specular reflection of the light source, then the color of the corresponding polygon will be significantly distorted.

➔ 9.5.2 Gouraud Shading

➔ (May 2013, May 2014, Dec. 2014, Dec. 2015, May 2016, Dec. 2016)

Q. Write short note on : Gouraud Shading. **MU - Dec. 2015, May 2016, 5 Marks**

Q. Explain Gouraud shading with its advantages and disadvantages. **MU - May 2013, May 2014, Dec. 2014, Dec. 2015, 5 Marks**

- **Gouraud shading** is an example of intensity interpolation of method. Here polygon surfaces are rendered by linearly interpolating intensity values. This method removes intensity discontinuities between adjacent planes of a surface representation by linearly varying the intensity over each plane. Here intensity values are matched at plane boundaries.
- In Gouraud shading all polygon surfaces are accessed by performing following calculations :
  - o At each vertex of polygon, determine average unit normal vector.
  - o Calculate vertex intensity.
  - o Linearly interpolate that vertex intensity over the surface of polygon.
- To determine a normal vector N at vertex V in Fig. 9.5.2, we use the average of the normal vectors of the polygons that meet at vertex V.

$$N = N_1 + N_2 + N_3 + N_4 + N_5$$

where N is then normalized by dividing it by 5.

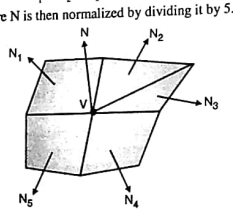


Fig. 9.5.2

- Fig. 9.5.3 demonstrates the interpolation scheme. This shading model determines intensity values at the

vertices of each polygon. All other intensities for the surface are then calculated from these values.

- Referring Fig. 9.5.3, a scan line is getting intersected with edge AB at point P. To find the intensity of this intersection point P we are interpolating between the intensities of point A and point B. Let's call intensity of point A as I_A and for B as I_B. Now the intensity of point P, which we call as I_p will be calculated as,

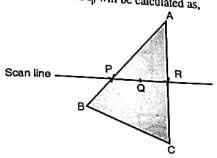


Fig. 9.5.3

$$I_p = \frac{y_p - y_R}{y_A - y_B} I_A + \frac{y_A - y_p}{y_A - y_B} I_B$$

- By using same procedure we can find the intensity of point R, which will be interpolated from points A and C. Once we know the intensities of P and R point then we can easily find intensities of interior points of the line PR. Let's call intensity for point Q as I_Q.

$$I_Q = \frac{X_R - X_Q}{X_R - X_P} I_P + \frac{X_Q - X_P}{X_R - X_P} I_R$$

- This process is repeated for each scan line passing through polygon. By combining this method with hidden surface algorithm we can fill the visible polygons along each scan line.

Advantage

Removal of discontinuities associated with the constant shading model.

Disadvantage

Highlighted surfaces are sometimes displayed with anomalous shape and the linear intensity interpolation can use bright or dark intensity strips. This effect can be reduced by using Phong shading method.

➔ 9.5.3 Phong Shading

➔ (May 2013, May 2014, Dec. 2014, Dec. 2015, May 2016, Dec. 2016)

Q. Explain Phong shading with its advantages and disadvantages. **MU - May 2013, May 2014, Dec. 2014, Dec. 2016, 5 Marks**

Q. What is Phong's Shading Model ? **MU - Dec. 2015, May 2016, 5 Marks**

- This method involves approximation of the surface normal at each point along a scan line, then calculating the intensity using the approximated normal vector at that point, displaying more realistic highlights on a surface. Phong shading method performs following steps to render a polygon surface.
  - o At each vertex of polygon, determine average unit normal vector.
  - o Linearly interpolate the vertex normal.
  - o Calculate the pixel intensity of each scan line.
- This method interpolates the normal vectors at the bounding points along a scan line. See Fig. 9.5.4.
- Refer Fig. 9.5.4, to find the normal vector N_Q at point Q in a triangle ABC, we first interpolate N_1 and N_2 to get Np. Then we interpolate N_2 and N_3 to get N_R. And finally we interpolate N_P and N_R to get N_Q.

Disadvantage

- This technique is relatively time-consuming since the illumination model is evaluated at every point using interpolated normal vectors.

Advantage

- But it is very effective in dealing with specular highlights.

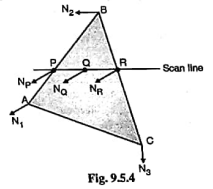


Fig. 9.5.4

9.5.4 Comparison of Phong and Gouraud Shading Algorithms

➔ (Dec. 2016)

Q. Compare Phong Shading and Gouraud Shading. **MU - Dec. 2016, 5 Marks**

Sr. No.	Phong shading	Gouraud shading
1.	It displays more realistic highlights on a surface.	It removes the intensity discontinuities exist in constant shading model.
2.	It greatly reduces the Mach - band effect.	It can be combined with a hidden surface algorithm to fill in the visible polygons along each scan line.
3.	It gives more accurate results	Highlights on the surface are sometimes displayed with anomalous shapes.

Sr. No.	Phong shading	Gouraud shading
4.	It requires more calculations and greatly increases the cost of shading steeply	The linear intensity interpolation can result bright or dark intensity streaks to appear on the surface. These bright or dark intensity streaks are called 'Mach bands.' The Mach band effect can be reduced by breaking the surface into a greater number of smaller polygons.

Syllabus Topic : Halftone and Dithering Techniques

9.6 Halftone

→ **(May 2014, Dec. 2014, Dec. 2015, Dec. 2016, May 2017)**

Q. Write short note on : Half Toning.
MU - May 2014, Dec. 2014, Dec. 2015, Dec. 2016, May 2017, 5 Marks

- **Halftone** is the reprographic technique that simulates continuous tone imagery through the use of dots, varying either in size or in spacing. 'Halftone' can also be used to refer specifically to the image that is produced by this process.
- Where continuous tone imagery contains an infinite range of colors or grays, the halftone process reduces visual reproductions to a binary image that is printed with only one color of ink. This binary reproduction relies on a basic optical illusion - that these tiny halftone dots are blended into smooth tones by the human eye. At a microscopic level, developed black and white photographic film also consists of only two colors, and not an infinite range of continuous tones.
- **Halftoning** is the process of turning continuous tone grayscale or color images into a series of dots for printing that fool the eye.
- Each intensity position in the original scene is replaced with a rectangular pixel grid.
- This method is used for printing to reproduce photographs in magazines, books or newspaper.
- For bi-level systems one can represent shading intensities with a halftoning method that converts the intensity of each point on a surface into a regular pixel grid that can display a number of intensity level. The number of intensity level with this method depends on how many pixels we include on the grid.
- A graphics package employing a halftone technique displays a scene by replacing each position in the original scene with an (n × n) grid of pixels. Intensity level of the corresponding position in the scene determines number of pixels that are turned on. Such a technique of turns on two level systems into one with the five possible intensities is shown in Fig. 9.6.1. These levels are labelled 0 through 4.

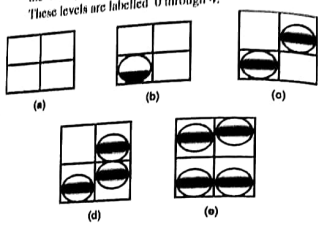


Fig. 9.6.1

- With this technique one can obtain n2 intensity levels above zero for any (n × n) grid. To avoid unwanted pattern introduction into the original possible, symmetrical pixel arrangements have to be avoided whenever possible.
- A symmetrical pattern would produce either vertical and horizontal or diagonal streaks in a halftoning representation. To avoid such patterns, select different pixel arrangements for the representation of an intensity level. Fig. 'c' can be selected to represent the second intensity level above zero.
- Another method is to form successive grid patterns with the same pixels turned on i.e. if the pixel is on for one grid level, it is on for all higher levels.

9.6.1 Thresholding

- **Thresholding** is the simplest method of image **segmentation**. From a **grayscale** image, thresholding can be used to create **binary images**. Patterning results in the loss of spatial resolution. This is acceptable if the image is of lower resolution than the display. Techniques for improving visual resolution while maintaining spatial resolution have also been developed. The simplest is to use a fixed threshold for each pixel.
- If the image intensity exceeds some threshold value the pixel is white, otherwise it is black. It means if I(x,y) > T then WHITE else BLACK. Where I(x,y) is the intensity of the image at pixel (x,y), WHITE corresponds to the maximum display intensity and BLACK corresponds to the minimum display intensity. The threshold is usually set at approximately half the maximum display intensity. Refer Fig. 9.6.2.

Original Image — Example of a threshold effect used on an image

Fig. 9.6.2

- During the threshold process, individual pixels in an image are marked as "object" pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as "background" pixels otherwise. This convention is known as *threshold above*. Variants include *threshold below*, which is opposite of threshold above; *threshold inside*, where a pixel is labeled "object" if its value is between two thresholds; and *threshold outside*, which is the opposite of threshold inside. Typically, an object pixel is given a value of "1" while a background pixel is given a value of "0." Finally, a binary image is created by coloring each pixel white or black, depending on a pixel's label.
- **Threshold** is called **adaptive threshold** when a different threshold is used for different regions in the image. This may also be known as *local* or *dynamic* threshold.
- The key parameter in the thresholding process is the choice of the threshold value (or *values*, as mentioned earlier). Several different methods for choosing a threshold exist; users can manually choose a threshold value, or a thresholding algorithm can compute a value automatically, which is known as **automatic thresholding.**
- A simple method would be to choose the mean or median value, the rationale being that if the object pixels are brighter than the background, they should also be brighter than the average. In a noiseless image with uniform background and object values, the mean or median will work well as the threshold, however, this will generally not be the case.
- A more sophisticated approach might be to create a histogram of the image pixel intensities and use the valley point as the threshold. The histogram approach assumes that there is some average value for the background and object pixels, but that the actual pixel values have some variation around these average values.
- However, this may be computationally expensive, and image histograms may not have clearly defined valley points, often making the selection of an accurate threshold difficult. One method that is relatively simple, does not require much specific knowledge of the image, and is robust against image noise, is the following iterative method :
  1. An initial threshold (T) is chosen; this can be done randomly or according to any other method desired.
  2. The image is segmented into object and background pixels as described above, creating two sets :
     (i) G1 = {f(m,n):f(m,n)>T} (object pixels)
     (ii) G2 = {f(m,n):f(m,n)≤T} (background pixels) (note, f(m,n) is the value of the pixel located in the mth column, nth row)
  3. The average of each set is computed.
     (i) m_1 = average value of G_1
     (ii) m_2 = average value of G_2
  4. A new threshold is created that is the average of m_1 and m_2
     (i) $T' = (m_1 + m_2)/2$
  5. Go back to step two, now using the new threshold computed in step four, keep repeating until the new threshold matches the one before it
- This iterative algorithm is a special one-dimensional case of the k-means clustering algorithm, which has been proven to converge at a *local* minimum meaning that a different initial threshold *may* give a different final result.

9.6.2 Dithering Techniques

→ **(May 2014, Dec. 2014, Dec. 2015, May 2016, Dec. 2016, May 2017)**

Q. Write short note on : Dithering Techniques.
MU - May 2014, Dec. 2014, Dec. 2015, May 2016, Dec. 2016, May 2017, 5 Marks

- Dither is an intentionally applied form of noise, used to randomize quantization error, thereby preventing large-scale patterns such as "banding" in images, or noise at discrete frequencies in an audio recording, that are more objectionable than uncorrelated noise. Dither is routinely used in processing of both digital audio and digital video data, and is often one of the last stages of audio production to CD.
- **Ordered dithering** is an image dithering algorithm. It is commonly used by programs that need to provide continuous image of higher colors on a display of less color depth. For example, Microsoft Windows uses it in 16-color graphics modes. It is easily distinguished by its noticeable crosshatch patterns.
- The ordered dither technique for bi-level displays also increases the visual resolution without reducing the spatial resolution, by introducing a random error into the image. This random error is added to the image intensity of each pixel before comparison with the selected threshold value. Adding a completely random

error does not yield an optimum result. Because the error pattern is small in size than image it is tiled across the image. This technique is a form of dispersed dot-ordered dither.

- Dither should be added to any low-amplitude or highly-periodic signal before any quantization or re-quantization process, in order to de-correlate the quantization noise with the input signal and to prevent distortion; the lesser the bit depth, the greater the dither must be. The results of the process still yield distortion, but the distortion is of a random nature so its result is effectively noise. Any bit-reduction process should add dither to the waveform before the reduction is performed.

Types of Dither

- There are different types of dither which are as follows :

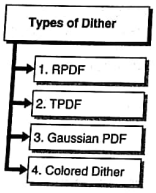


Fig. C9.3 : Types of Dither

1. RPDF

RPDF stands for **"Rectangular Probability Density Function,"** equivalent to a roll of a dice. Any number has the same random probability of surfacing.

2. TPDF

TPDF stands for **"Triangular Probability Density Function,"** equivalent to a roll of two dice (the sum of two independent samples of RPDF).

3. Gaussian PDF

Gaussian PDF is equivalent to a roll of a large number of dice. The relationship of probability of results follows a bell-shaped, or Gaussian curve, typical of dither generated by analog sources such as microphone preamplifiers. If the bit depth of a recording is sufficiently great, that noise will be sufficient to dither the recording.

4. Colored Dither

- **Colored Dither** is sometimes mentioned as dither that has been filtered to be different from white noise. Some dither algorithms use noise that has more energy in the higher frequencies so as to lower the energy in the critical audio band.
- Dithering is the attempt by a computer program to approximate a color from a mixture of other colors when the required color is not available. For example, dithering occurs when a color is specified for a Web page that a browser on a particular operating system can't support.
- The browser will then attempt to replace the requested color with an approximation composed of two or more other colors it can produce. The result may or may not be acceptable to the graphic designer. It may also appear somewhat grainy since it's composed of different pixel intensities rather than a single intensity over the colored space. Dithering also occurs when a display monitor attempts to display images specified with more colors than the monitor is equipped to handle.

9.7 Polygon Rendering

- **Rendering** is the process of generating an image from a model, by means of computer programs. The model is a description of three-dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image. Rendering is also used to describe the process of calculating effects in a video editing file to produce final video output.
- It is one of the major sub-topics of 3D computer graphics, and in practice always connected to the others. In the graphics pipeline, it is the last major step, giving the final appearance to the models and animation.
- In the case of 3D graphics, rendering may be done slowly, as in pre-rendering, or in real time. Pre-rendering is a computationally intensive process that is typically used for movie creation, while real-time rendering is often done for 3D video games which rely on the use of graphics cards with 3D hardware accelerators. Many rendering algorithms have been researched, and software used for rendering may employ a number of different techniques to obtain a final image.
- Tracing every ray of light in a scene is impractical and would take an enormous amount of time. Even tracing a portion large enough to produce an image takes an inordinate amount of time if the sampling is not intelligently restricted.
- Therefore, four loose families of more-efficient light transport modeling techniques have emerged:
  1. Rasterization, including scanline rendering, geometrically projects objects in the scene to an image plane, without advanced optical effects;
  2. Ray casting considers the scene as observed from a specific point-of-view, calculating the observed image based only on geometry and very basic optical laws of reflection intensity, and perhaps using Monte Carlo techniques to reduce artifacts;
  3. Radiosity uses finite element mathematics to simulate diffuse spreading of light from surfaces;
  4. Ray tracing is similar to ray casting, but employs more advanced optical simulation, and usually uses Monte Carlo techniques to obtain more realistic results at a speed that is often orders of magnitude slower.
- Most advanced software combines two or more of the techniques to obtain good-enough results at reasonable cost.
- Another distinction is between image order algorithms, which iterate over pixels of the image plane, and object order algorithms, which iterate over objects in the scene. Generally object order is more efficient, as there are usually fewer objects in a scene than pixels.

9.7.1 Rendering Equation

- The rendering equation describes the total amount of light emitted from a point x along a particular viewing direction, given a function for incoming light and a bidirectional reflectance distribution function (BRDF.)
- In computer graphics, the **rendering equation** is an integral equation in which the equilibrium radiance leaving a point is given as the sum of emitted plus reflected radiance under a geometric optics approximation.
- The physical basis for the rendering equation is the law of conservation of energy. Assuming that L denotes radiance, we have that at each particular position and direction, the outgoing light (L_o) is the sum of the emitted light (L_e) and the reflected light. The reflected light itself is the sum of the incoming light (L_i) from all directions, multiplied by the surface reflection and cosine of the incident angle.

  The rendering equation may be written in the form

  $$L_o(x, \omega, \lambda, t) = L_e(x, \omega, \lambda, t) + \int_\Omega fr(x, \omega', \omega, \lambda, t)\, L_i(x, \omega', \lambda, t)(-\omega' \cdot n)\, d\omega'$$

  Where,
  - λ is a particular wavelength of light
  - t is time
  - Lo (x, ω, λ, t) is the total amount of light of wavelength λ directed outward along direction ω at time t, from a particular position x
  - Le (x, ω, λ, t) is emitted light
  - $\int_\Omega \ldots d\omega'$ is an integral over a hemisphere of inward directions
  - fr(x, ω', ω, λ, t) is the bidirectional reflectance distribution function, the proportion of light reflected from ω' to ω at position x, time t, and at wavelength λ.
  - Li (x, ω', λ, t) is light of wavelength λ coming inward toward x from direction ω' at time t
  - $-\omega' \cdot n$ is the attenuation of inward light due to incident angle

Important features

- **Linearity :** it is composed only of multiplications and additions, and
- **Spatial homogeneity :** It is the same in all positions and orientations. These mean a wide range of factorings and rearrangements of the equation are possible.

- Note this equation's spectral and time dependence : L_o may be sampled at or integrated over sections of the visible spectrum to obtain, for example, a trichromatic color sample. A pixel value for a single frame in an animation may be obtained by fixing t; motion blur can be produced by integrating L_o over t.
- Although the equation is very general, it does not capture every aspect of light reflection. Some missing aspects include the following :
  - Phosphorescence, which occurs when light is absorbed at one moment in time and emitted at a different time,
  - Fluorescence, where the absorbed and emitted light have different wavelengths,
  - Interference, where the wave properties of light are exhibited, and
  - Subsurface scattering, where the spatial locations for incoming and departing light are different. Surfaces rendered without accounting for subsurface scattering may appear unnaturally opaque.
- Solving the rendering equation for any given scene is the primary challenge in realistic rendering. One approach to solving the equation is based on finite element methods, leading to the radiosity algorithm. Another approach using Monte Carlo methods has led to many different algorithms including path tracing, photon mapping, and Metropolis light transport.

Important Questions

Here we are mentioning some important questions from the topic which we have seen. And also, how to write the answers for these questions. We are not writing the full answers for the questions, but we are insisting students to write the answer in his/her own words. Here we are providing guidelines for the answers. So that, with the help of those guidelines or points, student can write his own answer.

Q. 1 Explain diffuse illumination and point source illumination.

Ans. :

The answer for this question should include following points :

- What is illumination model
- Sources of light
- Explanation of diffused and point source illumination with diagrams
- Explain reflectivity.

Q. 2 What is specular reflection.

Ans. :

To write the answer of this question we have to explain following points :

- Definition of reflection.
- Angle of reflection with suitable diagram.
- Reflection on shiny and rough surfaces.

Q. 3 Explain any one color model.

Ans. :

Here we have to explain one of the color model out of RGB, CMY, HSI, HSV. To explain this color model we can take help of following points.

- Draw a diagram of color model
- Explanation of co-ordinate axes
- Black and white color co-ordinates
- Generation of other colors from primary colors
- Conversion of model from one from to another.

Q. 4 Explain Gouraud shading and phong shading.

Ans. :

The answer for this question should include following points :

- Brief explanation and drawbacks of constant shading
- Interpolating intensity scheme
- Draw diagram and explain how to fill the surface of polygon from the intensity of vertices.
- Mention equations to calculate intensity.
- State advantages and disadvantages.

9.8 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Basic Illumination Models- Diffused Reflection, Phong Specular Reflection**

Q. What do you understand by Diffuse Illumination ? *(Refer Section 9.2.2)* **(5 Marks)**

Q. What do you understand by Point Source Illumination ? *(Refer Section 9.2.3)* **(5 Marks)**

Q. Write short note on : Specular Reflection. *(Refer Section 9.2.4)* **(5 Marks)**

Q. What do you mean by specular Reflection ? *(Refer Section 9.2.4)* **(5 Marks)**

Q. Write short note on : Ray-tracing. *(Refer Section 9.2.6)* **(5 Marks)**

Q. Explain the various color models in detail. *(Refer Section 9.4)* **(Dec.2015, 5 Marks)**

Q. Write short note on :Color Models. *(Refer Section 9.4)* **(5 Marks)**

Q. Describe RGB color model. *(Refer Section 9.4.1)* **(May 2013, 4 Marks)**

Q. Explain CMY color models. *(Refer Section 9.4.2)* **(May 2013, 5 Marks)**

Q. Describe HSV color model. *(Refer Section 9.4.4)* **(4 Marks)**

☞ **Syllabus Topic : Polygon rendering – Constant Shading, Gouraud Shading, Phong Shading**

Q. Write short note on : Shading algorithms. *(Refer Section 9.5)* **(5 Marks)**

Q. Write short note on : Gouraud Shading. *(Refer Section 9.5.2)* **(Dec. 2015, May 2016, 5 Marks)**

Q. Explain Gouraud shading with its advantages and disadvantages. *(Refer Section 9.5.2)* **(May 2013, May 2014, Dec. 2014, Dec. 2015, 5 Marks)**

Q. Explain Phong shading with its advantages and disadvantages. *(Refer Section 9.5.3)* **(May 2013, May 2014, Dec. 2014, Dec. 2014, 5 Marks)**

Q. What is Phong's Shading Model ? *(Refer Section 9.5.3)* **(Dec.2015, May 2016, 5 Marks)**

Q. Compare Phong Shading and Gouraud Shading. *(Refer Section 9.5.4)* **(Dec. 2016, 5 Marks)**

☞ **Syllabus Topic : Halftone and Dithering Techniques**

Q. Write short note on: Half Toning. *(Refer Section 9.6)* **(May 2014, Dec. 2014, Dec. 2015, Dec. 2016, May 2017, 5 Marks)**

Q. Write short note on : Dithering Techniques. *(Refer Section 9.6.2)* **(May 2014, Dec. 2014, Dec. 2015, May 2016, Dec. 2016, May 2017, 5 Marks)**

□□□

List of Experiments

L1. Study and apply basic openGL functions to draw basic primitives.

Ans. :Refer sections 1.9.2, 1.9.3 and 1.9.4

L2. Implement DDA Line Drawing algorithms and Bresenham algorithm

Ans. :Refer sections 2.3.1 and 2.3.2.

L3. Implement midpoint Circle algorithm

Ans. :Refer section 2.5.1

L4. Implement midpoint Ellipse algorithm

Ans. :Refer section 2.6

L5. Implement Area Filling Algorithm: Boundary Fill, Flood Fill, Scan line Polygon Fill.

Ans. : Refer sections 3.3.1, 3.3.2 and 3.4.1

L6. Implement Curve : Bezier for n control points , B Spline (Uniform) (atleast one)

Ans. :Refer section 6.4.2

L7. Implement Fractal (Koch Curve)

Ans. :Refer section 6.5.2

L8. Character Generation : Bit Map method and Stroke Method

Ans. :Refer section 2.9, 2.3.1 and 2.3.2

L9. Implement 2D Transformations: Translation, Scaling, Rotation, Reflection, Shear.

Ans.: Refer section 4.7

L10. Implement Line Clipping Algorithm: Cohen Sutherland / Mid point subdivision

Ans. :Refer section 5.3.1(A) and 5.3.2(B)

L11. Implement polygon clipping algorithm (atleast one)

Ans. :Refer section 5.3.2(A)

L12. Program to represent a 3D object and then perform 3D transformation

Ans. :Refer section 7.2.4

L13. Program to perform projection of a 3D object on Projection Plane using parallel projection

Ans.: Refer section 7.3.2(A)

□□□