CHAPTER 1

UNIT I

Introduction to Computer Graphics

Syllabus

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

1.1 Introduction

Q. What is computer graphics?

SPPU - Dec. 19, 3 Marks

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 Old slow and time-consuming command line interfaces have been replaced by quick, fascinating Graphical User Interface (GUI).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

Technical Knowledge Publication

- To name a few are flight simulators, which are used to train pilots for extreme conditions.
- Interactive Computer Graphics is also been used in Medical, wherein its usage is valuable. Recent advances in imaging technology such as computer tomography and magnetic resonance imaging allow physicians to take 3D X-rays of the human body. It allows the physician to interpret this large volume of data in new and useful ways.
- Non Interactive Computer Graphics : It is also known as passive computer graphics. In this type user does not have any kind of control over the image. Image is just the product of static stored program and will function according to the instructions given in the program linearly. The image is completely under the control of program instructions not under the user, example : screen savers.
- To generate a realistic scene, computer graphics performs various operations on objects such as translation, rotation, scaling, reflection, shearing, hidden surface removal, texture mapping, shading.
- Operations of computer graphics can be classified as follows :
  1. **Geometry :** How to draw and represent the scene.
  2. **Animation :** How to add dynamic content in the scene.
  3. **Rendering :** How to add realistic light in the scene.
  4. **Imaging :** Image acquisition and image editing.

1.2 Graphics Primitives

- On the monitor, graphics primitives are generated using scan conversion.
- **Scan conversion** is the process of turning '*on*' or '*off*' the pixels along the boundary of the primitive shape.
- For each pixel on the screen, scan conversion algorithm performs the computation based on the initial parameters of the primitives and determines which pixel is to be turned '*on*'
- Few basic primitives are discussed in following subsections.

1.2.1 Lines

- The line is a collection of points along a straight path. Typically, the line is described by its two endpoints (x_1, y_1) and (x_2, y_2). Intermediate points on the line are calculated by line generation algorithm.

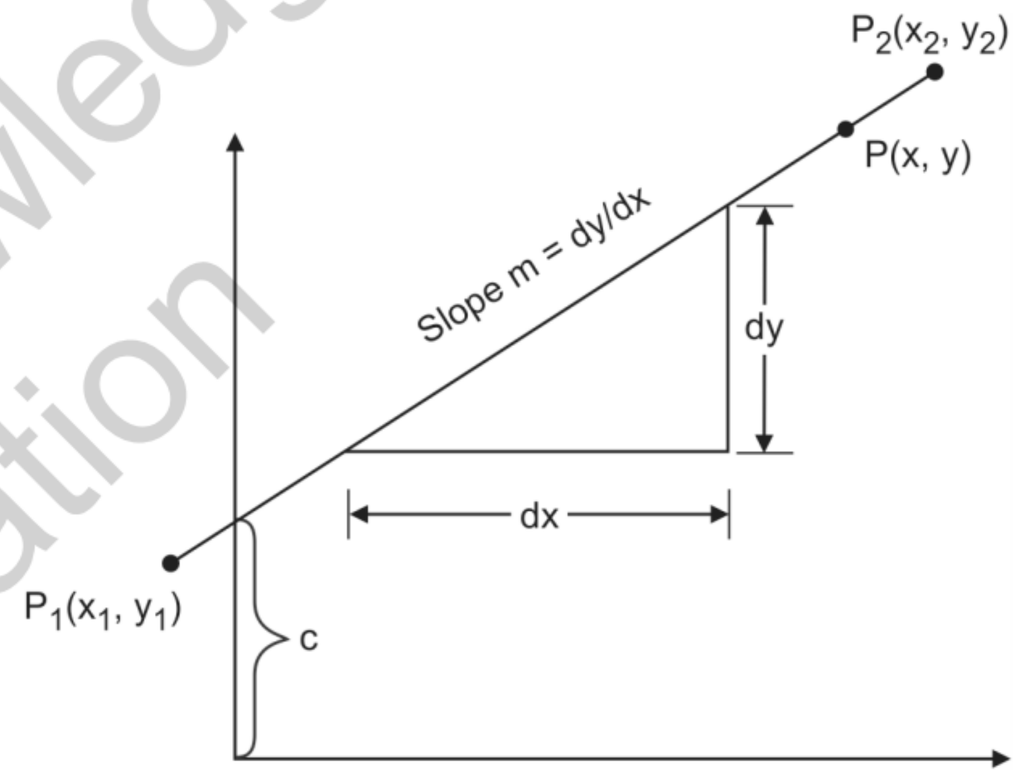


Fig. 1.2.1 : Representation of line

- Given the endpoints $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, any point P(x, y) on the line can be computed using the slope-intercept formula as follow.

  It is evident from Fig. 1.2.1 that the slope of the line is,

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

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

$$y = mx + c$$

- Lines with different slopes are shown in Fig. 1.2.2.

Tech Knowledge Publications

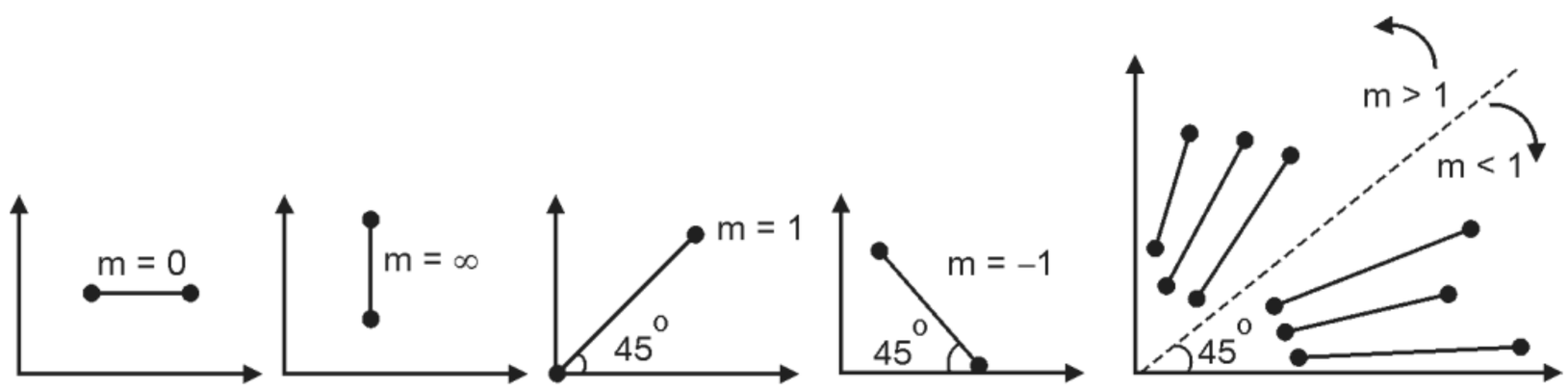


Fig. 1.2.2 : Lines with different slope

1.2.2 Line Segments

- A line segment is the part of the line. Curves are often approximated using small line segments. Almost any shape can be approximated by line segments.
- Like line, a line segment is also bounded by some start and endpoints.

1.2.3 Vectors

- In geometry, a vector is defined by two properties : direction and the magnitude.
- We can represent a vector as a directed line, which grows from tail to head.

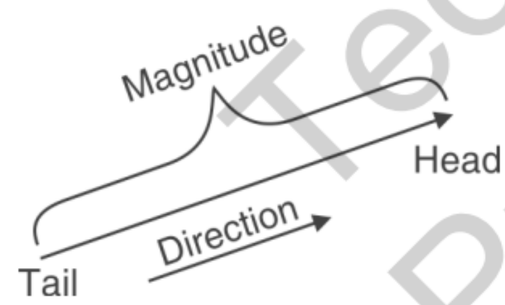


Fig. 1.2.3 : Vector representation

- Two vectors are said to be same if, they have the same direction and magnitude.
- A vector in 2D is defined as $v = ai + bj$, where (i, j) represents the direction and (a, b) represents the magnitude of a vector in x and y direction, respectively.
- Given the two points $A(x_1, y_1)$ and $B(x_2, y_2)$, the vector in the direction from A to B is defined as,

$$v = \overrightarrow{AB} = B - A$$
$$= (x_2 - x_1, y_2 - y_1)$$

- And the gradient of vector $\overrightarrow{AB}$ is given as,

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

Ex. 1.2.1 : Compute the direction and magnitude of the vector $\overrightarrow{AB}$ given the points A(2, 4) and B(7, 8).

Soln. :

Direction of the vector is computed as,

$$v = \overrightarrow{AB} = B - A$$
$$= (x_2 - x_1, y_2 - y_1)$$
$$= (7 - 2, 8 - 4) = (5, 4)$$

Magnitude of the vector is computed as,

$$||v|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$= \sqrt{5^2 + 4^2} = \sqrt{41} = 6.4$$

Thus, the vector is pointing in the direction 5i + 4j and has magnitude 6.4. The geometric representation of the points and vector is shown in following Fig P. 1.2.1.

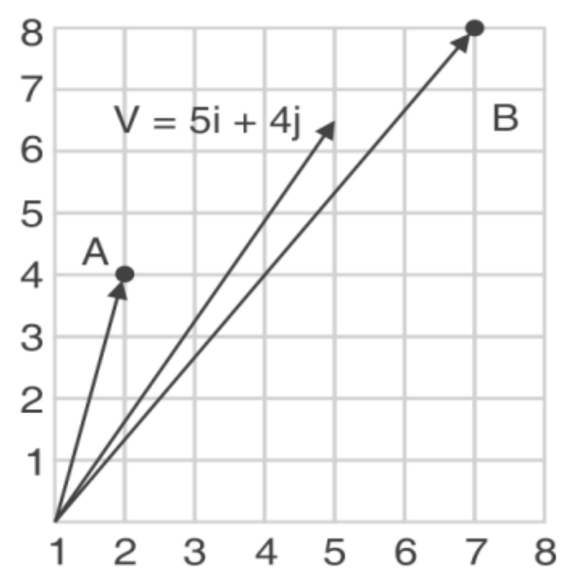


Fig. P. 1.2.1 : Geometric interpretation of vector

Technknowledge Publications

1.3 Basic Terminology

1.3.1 Pixel

Q. Define the term Pixel.

SPPU - Dec. 15, May 16, 2 Marks

- The ***pixel*** is the smallest addressable unit of an image/screen. It is also known as **pel** or **picture element**. Real-world objects are continuous in nature and hence they have infinite samples.
- When they are converted into an image and displayed on the monitor screen, only finite samples are taken into consideration. Ultimately, the object on the computer screen is a collection of pixels.

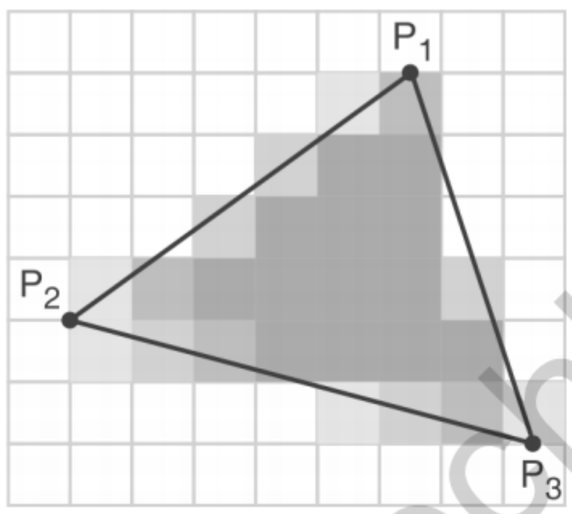


Fig. 1.3.1 : Pixel-wise representation of shape on the monitor screen

- Like the digital image, the monitor screen is also divided into a grid, consisting of 'n' rows and 'm' columns. Each cell on this grid is also referred to as a pixel. Each grey/black cell in Fig. 1.3.1 corresponds to one pixel.
- On zooming in the image, we can get the better idea of the pixel. Each square in Fig. 1.3.1 represents one pixel.
- Pixels contained within the object (triangle here) are displayed with full intensity. If pixel is partially covered by the object border, the pixel is illuminated with proportional intensity. The pixel with more area inside triangle will have higher intensity then the pixel with less area inside the triangle boundary.
- This is sufficient to clarify the difference between the object drawn on paper and displayed on monitor. Objects drawn on paper have smooth boundary whereas objects displayed on monitor have zig-zag boundary.
- Pixels outside the object will not be illuminated.
- For a grayscale image, pixel value varies from 0 to 255, where 0 represents the black color and 255 represents the white color. In between range represents the grey shades with varying intensities.
- For a color image, each pixel is a triplet of color(R, G, B) consisting proportion of red, green and blue color. The mixture of these three components determines the final color of the pixel on the monitor screen. Value of each component varies from 0 to 255, resulting in 255 × 255 × 255 i.e. approximately 17 million different shades of color.
- The header file graphics.h contains pupixel() function that plots a pixel at location(x, y) of specified color every pixel has a name or address

  Syntax :

```
void  putpixel (int x, int y, int color)
```

  Example:

```
putpixel (100, 200, RED);
```

- A pixel of red color can be drawn at position (100, 200). Putpixel function can be used to draw circles, lines and ellipse using different algorithms.

1.3.2 Resolution

Q. Write short note on resolution.

SPPU - May 15, 2 Marks

Q. Explain term resolution with suitable example.

SPPU - Dec. 15, May 19, 2 Marks

Tech Knowledge Publications

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

Fig. 1.3.2 : Images with different resolution

1.3.3 Aspect Ratio

Q. Write short note on aspect ratio.

SPPU - May 15, Dec. 15, Dec. 18, May 19, 2 Marks

- Another property of video monitor is the aspect ratio, which is the ratio of number of vertical pixels to the number of horizontal pixels required to draw the same length of a line in both the direction.
- An aspect ratio of 9/16 means that a vertical line plotted with 9 points has the same length as a horizontal line plotted with 16 points.
- Generally, the distance between two vertical pixels is more than a distance between two horizontal pixels.
- Different aspect ratios are used in different devices. For example, 20th century monitors were using aspect ratio of 4 : 3, wide screen televisions use aspect ratio of 16 : 9, ultra-wide screen devices use aspect ratio of 64 : 27.

1.3.4 Frame Buffer

Q. Define the term Frame buffer.

SPPU - Dec. 15, May 16, Dec. 18, 2 Marks

Q. Describe Frame buffer display in computer graphics. **SPPU - May 15, 4 Marks**

- Image is 2D array of intensity values. We can consider the screen picture as a two-dimensional grid of pixels.
- Raster system displays the picture by drawing pixel in a row by row from left to right. Picture definition is stored in a memory called the **refresh buffer** or **frame buffer**.
- Scene to be displayed is first loaded into a frame buffer in the form of intensity values. Ideally, the size of the frame buffer is the same as screen resolution. Intensity value from the top-left location of the frame buffer is retrieved and painted at a top-left location on the screen. The second pixel of the same row is painted soon after that and this process continues.
- A frame buffer is large Contiguous section of computer memory. There is one memory bit for each pixel. This memory is referred to bit plane. The image is built in the frame buffer one bit at a time. CRT is an analog device whereas frame buffer is a digital device, so there is a need for conversion from digital to analog signal.
- For this digital to analog converter (DAC) is used. Every pixel in the frame buffer must be read and converted prior to its visibility on the raster CRT.

Tech Knowledge Publications

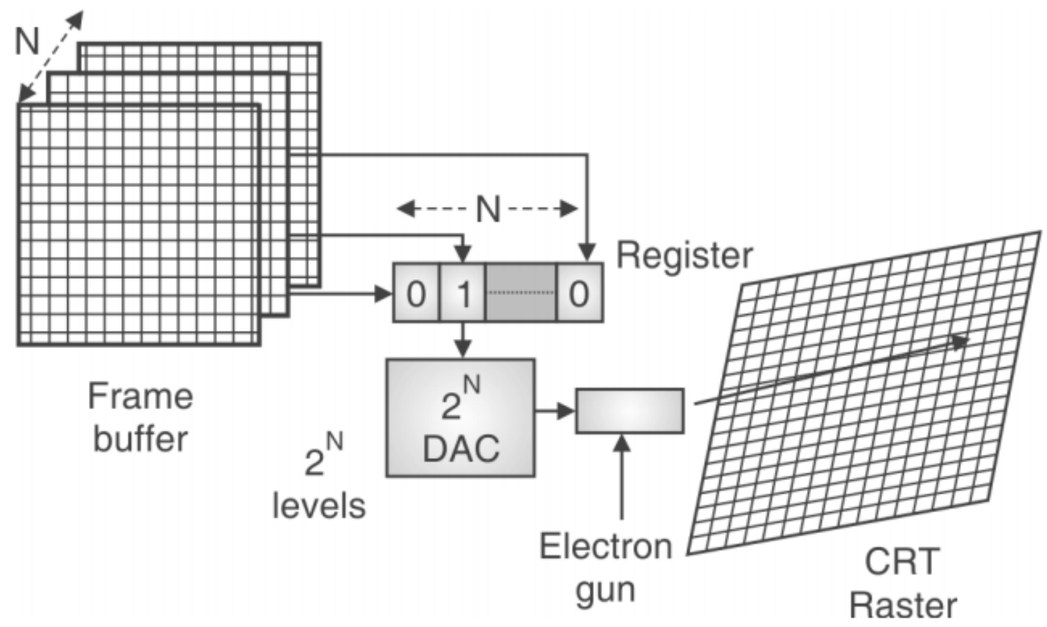


Fig. 1.3.3 : An N-bit plane gray level frame buffer

- The intensity of every pixel on the CRT is managed by a corresponding pixel location in each of the N bit planes. The binary value from each of the N bit planes is loaded into corresponding positions in a register. The resulting binary number is read as an intensity level between 0 (dark) and 2n – 1 (full intensity).
- This is converted into an analog voltage between 0 and the maximum voltage of the electron gun by the DAC. A total of 2N intensity levels are possible.
- After reading the bit planes in the frame buffer, the resulting number is used as an index into the lookup table. The look up table must contain 2N entries. Each entry in the lookup table is W bit wise. W may be greater than N. When this occurs, 2W intensities are available; but only 2N different intensities are available at a time. To get additional intensities, the lookup table must be changed.
- As there are three primary colors, a simple color frame buffer is implemented with three bit planes, one for each primary color. Each bit plane uses an individual color gun for each of the three primary colors used in color video. These three primaries (red, green, and blue) are merged at the CRT to produce eight different colors.
- Frame buffer in raster display stores individual pixels of the scene, whereas in case of random scan display, frame buffers store commands for drawing the scene.
- Suppose a system has resolution 1280 × 1280 and it supports 24 bits per pixel. The memory requirement for the frame buffer is;

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

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

= 16,38,400 × 24

= 3,93,21,600 bits

= 39321600 / 8

= 49,15,200 Bytes

(∵ 1 Byte = 8 bits)

= 49,15,200 / 1024 = 4800 KB

(∵ 1 KB = 1024 Bytes)

= 4800 / 1024 = 4.69 MB

(∵ 1 MB = 1024 KB)

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

Techknowledge Publications

1.4 Display Adapters

- Desktop computer that makes use of a plug-in card which performs graphics processing is display adapters. Most commonly referred to as "graphics card" or "video card" Display Settings.
- The following display settings determine how much information a video adapter and driver display and how often that information is updated.

1.4.1 Refresh Rate

- The number of times per second that the video card replaces the current screen image with a new image is called the refresh rate. The refresh rate of image varies with the individual, monitor size, persistence of monitor phosphors, ambient lighting, the image being displayed, and other factors, but is typically somewhere between 60 and 90 images per second. Setting refresh frequency too low results in visible flicker.
- Generally a refresh rate of 75 Hz or higher is used on 15-inch monitors having resolution 800 × 600; 75 to 85 Hz on 17-inch monitors having resolution 1024 × 768; and 85 Hz or higher on 19-inch and larger monitors having resolution 1280 × 1024 or higher.

1.4.2 Modes

There are many video adapters which support various modes of resolution. They are broadly categorized as text mode and graphics mode.

Text Mode

In this mode the display screen consists of an array of characters that holds one ASCII character.

Graphics Mode

In this mode display screen consists of an array of pixels. Number of different shapes of an object can be displayed

1.4.3 Graphics Adapters

The graphics adapters is a card used to enable the computer users to view the information in graphical form. The information is sent from the computer system to the monitor.

There are two formats for Graphics cards

1. **Dedicated video card :** It has its own graphics processor and memory. This types of cards require extra power supply and fits into a PCI slot on the motherboard. It has several different formats so the user has to ascertain the accurate video card.
2. **Built in video card :** It is built in video card on the motherboard, utilizes a part of system ram. This type is known as Graphics Processing Unit(GPU) which works in combination with CPU. Onboard video card is popularly used due to its cheap cost. Modern graphics cards are The Nvidia GeForce RTX 2070, different Radeon RX 570 models etc.

The different graphics adpters are listed below

- MDA (Monochrome Display Adapter)
- CGA (Color Graphics Adapter)
- EGA (Enhanced Graphics Adapter)
- VGA (Video Graphics Array) SVGA (Super Video Graphics Array)
- XGA (Extended Graphics Array)

MDA (Monochrome Display Adapter)

It was the first display adapter. It was designed for transistor-transistor logic(TTL). It used only text. It could not display any graphics or color. It makes use of 9 × 14 dot box character. It has become outdated.

CGA (Color Graphics Adapter)

- It was once the most commonly used color graphics adapter (CGA). It supported RGB monitors with a maximum resolution of 640 × 200 pixels. It has two modes of operation 640 × 200 pixels alphanumeric (A/N) and all point addressable (APA). It displays card displays either 40 or 80 columns with 25 lines of text.

Technknowledge Publication

– 16 colors can be displayed in the A/N mode, whereas the all points addressable (APA) addresses each pixel individually.

EGA (Enhanced Graphics Adapter)

Monitors using EGA can display 43 lines of text. It could display 16 colors with a resolution of 640 × 350 pixels (640 × 350), much finer graphics than were possible in CGA mode.

VGA (Video Graphics Array) SVGA (Super Video Graphics Array)

It supported a resolution of 720 × 400 in text mode and a resolution of 640 by 480 (640 × 480) in 16-color graphics mode. It could also display 256 colors with a definition of 320 × 200 (a mode also known as MCGA for Multi-Color Graphics Array). The VGA quickly became the baseline display mode for PCs.

XGA (Extended Graphics Array)

It was introduced by IBM Supported a resolution of 800 × 600 in 16 million colors and 1024 × 768 in 65536 colors.

1.5 Display Devices

Display devices are output devices used to display information such as the monitor, television set. The information displayed on video display devices is transient. It is displayed for some time and then erased. In following section, we will discuss few popular display devices.

1.5.1 Cathode Ray Tube

– Cathode ray tube (CRT) is a common technology for traditional monitors and television sets. Functional diagram of CRT is depicted in Fig. 1.5.1.

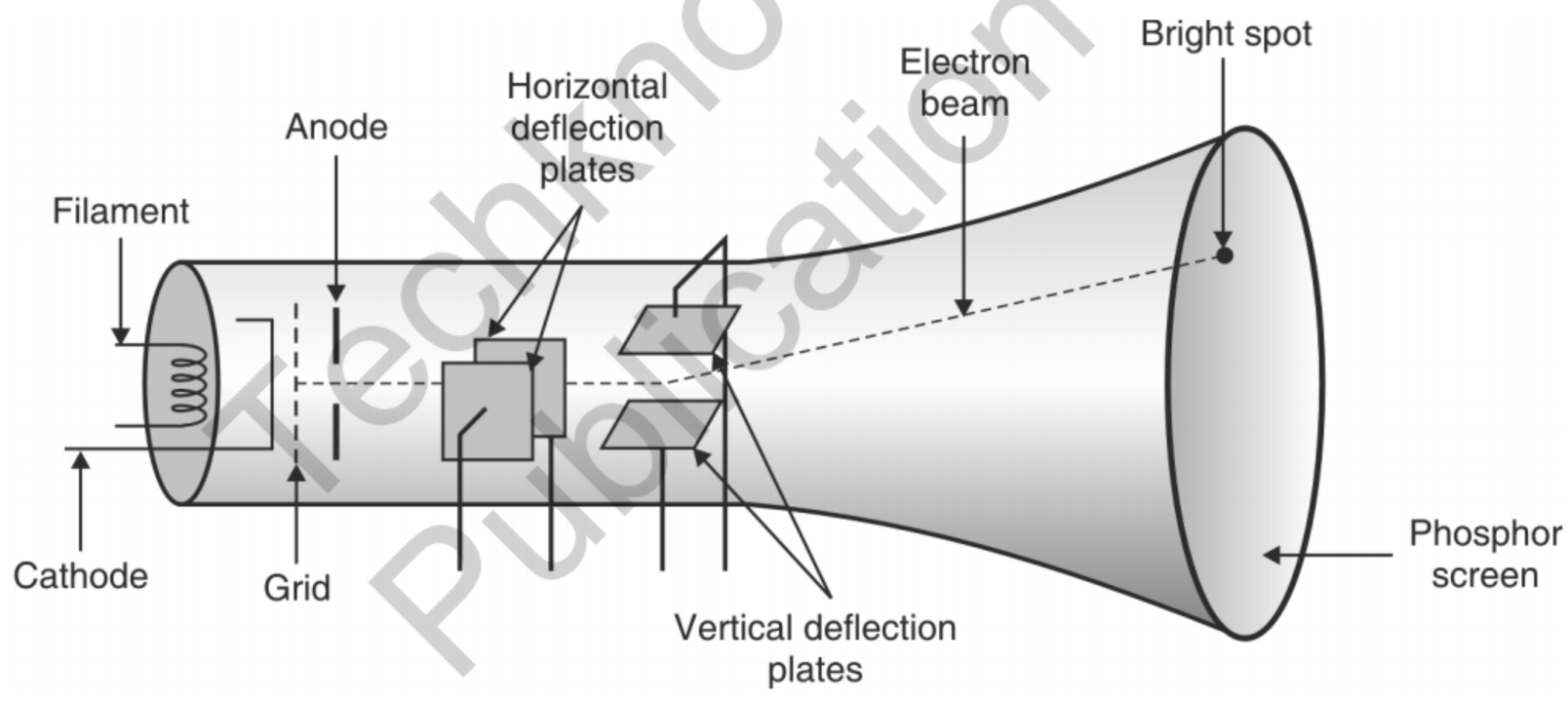


Fig. 1.5.1 : Functional diagram of CRT

– When current is supplied to the filament, electrons are produced and they get dispersed all around. Electrons generated from electron guns fly in arbitrary directions. Focusing anode focuses all dispersed electrons in a beam called an *electron beam*. For higher precision, additional focusing hardware is used to keep the beam in focus at all screen positions.

– The monitor screen is generally curvature in shape, the distance of all screen pixels is not the same from the electron beam. Electron beam focuses properly at the center of the screen. As the beam moves away from the center i.e. near to the edges, the scene on screen become blurred. To overcome this effect, the system can adjust the focusing according to the screen position of the beam.

- All those fired electrons form a beam and start flowing towards phosphorus screen, which is exactly behind the glass of the monitor screen. A positively charged anode, next to the grid, provides acceleration to these electrons. Electrons are negatively charged and an anode is positively charged, so they get attracted towards the anode and get acceleration.
- The beam of electrons in acceleration collides with phosphorus screen and their kinetic energy is absorbed by the screen. This action moves the electrons into their higher quantum energy level and electrons become unstable. As nature, electrons release the energy and try to become stable. This energy appears as an illuminated bright spot on the screen, which we call ***pixel***.
- In RGB monitors, there are three layers of phosphorus behind screen, Red, Green and Blue. So color and intensity of illuminated pixel depend on how deep electron beam has penetrated particular layer and what amount of electrons were generated from electron guns.
- When electrons become stable, they stop releasing energy and after a few milliseconds pixel goes off from the screen. To persist the scene on the screen, it is essential to keep redrawing all the pixels repeatedly on the monitor screen. This process is called ***refreshing*** and such a display system is called ***refresh CRT***.
- Apart from color, the main difference between phosphors is their persistence rate. Persistence rate defines how long pixels continue to emit light after the CRT beam is removed. The time it takes the emitted light to decay to one-tenth of its original intensity is called ***persistence*** of phosphor. If phosphor has a lower persistence rate, it requires higher refresh rates to maintain a picture on the screen without flicker.
- Animation requires quick erase and redrawing of the scene. So low presentence phosphor is preferred in such applications. High-persistence phosphor is useful for displaying highly complex and static pictures. Usually, graphics monitors use phosphorus with persistence in the range from 10 to 60 microseconds.
- Brightness or intensity of a pixel is directly proportional to the number of electrons colliding on phosphorus layers. We can control the number of electrons by controlling the width of the control grid.
- Control grid acts as a gate for electrons. On applying a high negative voltage, control grid shuts off the beam by repealing and not allowing electrons to pass through it means the gate is closed for electrons. On applying a smaller negative voltage, control grid open ups the gate and more electrons can pass through it.
- If the movement of the flowing electron beam is not controlled, it continuously keeps hitting the center pixel of the screen. The energy released by electrons cumulated at a single point, and very soon the phosphorous layer will burn out at that location.
- The position of the colliding beam is controlled either by electric fields or by magnetic fields. Cathode-ray tubes are constructed with magnetic deflection coils which are mounted outside of the CRT.
- Electrostatic deflection system uses two pairs of parallel plates mounted inside the CRT(as shown in Fig. 1.5.1). In both cases, a pair of coils are used to control the movement of the electron beam in the horizontal and vertical direction.
- One set of plate controls the horizontal deflection of the beam and other sets of plate control the vertical deflection. Using such a focusing system, we can make an electron beam to collide with each pixel location on the screen to illuminate the entire scene at a time.

Tech Knowledge Publications

– A maximum number of pixels that can be displayed without overlap on a screen is called ***screen resolution***. Typical resolution on high-quality systems is 1280 by 1024. Such high-resolution systems are often referred to as ***high-definition systems***.

– Another property of video monitor is the ***aspect ratio,*** which is the ratio of the number of vertical pixels to the number of horizontal pixels required to draw the same length of a line in both the direction. An aspect ratio of 9/16 means that a vertical line plotted with 9 points has the same length as a horizontal line plotted with 16 points. Generally, the distance between two vertical pixels is more than the distance between two horizontal pixels.

1.5.2 Raster Scan Display

– We can consider the screen picture as a two-dimensional grid of pixels.

– Raster system displays the picture by drawing pixel in a row by row from left to right. Picture definition is stored in a memory called the **refresh buffer** or **frame buffer**.

– Scene to be displayed is first loaded into a frame buffer (a reserved memory for storing the scene to be displayed on a monitor screen) in the form of intensity values.

– Ideally, the size of the frame buffer is the same as screen resolution. Intensity value from the top-left location of the frame buffer is retrieved and painted at a top-left location on the screen. The second pixel of the same row is painted soon after that and this process continues.

– After plotting all pixels in the first row, the electron beam moves to the first pixel of the second row and paints all pixels in the second row on-screen by retrieving intensity values in horizontal order. Fig. 1.5.2 describes the working principle of the raster scan system.

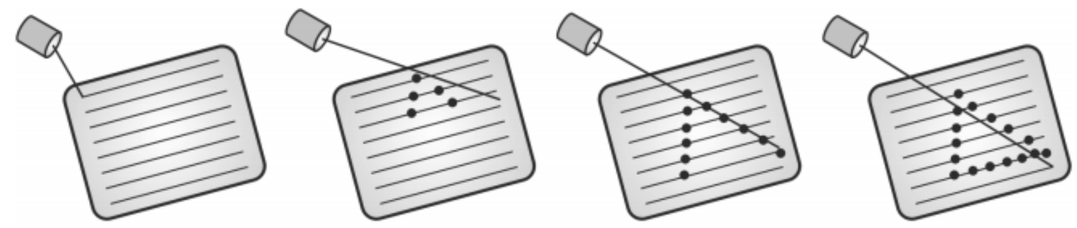


Fig. 1.5.2 : Scene rendering using a raster scan system

– After finishing each scan line (row), electron guns are turned off and moved back to the first pixel of the next scan line. That horizontal movement is called **horizontal retrace**. After reaching to the last pixel of the screen, electron guns are moved at the first pixel of the first row. That vertical movement of electron guns is called **vertical retrace**.

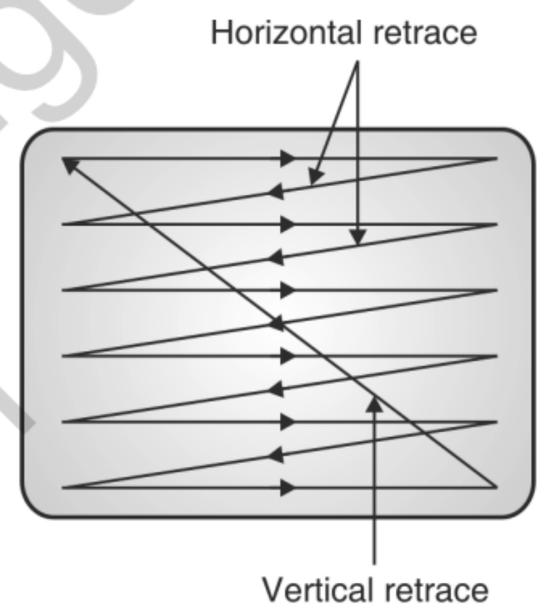


Fig. 1.5.3 : Horizontal and vertical retrace

– The downside of the raster system is that it scans the entire screen even if we want to display a few pixels, as shown in Fig. 1.5.2. It checks for each location in the frame buffer to turn *on* or *off* the electron guns.

– Pixel is the smallest unit that can be addressed and displayed. Raster scan displays are well opted for realistic scene due to its capability of storing intensity for each pixel. Intensity range for pixel depends on the capability of the raster system.

– A system which supports 1 bit per pixel can display only black and white scene, called a *bi-level system*. A bit value of 1 indicates that the electron beam should be *on* for that location, and *off* otherwise.

– With n bits per pixel, the system can show maximum 2^n different colors / intensities. Current color CRT system provides 24 bits per pixel, which supports 2^{24} (approximately 17 million) colors.

Tech Knowledge Publications

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

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

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

= 16,38,400 × 24

= 3,93,21,600 bits

= 3,93,21,600 / 8

= 49,15,200 Bytes

(∵ 1 Byte = 8 bits)

= 49,15,200 / 1024 = 4800 KB

(∵ 1 KB = 1024 Bytes)

= 4800 / 1024 = 4.69 MB

(∵ 1 MB = 1024 KB)

– Thus, the system having screen resolution 1280 × 1280 and supporting 24 bits per pixel require 4.69 MB of memory for the frame buffer.

– The frame buffer is known as **bitmap** if it uses one bit per pixel. And it is known as **pixmap** if it uses multiple bits per pixel. A bitmap is used in the bi-level system while pixmap is used in a system supporting multiple colors.

– In raster display, the screen is refreshed nearly 60 times per second to display picture without flickering. Rate of the refreshing screen in one second is called **refresh rate**. The refresh rate is measured in frames per second or Hertz. If the refresh rate is set to less than 60, the screen starts flickering. It may happen that while the bottom part of the screen is being painted, the upper part may wipe out due to a lower refresh rate.

Interlaced Display System

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

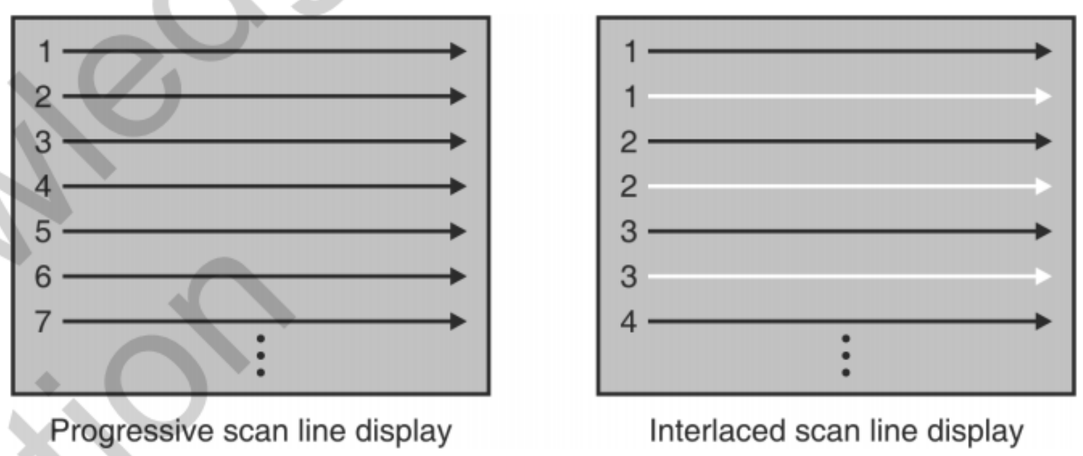
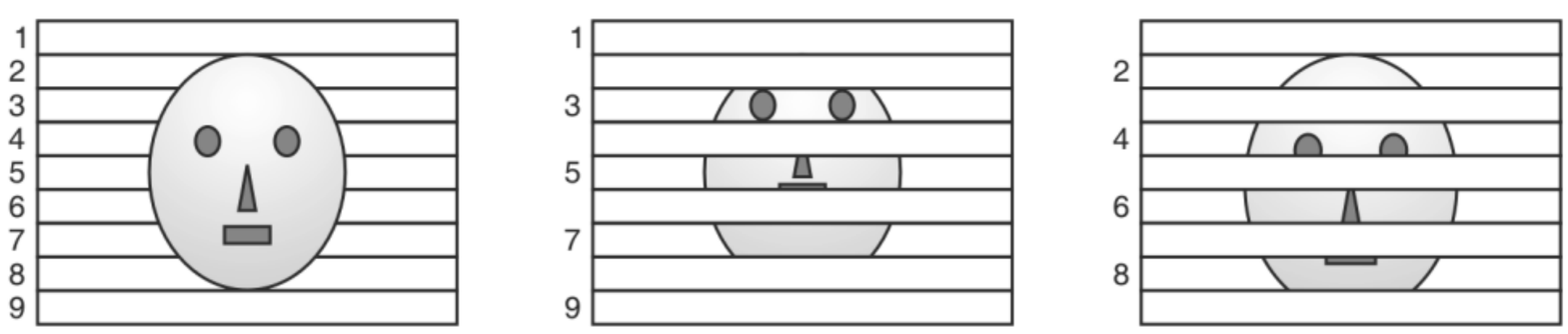


Fig. 1.5.4 : Displaying using progressive and interlaced scan line

– Each frame contains exactly two fields. ***The odd field*** is made up of the odd horizontal scan lines in a frame. The odd field is also called the ***top field*** since it contains the top line of the image. After the vertical retrace, the beam paint out the entire remaining even scan lines called ***Even field***.

– Even the field is called the ***bottom field***. Fig. 1.5.5(a) ad 1.5.5(b) shows the working of the progressive and interlaced display.



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

Fig. 1.5.5 : Progressive vs interlaced display

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

1.5.3 Random Scan Display

– In a random scan system, the electron beam directly follows the part of the screen where the picture is to be displayed. Instead of an individual pixel, the picture is stored in terms of line drawing commands, and for this reason, it is also referred to as **vector displays** or **stroke-writing** or **calligraphic displays**.

– The memory area where picture definition is stored is referred to as a **refresh display file**. Sometimes it is also called **display list, display program**, or **refresh buffer**. In random scan displays, display processor reads line drawing command one by one from the display list and draws it. Fig. 1.5.6 shows the rendering of a polygon on the screen. Electron beam only follows the border of polygon instead of scanning each scan line and deciding which pixel should be turned on or off.

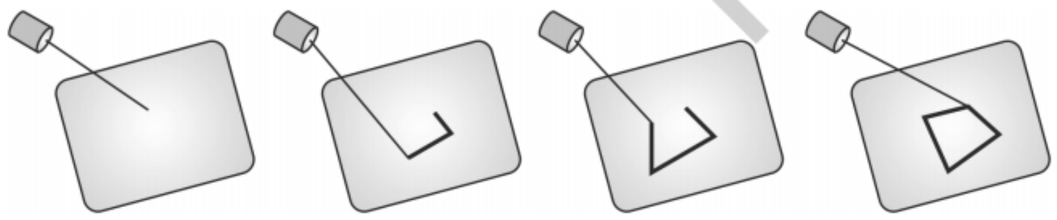


Fig. 1.5.6 : Scene rendering using random scan system

– A pen plotter is the example of a random-scan device. The refresh rate on a random-scan system depends on the number of lines to be displayed. High-quality vector systems are capable of handling near about 10^6 small lines at the refresh rate of 60 Hz. When frame buffer contains a small set of lines, each refresh cycle is delayed to avoid refresh rates greater than 60 Hz. Otherwise, faster refreshing could burn out the phosphor.

– Random-scan systems are useful for line drawing applications and are not suitable to display realistic shaded scenes. Random displays produce smooth line because the CRT beam directly follows the line path, while raster system produces jagged lines. Random scan devices have generally limited color capability.

Sr. No.	Advantages of Random Scan	Disadvantages of Random Scan
1.	Good choice for a device with very high resolution.	Cannot handle the complex or natural scene.
2.	Requires less memory.	Supports limited color only.
3.	Produces smooth lines.	Such devices are costly.
4.	Useful in displaying the static drawing.	Not useful to show animation.

1.5.4 Raster Scan Vs. Random Scan

Sr. No.	Raster Scan System	Random Scan System
1.	The electron beam scans the entire screen to draw a picture.	The electron beam scans only the part of the screen where picture information is present.
2.	Due to the discrete nature of the system, it has low resolution.	Electron beam directly follows the path of lines, it draws smooth lines, which gives better resolution.
3.	Picture definition is stored as a set of discrete intensity values in the frame buffer.	Picture definition is stored as a line drawing commands in the display list.

TechKnowledge Publications

Sr. No.	Raster Scan System	Random Scan System
4.	The intensity value is stored for each pixel, it is well suited to display realistic scene.	The system is designed to display lines, so it cannot be opted to display a realistic scene.
5.	Pixel / spatial location of the screen is used to draw an image.	Mathematical functions are used to draw an image.
6.	The refresh rate is independent of the number of objects in the scene.	When the numbers of primitives are too large, random scan device flickers.
7.	Scan conversion is required.	Scan conversion is not required.
8.	Scan conversion hardware is required as it displays the real-time dynamic scene with extensive computation.	Scan conversion hardware is not required.
9.	Such displays are economical.	They are more costly.
10.	Used to display a dynamic scene.	Used to display static picture.
11.	The video controller is required.	The video controller is not required.

1.5.5 Color CRT Monitors

As we discussed in the previous section, colors are generated by hitting the phosphor layers. Numbers of electrons and speed of electron beam decides the intensity and color of an illuminated pixel. There are two methods to produce color on the screen.

1.5.5.1 Beam Penetration Method

- This method is suitable for random scan system. This method uses two layers of phosphor, red and green.
- Color of a pixel depends on how far electron beam penetrates these phosphor layers. If an electron beam is fired with slow speed, it only penetrates the outer red phosphor layer.
- And if an electron beam is fired with high speed, it penetrates both layers and excites inner green layer. Intermediate speed produces a mixture of red and green i.e. orange and yellow. Speed of electrons is controlled by beam acceleration voltage.

Advantage

An inexpensive way to produce colors.

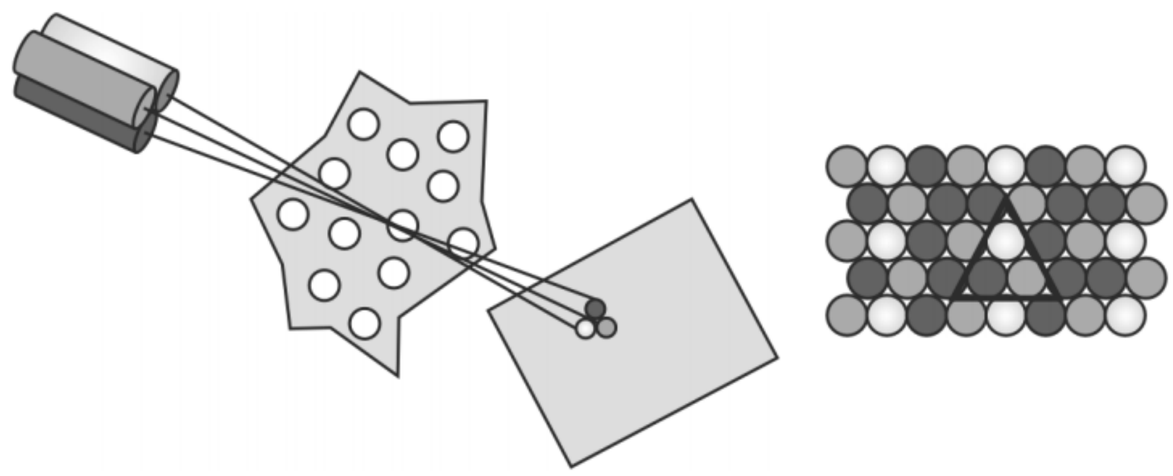
Disadvantage

- Suitable to random scan display only.
- Can produce only four colors.

1.5.5.2 Shadow Mask Method

- Beam penetration method cannot render realistic scene properly. Their application is limited to random scan devices only. Another method called shadow mask is used with raster scan devices to produce realistic scene.
- Shadow mask method provides a wider range of colors and intensities. Shadow mask CRT contains three phosphor layers, red, green and blue, each produces a dot at any pixel location. Such CRT has three electron guns, one for each color dot.
- Fig. 1.5.7 shows the delta-delta shadow-mask method. Shadow mask is a mask of holes placed between electron guns and phosphor layers. When the three beams pass through a hole in the shadow mask, they generate a triangle of three dots, which appears as a small color spot on the screen. And this is why such shadow masks are also referred to as ***delta shadow mask.***

Techknowledge Publication



(a) Shadow mask (b) Delta pattern

Fig. 1.5.7 : Shadow mask and its operation

- In the ***in-line*** shadow mask arrangement, three electron guns are aligned and placed side by side. This technique provides higher precision compare to delta shadow mask and should be used in a high resolution system. Color perceived by human eye depends on spatial integration of intensities. So we perceive the average color produced by three electron guns at any given pixel location.
- Numbers of colors on monitor depend on the number of bits allocated per pixel. With n bits per pixel, the system can support 2^n different colors. By varying the voltage levels of electron guns, we can produce different shades of colors. By turning *on* only green and red electron gun, we can produce a yellow color. Similarly magenta is the mixture of red and blue, and cyan can be generated by activating green and blue electron guns.
- In cheaper systems, the electron beam can only be turned to *on* or *off*, which limits the display to only eight colors. Whereas some sophisticated graphics systems allow several million different colors. High-quality raster-graphics systems support 24 bits per pixel, eight-bit per each color, allowing 256 shades of each color. Such a system can produce 256 × 256 × 256 (i.e. approximately 17million) colors. An RGB color system with 24 bits of storage per pixel is generally referred to as a ***full-color system*** or a ***true-color system.***

1.5.6 Direct-View Storage Tubes

Direct View Storage Tube (DVST) is an alternative way to display picture. DVST devices store the picture information inside the CRT itself instead of refreshing the screen. It stores the picture information as a charge distribution just behind the phosphor-coated screen. DVST uses two electron guns, the primary gun is used to store the picture pattern; and the flood gun maintains the picture display.

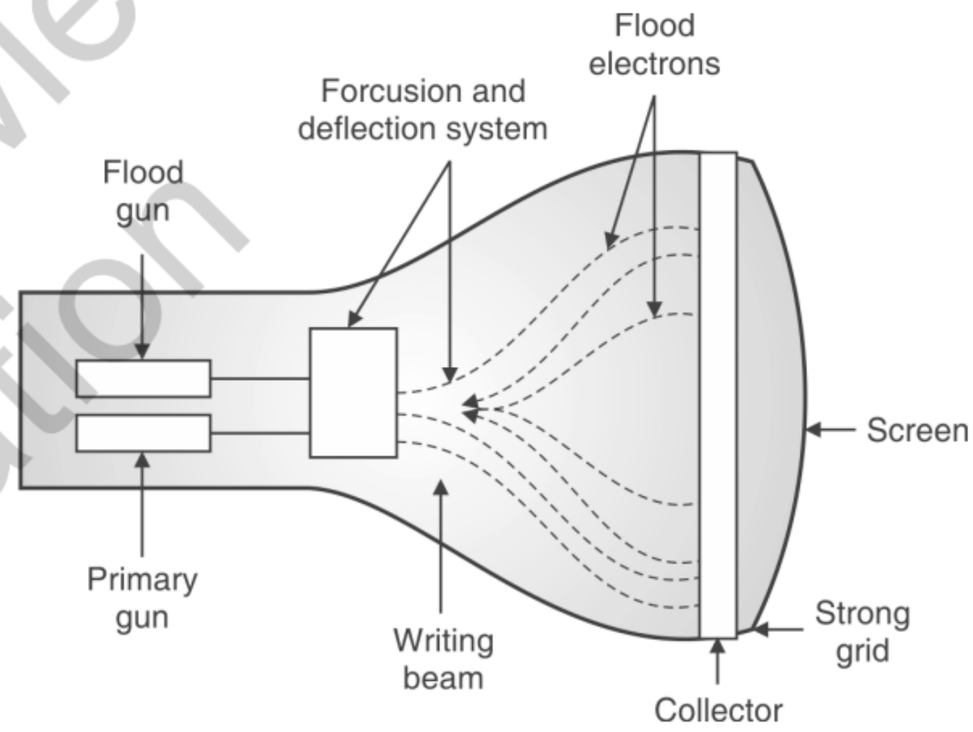


Fig. 1.5.8 : Direct view storage tube

Advantage

- No refreshing is required.
- Complex pictures can be displayed at very high resolutions without flicker.
- Time Consumption is less
- Less expensive

Disadvantage

- Do not display color.
- Selected parts of the picture cannot be erased.
- To manipulate a small part, the entire scene needs to be erased.
- Not useful to display animation.

TechKnowledge Publications

1.5.7 Flat Panel Display

- Most of the graphics monitors are built using CRTs. The downside of CRT is its size and power consumption. Due to the horizontal arrangement of electron guns, CRT monitors are very thick. Flat-panel displays are thin and lightweight. Even they require less power compare to CRT. We can hang large devices on the wall and small devices are wearable.
- Tablets, digital pads etc. allow users to write on it. Nowadays, flat panel is the first choice for laptop, television set, advertisement boards, displays at commercial blocks etc.
- Flat-panel displays can be classified into two categories, emissive display and non-emissive display :

  1. **Emissive display devices :** They convert electrical energy intothe light to generate a display, for example, LED (Light Emitting Diodes), plasma panels, electroluminescent displays etc. Flat CRT is an improved version of CRT, in which electron beams are accelerated parallel to screen rather than perpendicular. Then they are deflected 90 degrees. Although such flat CRT reduces the volume, they are not as successful as other emissive display devices.

  2. **Non-emissive display devices :** They use the optical effect to covert natural or synthetic light to display the graphics on the screen. Example of such a device is LCD (Liquid Crystal Display).

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

1.5.7.1 LCD

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

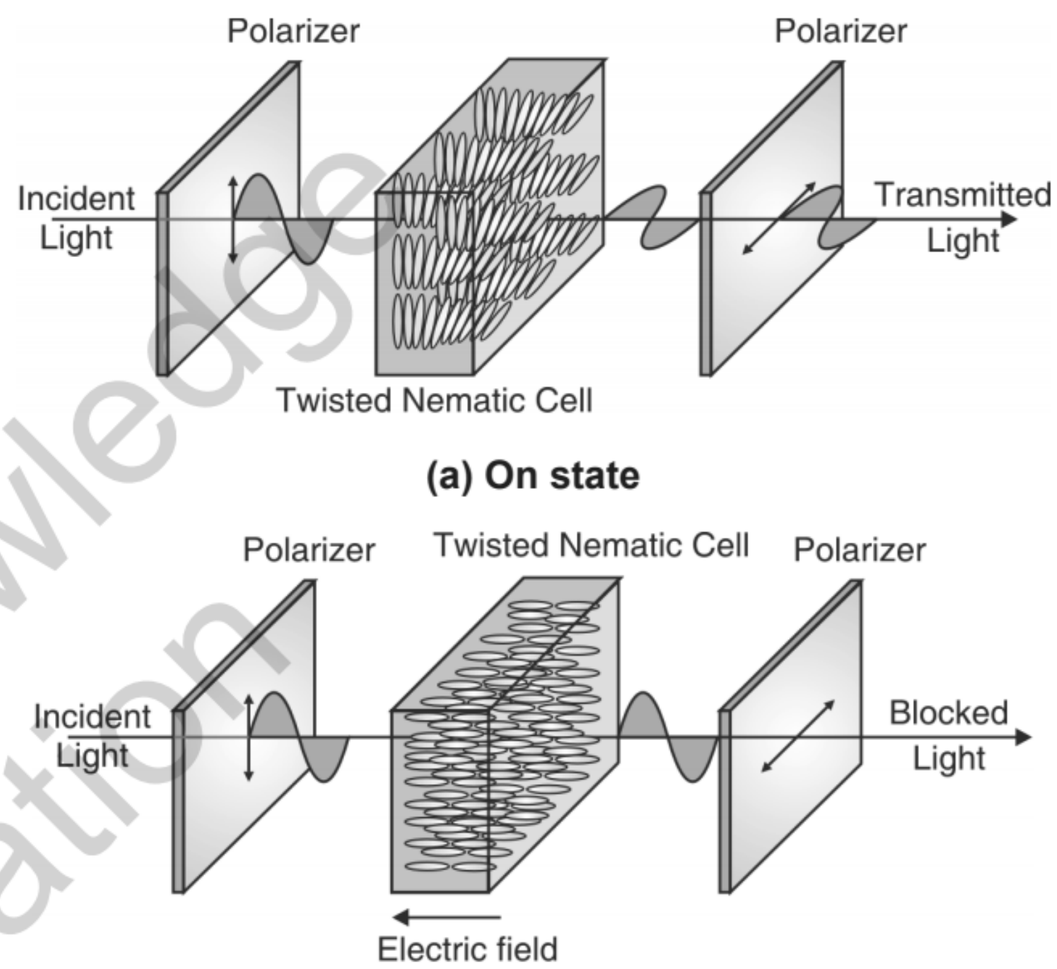


Fig. 1.5.9 : Working of liquid crystal display

- In *'on' state*, nematic cells are arranged as shown in Fig. 1.5.9(a).
- Cell twists the light so that it can pass through the opposite polarizer. The reflected light reaches the eye of the viewer and the scene would be visible. In the *off state*, cells are aligned as shown in Fig. 1.5.9(b). In the *off state*, a cell does not twist the light and so it cannot pass through the opposite polarizer.
- Such a flat-panel device is called **passive matrix LCD**. Another way is to use a transistor at each pixel location is TFT (Thin Film Transistor) technology. Transistors control voltage to decay the charge out of the nematic cell. Such an arrangement is called an **active matrix LCD**.

TechKnowledge Publications

1.5.7.2 LED

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

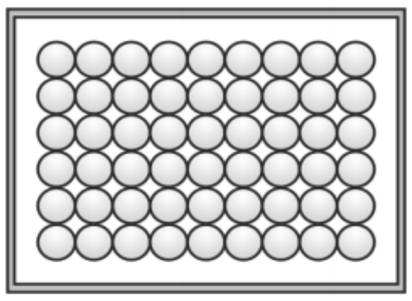


Fig. 1.5.10 : Arrangement of the LED grid

1.5.7.3 Plasma Panels

- Plasma panels are made up of glass plates filled with a mixture of gases like neon. They are also called gas discharge displays due to their characteristic component. Numbers of vertical and horizontal conducting plates are placed on opposite glasses.
- When voltage is applied to conducting plates, gas at the intersection of two conductors glows ups the plasma. The device reads the picture definition from a refresh buffer. Pixels are separated by providing an electric field to the conductor plates.

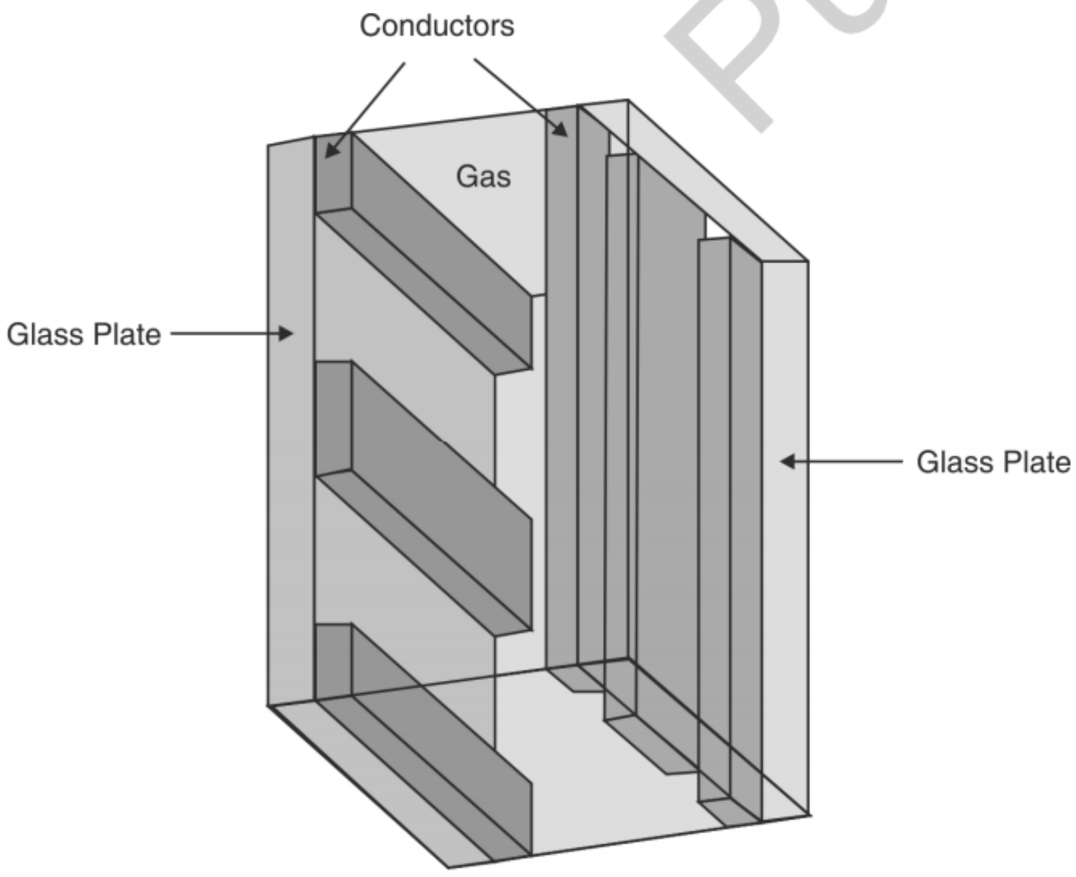


Fig. 1.5.11 : Plasma panel

Disadvantages

Strictly monochromatic.

1.5.7.4 Electroluminescent Displays

- Component arrangement of such display is similar to the plasma panel. The main difference is that the hollow region between opposite glass plates is filled with the phosphor, such as zinc sulfide doped with manganese, instead of gas in the electroluminescent display.
- On applying a high voltage to a pair of crossing electrodes, the phosphor works as a conductor in the intersection region of two electrodes. Generated electrical energy is absorbed by the manganese impurities, which release the energy to become stable and that energy is seen as a light spot similar to the glowing plasma in a plasma panel.

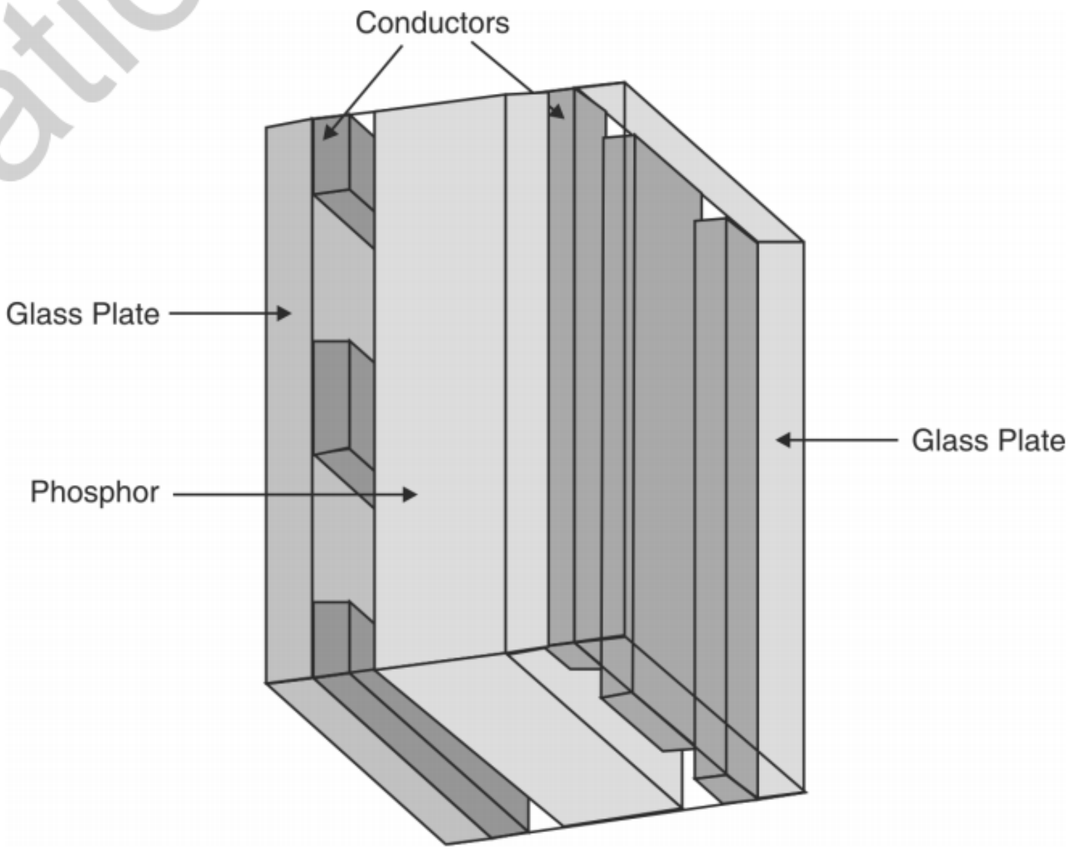


Fig. 1.5.12 : Electroluminescent display

Disadvantage

- More power consumption
- Difficult to achieve good colors or grayscales

Tech Knowledge Publications

1.5.8 Comparison of CRT, LCD, Plasma, and OLED Displays

Sr. No	Parameters for comparison	CRT	LCD	Plasma	OLED
1	Color depth	resolution 4-bit per pixel; It offers better resolution for grayscale	6 to 10-bit per subpixel panels smaller dot pitch, better detail	6 to 8-bit per subpixel panels	8 to 10-bit per subpixel, with some HDR models capable of 12-bit per subpixel.
2	Frame rate	60-85 fps typically, some CRTs can go even higher (200 fps at reduced resolution; internally, display refreshed at input frame rate speed	60 fps typically, gaming monitors can do up to 240 fps; internally, display refreshed at up to 240 Hz	60 fps typically, some can do 120 fps; internally, display refreshed at e.g. 480 or 600 Hz	60 fps typically.
3	Flicker	Noticeable on lower refresh rates (60 Hz and below)	Depends; as of 2013, most LCDs use PWM (strobing) to dim the backlight	Does not normally occur due to high refresh rate.	Does not normally occur.
4	Energy consumption and heat generation	High	Low	Varies with brightness but usually higher than LCD	Energy consumption and heat generation
5	Electro-magnetic radiation emission	Can emit a small amount of X-ray radiation.	Only emits non-ionizing radiation.	Emits strong radio frequency electromagnetic radiation	Electro-magnetic radiation emission
6	Environmental influences	Sensitive to ambient magnetic fields, which can adversely affect convergence and color purity.	Prone to malfunctions on both low (below –4°F/ –20°C) or high (above 45°C/ 113°F) temperatures	High altitude pressure difference may cause poor function or buzzing noises	Can have poor brightness.

Techknowledge Publications

1.6 Graphics Devices

1.6.1 Interactive Devices

Number of devices is available for input of data into graphics workstation. The different input devices are joystick, mouse, keyboard, light pen, touch panels, tablets etc.

1. Joystick

- A joystick consists of a vertical lever that is also referred to as a stick mounted on a base used to control screen cursor movement.
- Many joysticks select the screen position with actual stick movement while others respond to the pressure applied on the stick. Some joysticks are mounted on the keyboard while others function as stand alone units.
- The distance the stick is moved in any direction from it's centre position corresponds to the movement of the cursor on the screen. Potentiometer mounted at the base of the joystick measures the amount of the movement and the springs return the stick to the center position when it is released.

Fig. 1.6.1 : Joystick

2. Mouse

- Mouse is a handheld object used to position screen cursor. Earlier the mouse had wheels or rollers to record the amount and direction of movement. It can move and select text, icons, files, and folders. Nowadays wireless mouse is more in use.

Fig. 1.6.2 : Mouse

3. Keyboard

- An alphanumeric keyboard on graphics system is used primarily as a device to enter text strings. It is a device to input nongraphic data like image labels. It is also used to enter the screen coordinates. Generally the keys are categorized as cursor control keys and function keys on general purpose keyboard.
- Function keys allow the users to enter frequently used operations in a single key stroke. Cursor control keys are used to select displayed objects or coordinate positions by positioning the screen cursor.

Fig. 1.6.3 : Keyboard

4. Light Pens

- A pencil shaped device used to select screen positions by detecting the light coming from the points on the CRT screen. They are sensitive to short burst of light emitted from the phosphor coating at the instant electron beam a particular position.
- An activated light pen, pointed at a spot on the screen as the electron beam lights up that spot, generates an electrical pulse that causes the coordinate position of the electron beam to be located. It requires special implementations for some applications because it cannot detect positions within black areas.

Tech Knowledge Publications

Fig. 1.6.4 : Light pen

– It is a device that allows displayed objects or screen positions to be selected with the touch of a finger. Touch panel technologies are a key theme in current digital devices, including smartphones, slate devices like the iPad. It is used in number of applications like ATM, ticket vending machines, car navigation systems. The touch panels are categorized into optical touch panels, electrical touch panels and acoustical touch panels.

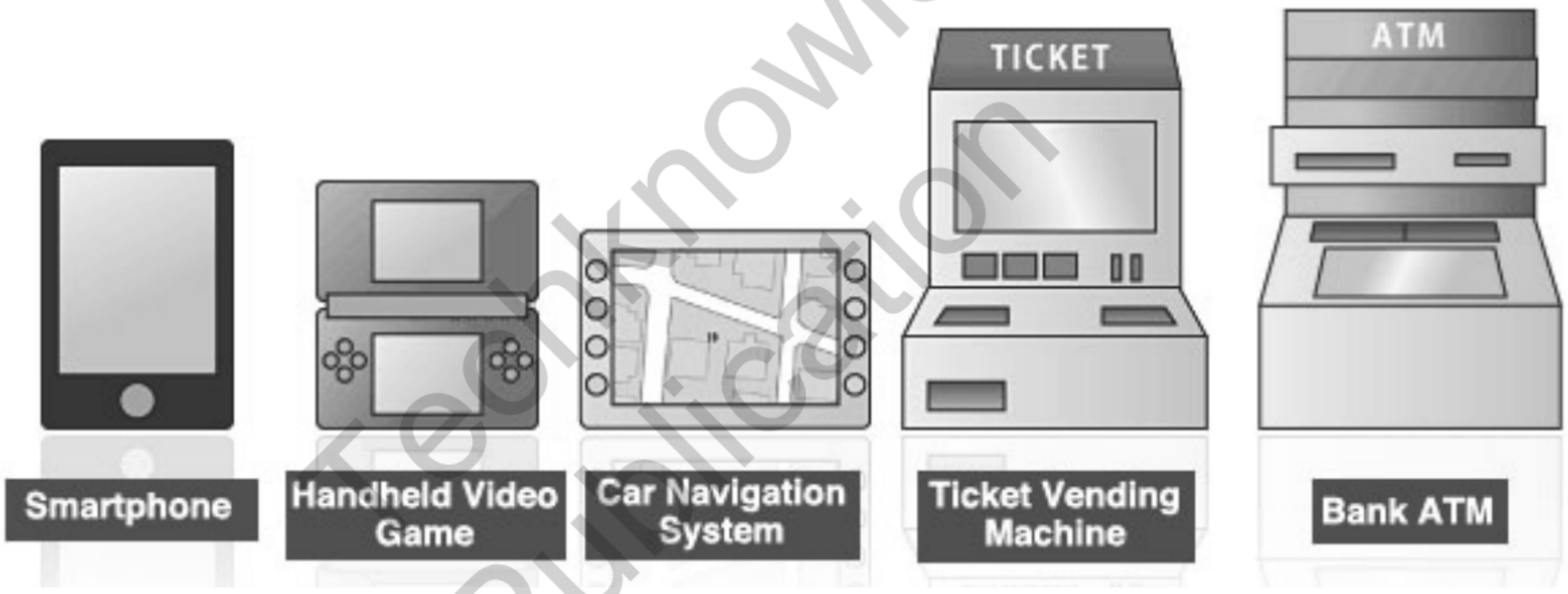


Fig. 1.6.5 : Touch panels

Technowledge Publication

1.7 Applications of Computer Graphics

Q.	Write and explain any four state of the applications of computer graphics.	**SPPU - May 16, 4 Marks**
Q.	State the applications of computer graphics.	**SPPU - Dec. 19, 3 Marks**

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

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

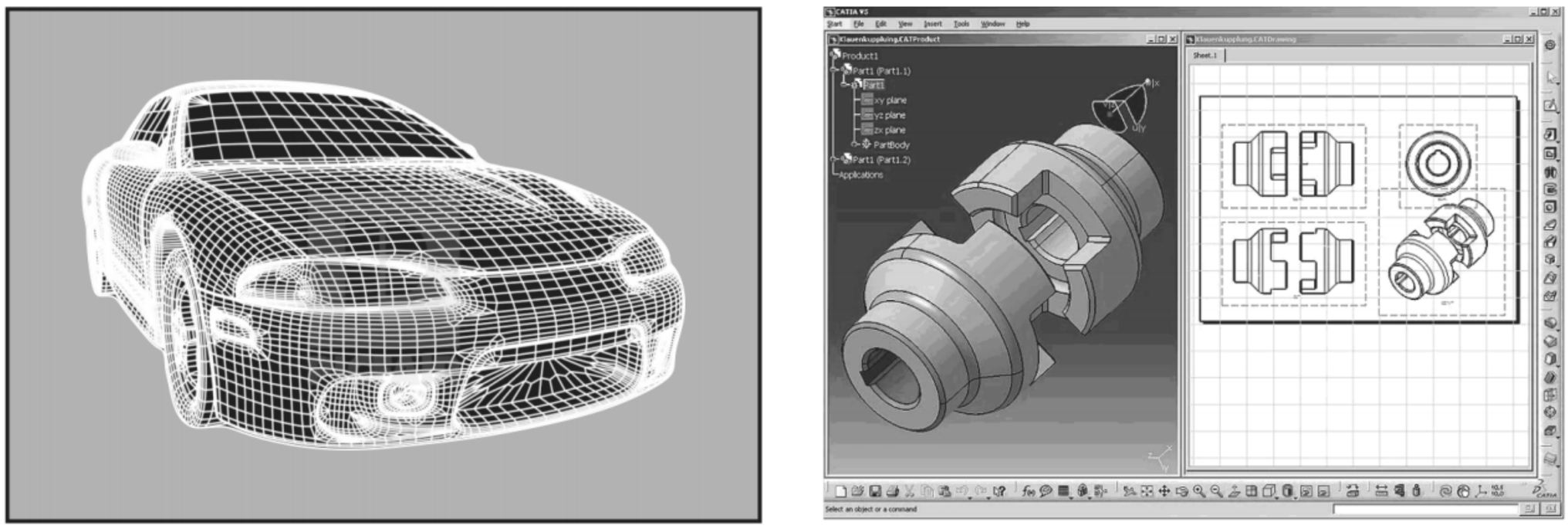


Fig. 1.7.1 : Role of computer graphics in Computer-Aided Designing

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

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

Fig. 1.7.2 : Virtual vehicle navigation system

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

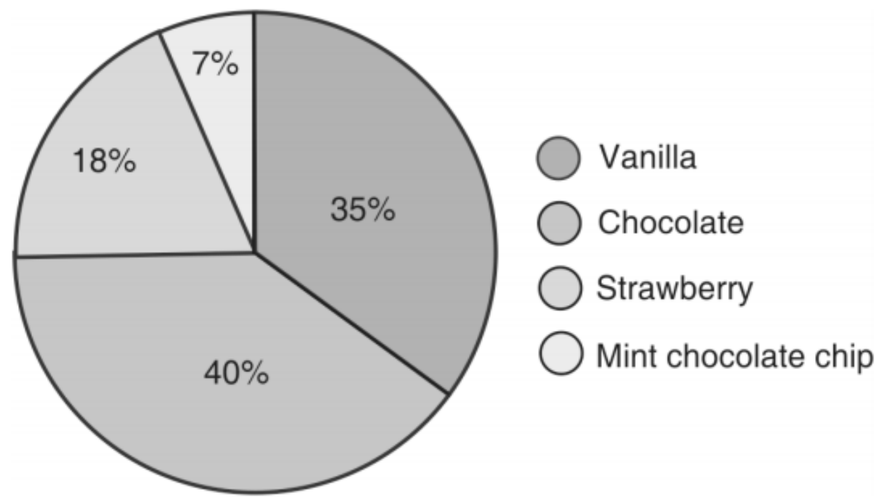


Fig. 1.7.3 : Graphical representation of the survey

Techknowledge Publications

4. **Computer Art :** Computer graphics play a major role in fine art and commercial art. Various computer methods, symbolic packages, software packages, CAD packages, etc. are used by the designer to generate computer art. Using such packages, the artist can easily design shapes and can add motion.

   Morphing is the common graphics processing method used in animation. It is a process of transforming one object into another. Morphing is being used in movies to transform one shape to others, e.g. human to tiger, car or any random shape etc.

Fig. 1.7.4 : Morphing

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

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

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

Fig. 1.7.6 : Scope of computer graphics in training

7. **Visualization :** Data produced from many research applications is too big to analyze on paper. Representing data in form of graphs, charts or some other visual form may help to understand the results in quick time.

   Colour coding can also provide a clue to understanding the data. For example, plain satellite image may not be as useful as if it is segmented using different colors i.e. green indicating forest region, blue indicating water region, black indicating mountains and roads, grey indicating soil etc.

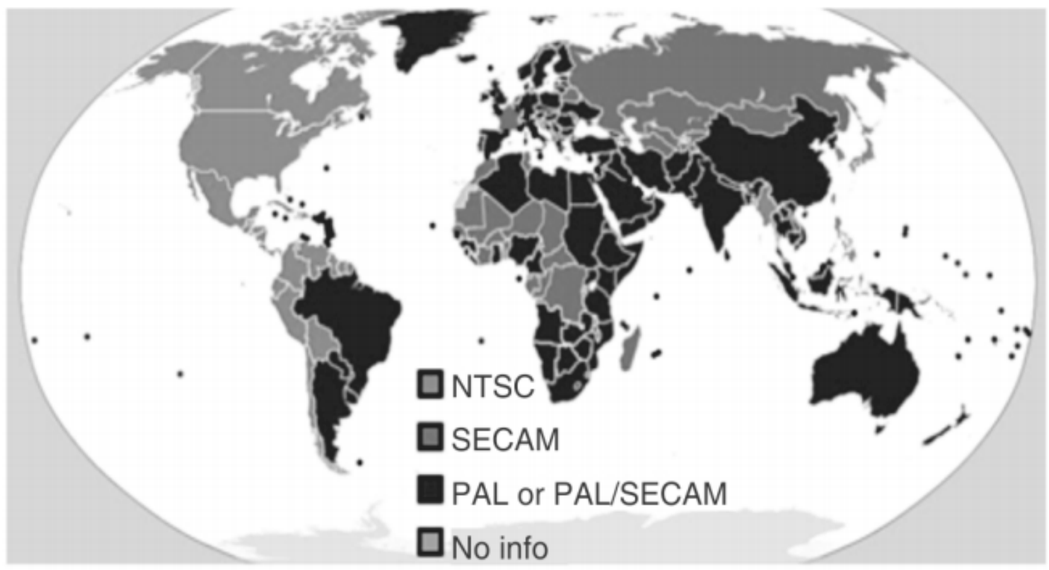


Fig. 1.7.7 : Satellite images with augmented data

Tech Knowledge Publications

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

Fig. 1.7.8 : Graphics user interface

Review Questions

Q. 1 What is computer graphics?

Q. 2 Enlist the classes of operations of computer graphics.

Q. 3 State and explain the applications of computer graphics.

Q. 4 Define the terms: Computer Graphics, Scan conversion, pixel, resolution, aspect ratio, frame buffer.

Q. 5 Why the border of the object displayed on monitor screen is zaggy?

Q. 6 Write short note on vector.

Q. 7 Find the direction and magnitude of the vector joining points A(2, 8) and B(7, 4).

Q. 8 Explain how pixels are related to the display memory.

Q. 9 Find the size of frame buffer for the system supporting binary colors and having resolution 680 x 420.

Q. 10 Explain the working mechanism of CRT.

Q. 11 Define: Refresh CRT, persistence, horizontal retrace, vertical retrace.

Q. 12 Explain the concept of odd and even field in interlaced display.

Q. 13 State the advantages and disadvantages of random scan display.

Q. 14 Write a short note on beam penetration method.

Q. 15 Write a short note on shadow mask method.

Q. 16 How in line shadow mask differs from delta shadow mask?

Q. 17 Discuss DVST with its pros and cons.

Q. 18 State working principle of emissive and non-emissive display devices.

Q. 19 Write a note on LCD.

Q. 20 Explain working of LED.

Q. 21 How plasma panel works?

Q. 22 Write short note on Electroluminescent displays.

Q. 23 Explain the way of computing pixel positions using slope-intercept formula.

Q. 24 Explain explicit formula of line drawing.

Technical Publication

❑❑❑

Introduction to OpenGL

Syllabus

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

2.1 Introduction to OpenGL

Q. Write short note on OpenGL.

SPPU - Dec. 19, 2 Marks

- OpenGL (Open Graphics Library) is a cross platform, hardware acceleration, language independence, standard industrial API for producing 2D and 3D graphical objects.
- Modern computers have provided the GPU (Graphics Processing Unit) in its memory to speed up the transmission of graphics. A low-level graphics programming API which contains over 250 function. It was ddeveloped by Silicon Graphics Inc. (SGI) in 1992.
- OpenGL is a better software interface than graphics hardware. In other words, OpenGL graphics rendering commands released by the applications can be directed to graphics hardware and accelerated.

Definition : Rendering is the process of generating an image from a model (or models in what collectively could be called a scene file), by means of computer programs.

The 3 sets of libraries is used in OpenGL programs :

1. **Core OpenGL (GL) :** It contains hundreds of commands, starting with the "gl" prefix (e.g. glColor, glVertex, glTranslate, glRotate). Core OpenGL models an object with a set of basic geometric primitives like a point, line and polygon.
2. **OpenGL Utility Library (GLU) :** It is built on top of Core OpenGL to provide essential features (such as setting up camera views and projections) and advanced building models (such as quadric faces and polygon tessellation). GLU commands start with the "glu" prefix (e.g., gluLookAt, gluPerspective).

   To display an object using Open GL, a display window is needed to be setup on video screen, which is a rectangular area on the screen where the object is to be displayed. The basic OpenGL functions do not allow to create the display window as it contains device independent graphics functions and window–management operations are dependent on the computer system that is being used by the user. On the other hand there are many window-system libraries which support OpenGL functions for many types of machines.

   The OpenGL **Extension to the X Window System** (**GLX**) provides a set of routines that are prefixed with the letters **glX**. Microsoft Windows systems, **WGL** routin`es provide a **Windows-to-OpenGL** interface.
3. **OpenGL Utility Toolkit (GLUT) :** OpenGL is designed to be independent of the windowing system or operating system. GLUT is required to interact with the Operating System (such as window creation, key management and mouse input); and provides multiple building models (such as sphere and torus).

   The GLUT commands start with the "glut" prefix (e.g. glutCreatewindow, glutMouseFunc). GLUT is platform independent, built on top of OpenGL extension such as GLX for the X Window System, Microsoft's Window WGL, and AGL, CGL or Cocaoa for Mac OS.

Technoknowledge Publication

To quote from opengl.org : "GLUT is designed to build small and medium-sized OpenGL applications. While GLUT is well-suited to learning OpenGL and simplifying OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small."

4. **OpenGL Extension Wrangler Library (GLEW)** : "GLEW is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform." Source and pre-build binary available at http://glew.sourceforge.net/. A standalone utility called "glewinfo.exe" (under the "bin" directory) can be used to produce the list of OpenGL functions supported by the graphics system.

Header Files

In all the graphics programs, the header file for the OpenGL core library is to be included. For most applications we will also need GLU, and on many systems we will need to include the header file for the window system. For example, with Microsoft Windows, the header file that accesses the WGL routines is **windows.h.** This header file must be listed before the OpenGL and GLU header files as it contains macros required by the Microsoft Windows version of the OpenGL libraries. Hence the source input file in this case would begin with

```
#include < windows.h >
#include < GL/gl.h >
#include < GL/glu.h >
```

On the other hand, if GLUT is used to handle the window-managing operations, gl.h and glu.h need not be included because GLUT makes sure that these will be included correctly. Hence, the header files for OpenGL and GLU can be replaced with #include <GL/glut.h> (we can include gl.h and glu.h also, but it would become redundant and affect portability of program).

2.1.1 OpenGL and Related APIs

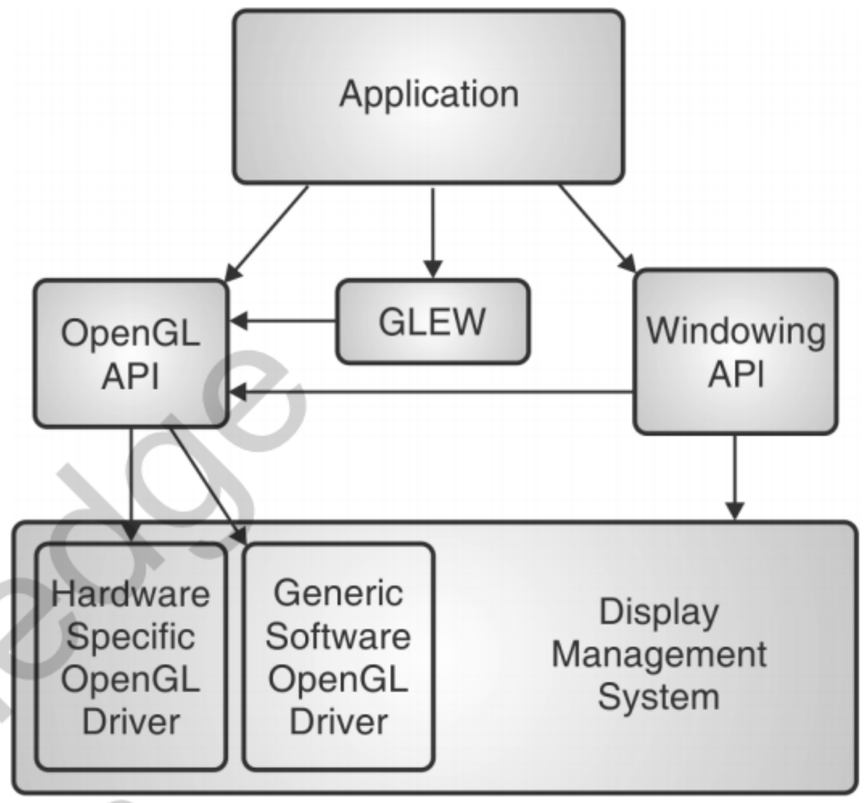


Fig. 2.1.1 : OpenGL API

- Fig. 2.1.1 illustrates the relationships of the various libraries and window system components. Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API. Prototype applications, or ones that don't require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.
- The OpenGL architecture is based on a client-server model. An application program written to use the OpenGL API is the "client" and runs on the CPU. The implementation of the OpenGL graphics engine (including the GLSL shader programs) is the "server" and runs on the GPU.
- Geometry and many other types of attributes are stored in buffers called Vertex Buffer Objects (or VBOs).

Technknowledge Publication

- These buffers are allocated on the GPU and filled by the CPU program. Modeling, rendering, and interaction is very much a cooperative process between the CPU client program and the GPU server programs written in GLSL.
- An important part of the design process is to decide how best to divide the work and how best to package and communicate required information from the CPU to the GPU.

2.2 OpenGL Architecture

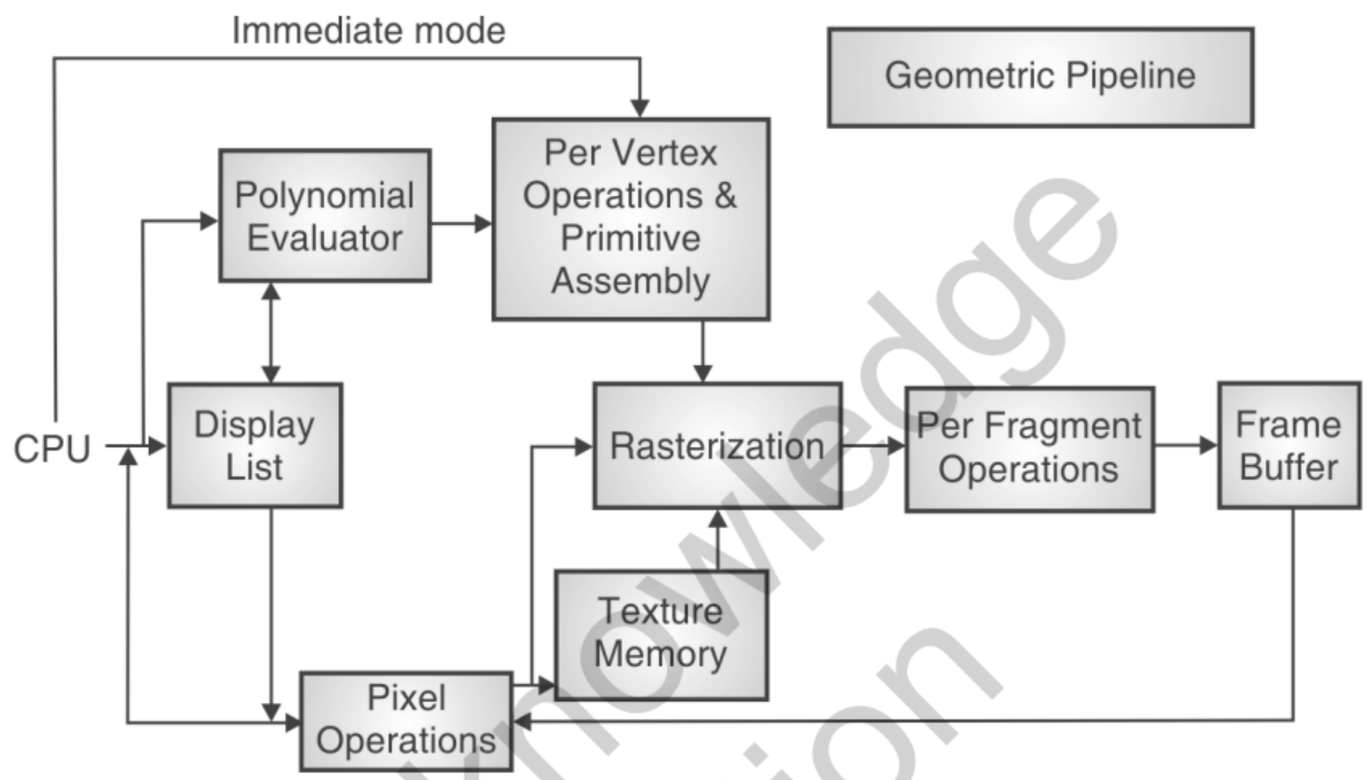


Fig. 2.2.1 : OpenGL architecture

The Commands or the instructions are given from the CPU as the type of object to be drawn.

- **Display List** : It can accumulate some commands in a display list for processing at a later time (Batch mode) , Or can proceed immediately through the pipeline. The glNewList( ) command begins the creation of a display list, and glEndList( ) ends it. OpenGL commands called between glNewList( ) and glEndList( ) are appended to the display list, and optionally executed as well. To trigger the execution of a list or set of lists, use glCallList( ) or glCallLists( ) and supply the identifying number of a particular list or lists. you can delete a set of display lists with glDeleteLists( ).
- **Polynomial Evaluator :** It provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values.
- **Per-Vertex Operations & Primitive Assembly :** Process geometric primitives like points, line segments, and polygons as vertices and are transformed, lit, and clipped to the viewport in preparation for the next stage.
- **Rasterization :** Produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.

  Texture Memory : The images and geometry pass through separate pipeline and join at rasterizer.
- **Per Fragment Operations :** Operations like Z-buffering, and blending of incoming pixel colors with stored colors, and masking and other logical operations on pixel values are done.

Technical Publications

- **Pixel Operations :** Input data can be in the form of pixels (image for texture mapping) is processed in the pixel operations stage.

The Framebuffer

- The fragments that are produced by rasterization are given to the framebuffer where they are displayed. The framebuffer is a rectangular array of n bitplanes. The bitplanes are organized into several logical buffers - Color, Depth, Stencil, and Accumulation.
- The **color buffer** contains the color information of the fragment.
- The **depth buffer** contains the depth information of the fragment, and it is mainly used for z-buffering for removal of hidden surfaces.
- **The stencil buffer** can be related with fragments that pass the conditional tests and make it into the framebuffer. It can be useful for multiple-pass algorithms.
- **The accumulation buffer** is also used for multiple-pass algorithms. It can average the values stored in the color buffer. Full-screen antialiasing can be achieved by jittering the viewpoint. Depth of Field can be achieved by jittering the view angle. Motion blur can be achieved by stepping the scene in time.
- **Stereo and double-buffering** are also supported by OpenGL, depending on the implementation. In this case it divides the framebuffer into up to 4 different sections namely front and back buffer, left and right. Other auxiliary buffers may be available on some implementations.

OpenGL Operation

- Geometric data (vertices, rows, and polygons) follows a row of boxes that include evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are handled in a separate part of the process. Both of these types of data are generated by rasterization and per-fragment operations before the last pixel data is written into the frame buffer.
- These operations include conditional updates to the frame buffer based on incoming or previously stored z-values (z-buffering) and mixing of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values. Input data can be in the form of pixels rather than vertices. Such details, which may describe the image to be used in the texture mapping, skip the first working step and are instead processed as pixels, in the pixel processing stage.
- The result of this section is stored as a texture memory, for use in the reorganization phase, or rasterized and the resulting fragments are merged into the frame buffer as if they were generated from geometric data. All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be accessed through the OpenGL application.
- The spatial coordinates of every vertex are transformed by the model view matrix, in the per-vertex operations stage of processing whereas while the normal vector is transformed by that matrix's inverse and renormalized if specified. The fragments (not pixels directly) are produced during the rasterization process that consists of color, depth and a texture. Fragments undergo tests and processing before it is written into the frame buffer as the pixel values.

2.2.1 GPU vs CPU Performance

- In order to understand the difference between a GPU and a CPU we need to compare both process the tasks.
- CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores designed for handling multiple tasks simultaneously.

2.3 Basic Syntax of OpenGL

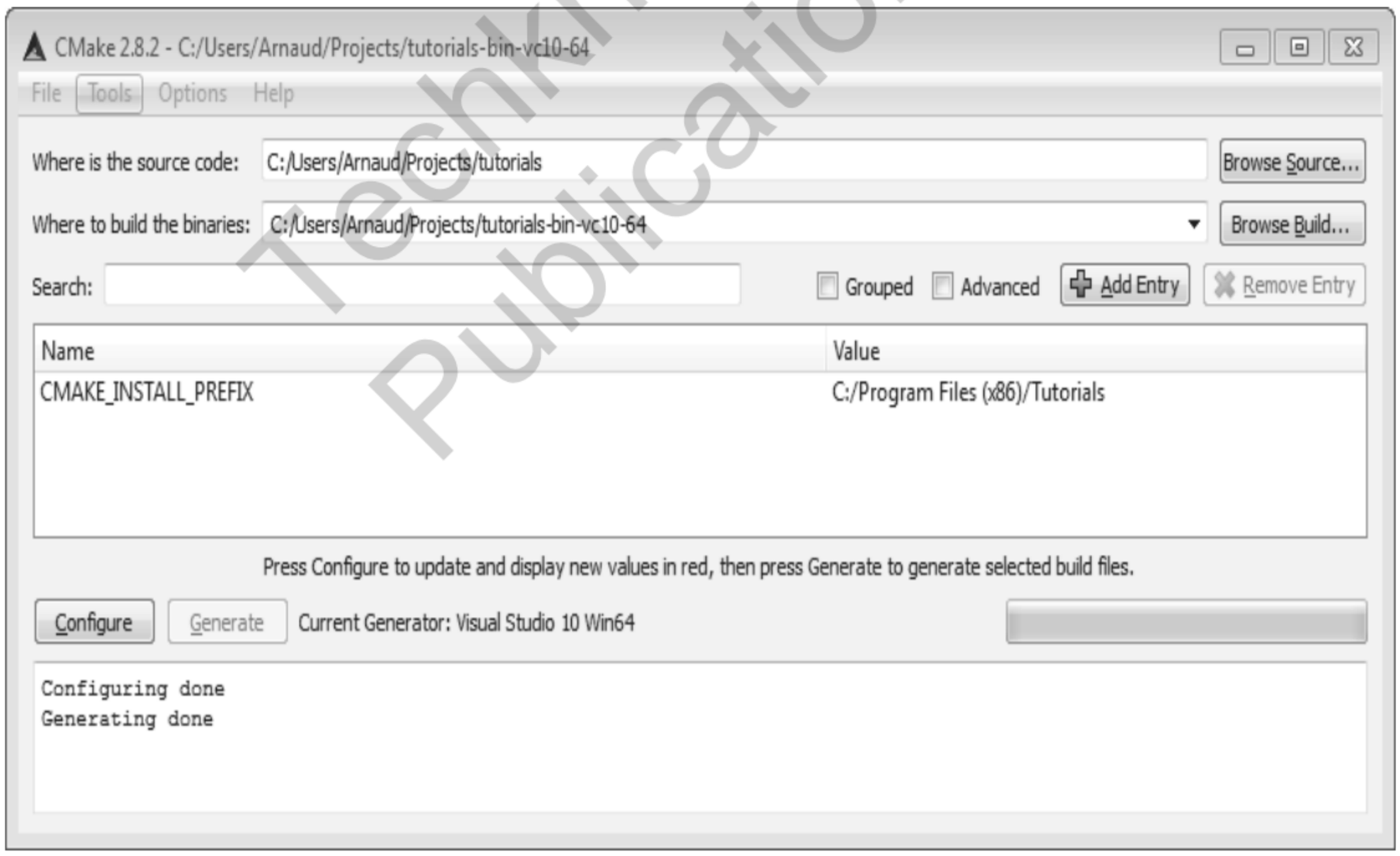
- In the Open GL basic library (OpenGL core library) the function names are prefixed with gl and every component word within a function name has first letter capital. Example glBegin , glEnd( ), glVertex, glClear.

Technknowledge Publication

- Some function needs one or more arguments to be assigned to a symbolic constant that signifies, for example name of a parameter, a specific mode, a value. All of these constants start with uppercase letters GL.
- The component words within the constant name are written in capital letters and the underscore is used as the separator between the component words. Example for symbolic constants GL_RGB, GL_AMBIENT_AND_DIFFUSE, GL_2D, GL_POLYGON

2.3.1 Building on Windows

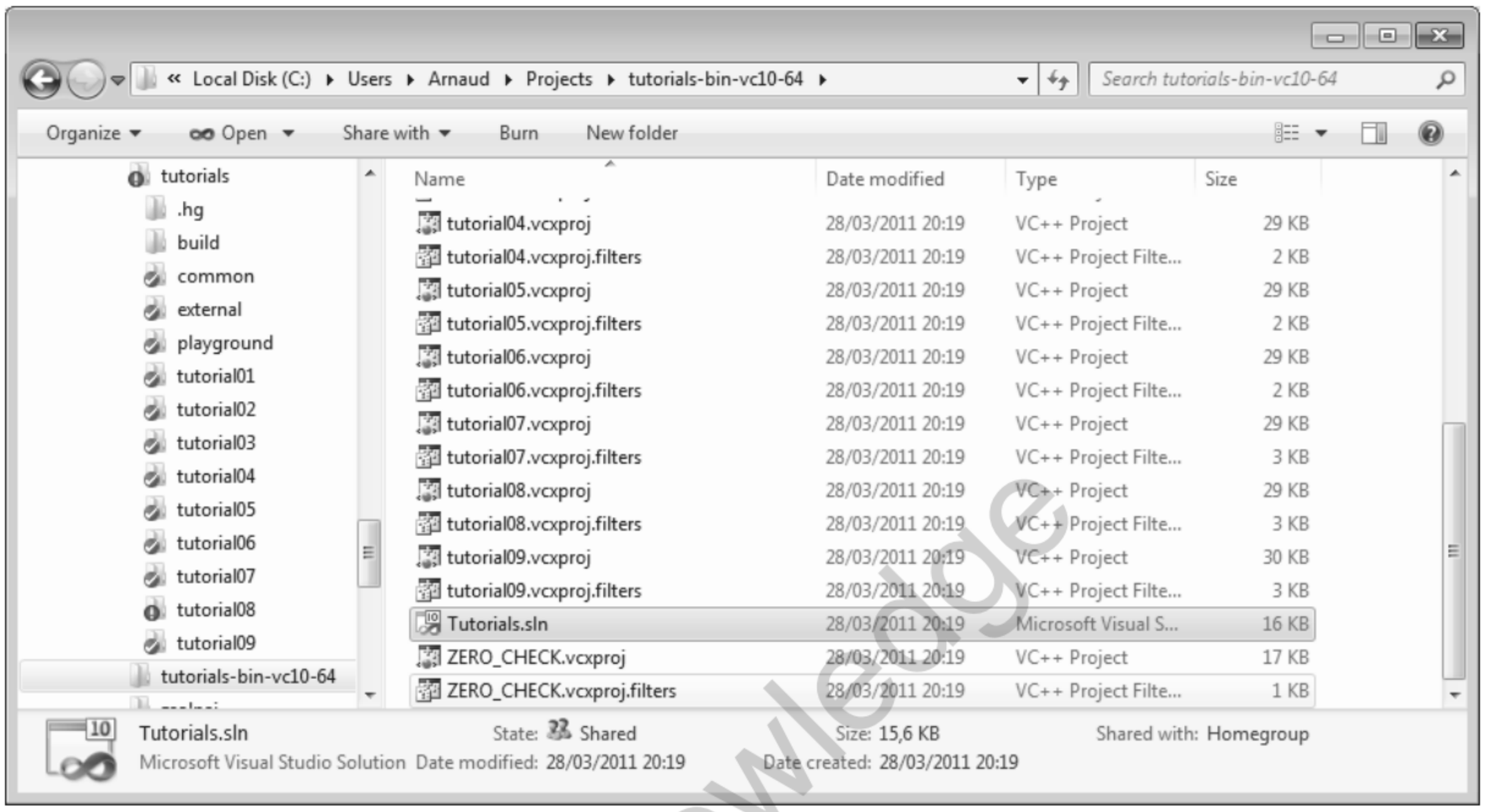
- Updating the drivers should be easy. Just go to NVIDIA's or AMD's website and download the drivers. If you are unsure about your GPU model : Control Panel -> System and Security -> System -> Device Manager -> Display adapter. If you have an integrated Intel GPU, drivers are usually provided by your OEM (Dell, HP).
- Visual Studio 2017 Express is always suggested for Desktop as a compiler. Make sure you choose custom installation and check C++. Subsequent steps are explained with Visual Studio, but should be similar with any other IDE.
- Download CMake and install it.
- Download the source code and unzip it, for instance in C:\Users\XYZ\Projects\OpenGLTutorials\.
- Launch CMake. In the first line, navigate to the unzipped folder. If unsure, choose the folder that contains the CMakeLists.txt file.
- In the second line, enter where you want all the compiler's stuff to live. For instance, you can choose C:\Users\XYZ\Projects\OpenGLTutorials-build-Visual2017-64bits\, or C:\Users\XYZ\Projects\OpenGLTutorials\build\Visual2017-32bits.
- Notice that it can be anywhere, not necessarily in the same folder.



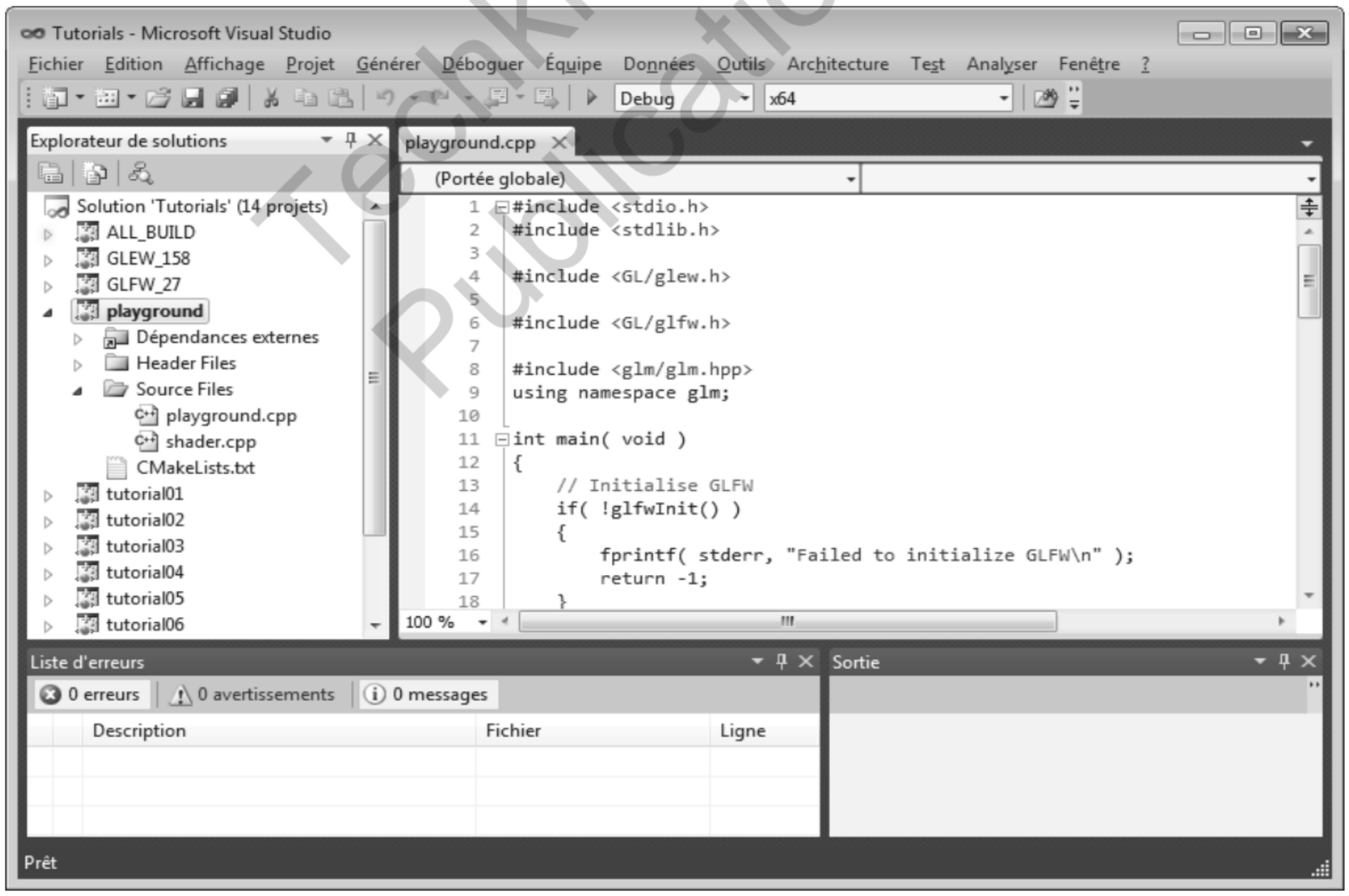
- Click on the Configure button. Since this is the first time you configure the project, CMake will ask you which compiler you would like to use. Choose wisely depending on step 1. If you have a 64 bit Windows, you can choose 64 bits; if you don't know, choose 32 bits.
- Click on Configure until all red lines disappear. Click on Generate. Your Visual Studio project is now created. You can now forget about CMake.

Technknowledge Publications

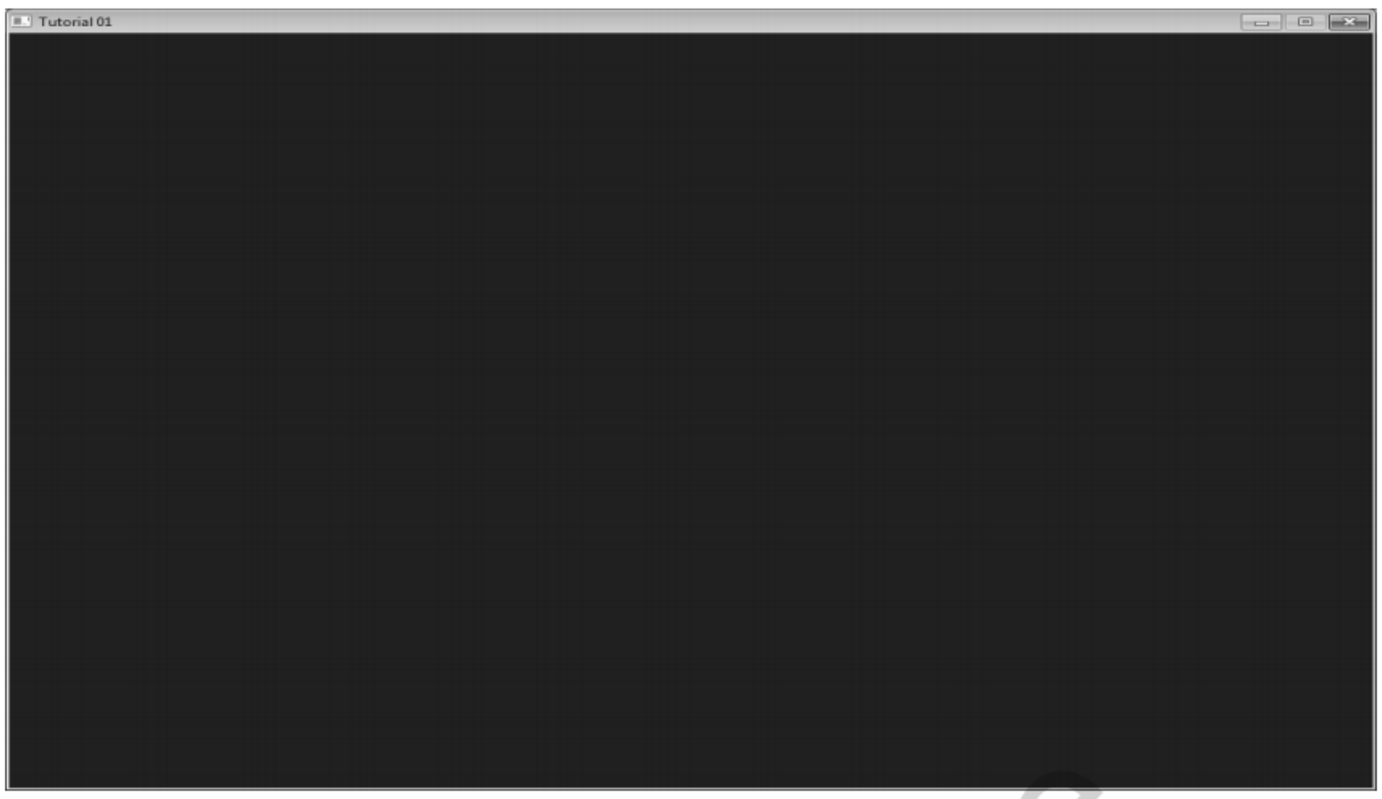
– Open C:\Users\XYZ\Projects\OpenGLTutorials-build-Visual2010-32bits. You will see a Tutorials.sln file, open it with Visual Studio.



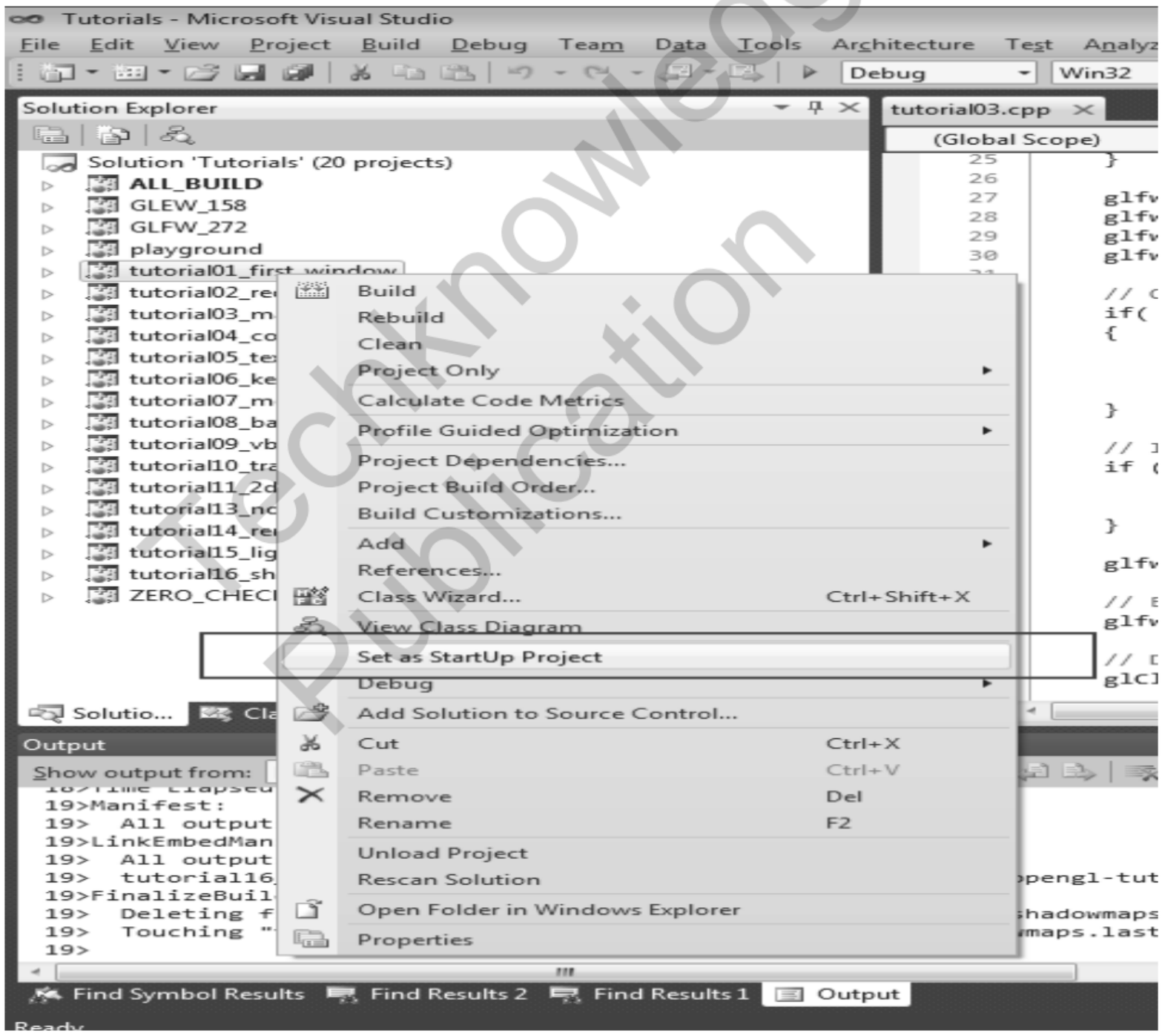
– In the *Build* menu, click *Build All*. Every tutorial and dependency will be compiled. Each executable will also be copied back into C:\Users\XYZ\Projects\OpenGLTutorials\.



– Open C:\Users\XYZ\Projects\OpenGLTutorials\playground, and launch playground.exe. A black window should appear.



– You can also launch any tutorial from inside Visual Studio. Right-click on Playground once, "Choose as startup project". You can now debug the code by pressing F5.



2.3.2 Building on Linux

There are so many Linux variants out there that it's impossible to list every possible platform. Adapt if required, and don't hesitate to read your distribution's documentation.

– Install the latest drivers. We highly recommend the closed-source binary drivers. It's not GNU or whatever, but they work. If your distribution doesn't provide an automatic install, try Ubuntu's guide.

- Install all needed compilers, tools & libs. Complete list is : *cmake make g++ libx11-dev libxi-dev libgl1-mesa-dev libglu1-mesa-dev libxrandr-dev libxext-dev libxcursor-dev libxinerama-dev libxi-dev* . Use sudo apt-get install ***** or su && yum install ******.
- Download the source code and unzip it, for instance in ~/Projects/OpenGLTutorials/
- Change directory(cd) in ~/Projects/OpenGLTutorials/ and enter the following commands :

  mkdir build

  cd build cmake ..
- A makefile has been created in the build/directory.
- Type "make all". Every tutorial and dependency will be compiled. Each executable will also be copied back into ~/Projects/OpenGLTutorials/ . Hopefuly no error occurs.
- Open ~/Projects/OpenGLTutorials/playground, and launch ./playground. A black window should appear.

Sample OpenGL Program

```
#include<stdio.h>
#include<GL/glut.h>                    // (or others, depending on the system in use)
void init (void)
{
    glClearColor (1.0, 1.0, 1.0, 0.0);     // Set display-window color to white.
    glMatrixMode (GL_PROJECTION);          // Set projection parameters.
    gluOrtho2D (0.0, 150.0, 0.0, 100.0);
}
void lineSegment (void)
{
    glClear (GL_COLOR_BUFFER_BIT);         // Clear display window.
    glColor3f (0.0, 0.4, 0.2);             // Set line segment color to green.
    glBegin (GL_LINES);
    glVertex2i (170, 15);                  // Specify line-segment geometry.
    glVertex2i (10, 135);
    glEnd ( );
    glFlush ( );                           // Process all OpenGL routines as quickly as possible.
}
void main (int argc, char** argv)
{
    glutInit (&argc, argv);                // Initialize GLUT.
```

Techknowledge Publications

```
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);        // Set display mode.
    glutInitWindowPosition (50, 100);                    // Set top-left display-window position.
    glutInitWindowSize (400, 300);                       // Set display-window width and height.
    glutCreateWindow (" OpenGL LineProgram");            // Create display window.
    init ( );                                            // Execute initialization procedure.
    glutDisplayFunc (lineSegment);                       // Send graphics to display window.
    glutMainLoop ( );                                    // Display everything and wait.
}
```

– To compile the above program in ubuntu,

```
gcc filename.c -lGL -lGLU -lglut -lm
```

where filename.c is the name of the file with which this program is saved.

glClearColor (1.0, 1.0, 1.0, 0.0);

The above functions takes values in RGB format(RED, GREEN, BLUE), setting first three arguments in this function are set to 1 gives a white background color for the display window. Else If first three arguments in this function are set to 0.0 gives a BLACK background color for the display window.

glColor3f (0.0, 0.4, 0.2);

The suffix **3f** on the **glColor** function indicates that the three RGB color components is specified using floating-point (**f**) values. The values range from 0.0 to 1.0, and we have set red = 0.0, green = 0.4, and blue = 0.2.

```
glMatrixMode (GL_PROJECTION);
gluOrtho2D (0.0, 150.0, 0.0, 100.0);
```

The above function specifies that an orthogonal projection is used to map the contents of a 2D rectangular area of world coordinates to the screen, and that the *x*-coordinate values within this rectangle range from 0.0 to 150.0 with *y*-coordinate values ranging from 0.0 to 100.0. The objects that is defined within this world-coordinate rectangle will be shown within the display window. Anything outside this coordinate range will not be displayed. Hence, the GLU function gluOrtho2D defines the coordinate reference frame within the display window to be (0.0, 0.0) at the lower-left corner of the display window and(150.0, 100.0) to the upper-right window corner. Since we are only displaying a 2D object, the orthogonal projection simply pastes the image in display window. Finally, we need to call the appropriate OpenGL routines to create our line segment.

The following code defines a two-dimensional, straight-line segment with integer, Cartesian endpoint coordinates (175, 15) and (10, 135).

```
glBegin (GL_LINES);
glVertex2i (170, 15);
glVertex2i (10, 135);
glEnd ( );
```

2.4 Primitives and Attributes

– In OpenGL the term **primitive** is referred to, two similar but separate concepts. The first meaning of "Primitive" refers to the interpretation method used by OpenGL to determine what a vertex stream (an ordered list of vertices) represents when being rendered e.g. "GL_POINTS". Such sequences of vertices can be arbitrarily long.

– The alternate meaning of "Primitive", also referred to as a "base primitive", refers to the *result* of the interpretation of a vertex stream, as part of primitive assembly. So, processing a vertex stream by one of these primitive interpretations results in an ordered sequence of primitives. The individual primitives are sometimes described as "base primitives".

Techknowledge Publications

- OpenGL supports several basic primitive types, including points, lines, quadrilaterals, and general polygons. All of these primitives are specified using a sequence of vertices. The primitive type determines how OpenGL interprets and renders the sequence of vertices.

Point Primitives

There is only one kind of point primitive: GL_POINTS. This will cause OpenGL to interpret each individual vertex in the stream as a point. Points that have a Texture mapped onto them are often called "point sprites".

- We call glVertex* where * is 2 or 3 characters of the form nt or ntv, and n = number of dimensions (2 or 3), t = data type, e.g. integer (i), float (f), double (d), v if present ⇒'use an array'.
- 2D Points (x, y) are treated as 3D points (x, y, z) with z = 0.
- We use OpenGL types : #define GLfloat float Examples : glVertex2i(GLint xi, GLintyi); glVertex3f(GLfloat x, GLfloat y, GLfloat z); Glfloat vertex[3]; glVertex3fv(vertex);
- Vertices are grouped together to form objects, using glBegin and glEnd.
- Line segments : glBegin(GL_LINES); glVertex2f(x1,y1); glVertex2f(x2,y2); glEnd();
- Sequences of points : glBegin(GL_POINTS); glVertex2f(x1,y1); glVertex2f(x2,y2); glEnd();
- OpenGL supports three classes of geometric primitives : *points*, *line segments*, and *closed polygons*. They are specified via vertices. Each vertex is associated with its attributes such as the position, color, normal and texture. OpenGL provides 10 primitives as shown. **Sphere, 3D box and pyramid** are not primitives. They are typically assembled using primitive triangle or quad.

OpenGL interprets the vertices and renders each primitive using the following rules :

Table 2.4.1

Sr. No	OpenGL commands	Description
1.	GL_POINTS	This type of primitive is used to render mathematical points. OpenGL renders a point for each specified vertex.
2.	GL_LINES	This type of primitive is used to draw unconnected line segments. OpenGL draws a line segment for each group of two vertices. If the application specifies n vertices, OpenGL renders $n/2$ line segments. If n is odd, OpenGL ignores the final vertex
3	GL_LINE_STRIP	This type of primitive is used to draw a sequence of connected line segments. OpenGL renders a line segment between the first and second vertices, between the second and third, between the third and fourth, and so on. If the application specifies n vertices, OpenGL renders n–1 line segments
4	GL_LINE_LOOP	This type of primitive is used to close a line strip. OpenGL renders this primitive like a GL_LINE_STRIP with the addition of a closing line segment between the last and first vertices.
5	GL_TRIANGLES	This type of primitive is used to draw individual triangles. OpenGL renders a triangle for each group of three vertices. If the application specifies n vertices, OpenGL renders n/3 triangles. If n isn't a multiple of 3, OpenGL ignores the excess vertices.

Technical Knowledge Publications

Sr. No	OpenGL commands	Description
6	GL_TRIANGLE_STRIP	This type of primitive is used to draw a sequence of triangles that share edges. OpenGL renders a triangle using the first, second, and third vertices, and then another using the second, third, and fourth vertices, and so on. If the application specifies n vertices, OpenGL renders n–2 connected triangles. If n is less than 3, OpenGL renders nothing.
7	GL_TRIANGLE_FAN	This type of primitive is used to draw a fan of triangles that share edges and also share a vertex. Each triangle shares the first vertex specified. If the application specifies a sequence of vertices v, OpenGL renders a triangle using v_0, v_1, and v_2; another triangle using v_0, v_2, and v_3; another triangle using v_0, v_3, and v_4; and so on. If the application specifies n vertices, OpenGL renders n–2 connected triangles. If n is less than 3, OpenGL renders nothing.
8	GL_QUADS	This type of primitive is used to draw individual convex quadrilaterals. OpenGL renders a quadrilateral for each group of four vertices. If the application specifies n vertices, OpenGL renders n/4 quadrilaterals. If n isn't a multiple of 4, OpenGL ignores the excess vertices.
9	GL_QUAD_STRIP	This type of primitive is used to draw a sequence of quadrilaterals that share edges. If the application specifies a sequence of vertices v, OpenGL renders a quadrilateral using v_0, v_1, v_3, and v_2; another quadrilateral using v_2, v_3, v_5, and v_4; and so on. If the application specifies n vertices, OpenGL renders (n-2)/2 quadrilaterals. If n is less than 4, OpenGL renders nothing.
10	GL_POLYGON	The primitive GL_POLYGON is used to draw a single filled convex n-g on primitive. OpenGL renders an n-sided polygon, where n is the number of vertices specified by the application. If n is less than 3, OpenGL renders nothing.

Tech Knowledge Publication

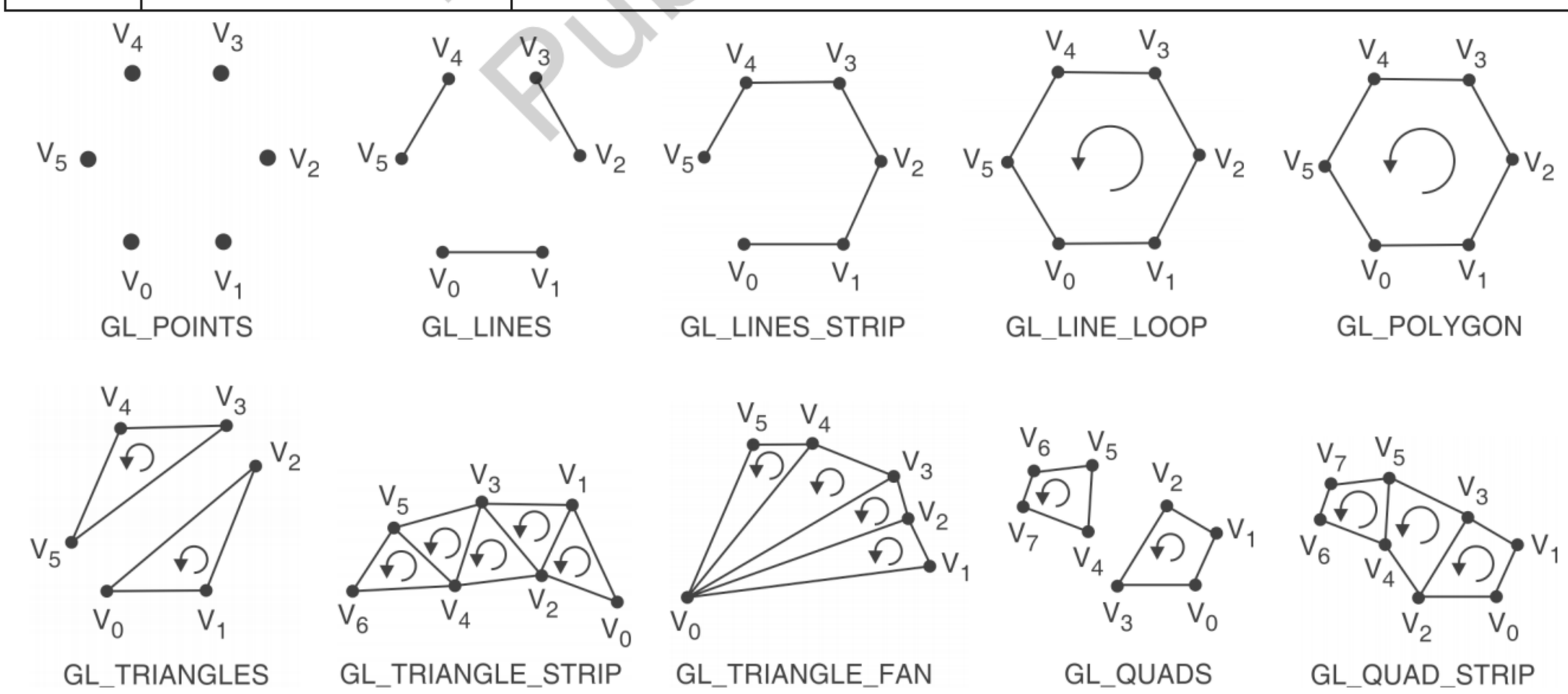


Fig. 2.4.1 : OpenGL primitives

For GL_QUADS, GL_QUAD_STRIP, and GL_POLYGON, all primitives must be both planar and convex. Otherwise, OpenGL behavior is undefined. The GLU library supports polygon tessellation, which allows applications to render filled primitives that are nonconvex or self-intersecting, or that contain holes.

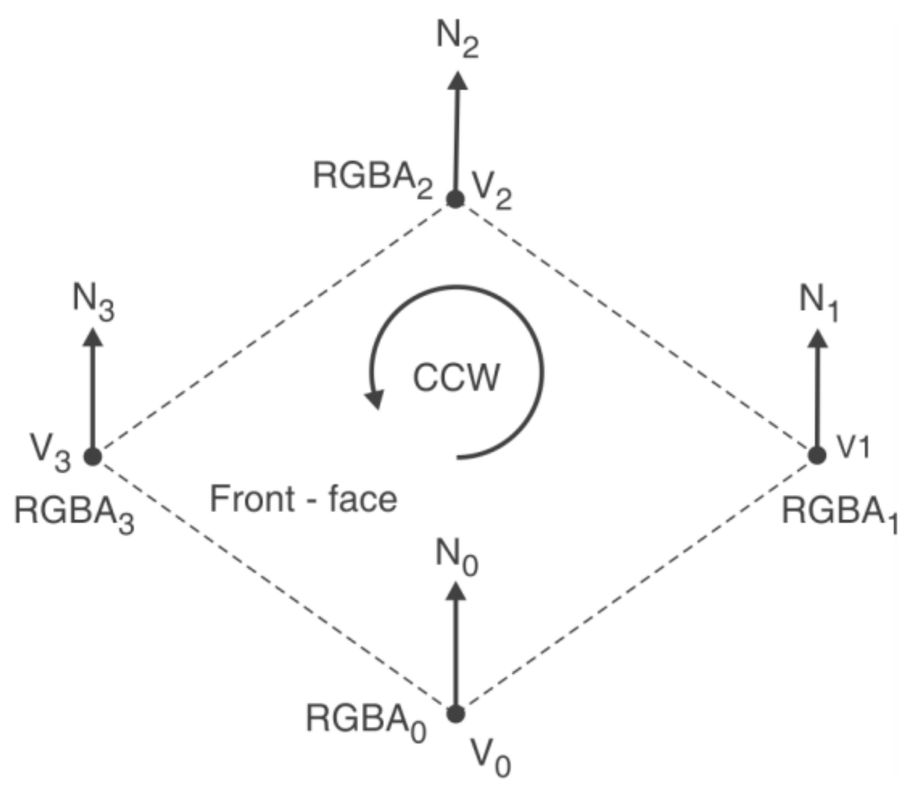


Fig. 2.4.2 : A quad formed by 4 vertices(with color and normal)

- As an example, the following OpenGL code segment specifies a color-cube, center at the origin.
- To construct a geometric object or model, we use a pair of glBegin (Primitive Type) and glEnd() to enclose the vertices that form the model. For primitive Type that ends with 'S' (e.g., GL_QUADS), multiple shapes of the same type can be defined.
- Each of the 6 faces is a primitive quad (GL_QUAD). first set the color via glColor3f(*red*, *green*, *blue*). This color would be applied to all succeeding vertices until it is overridden. The 4 vertices of the quad are specified via glVertex3f(*x*, *y*, *z*), in counter-clockwise manner such that the surface-normal is pointing outwards, indicating its front-face. All four vertices has this surface-normal as its vertex-normal.

```
glBegin(GL_QUADS);              // of the color cube
    // Top-face
glColor3f(0.0f, 1.0f, 0.0f);    // green
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, 1.0f);

    // Bottom-face
glColor3f(1.0f, 0.5f, 0.0f);    // orange
glVertex3f(1.0f, -1.0f, 1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(1.0f, -1.0f, -1.0f);

    // Front-face
glColor3f(1.0f, 0.0f, 0.0f);    // red
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, 1.0f);

    // Back-face
glColor3f(1.0f, 1.0f, 0.0f);    // yellow
glVertex3f(1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(1.0f, 1.0f, -1.0f);

    // Left-face
glColor3f(0.0f, 0.0f, 1.0f);    // blue
glVertex3f(-1.0f, 1.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, -1.0f);
glVertex3f(-1.0f, -1.0f, 1.0f);
```

Technknowledge Publication

```
    // Right-face
glColor3f(1.0f, 0.0f, 1.0f);          // magenta
glVertex3f(1.0f, 1.0f, -1.0f);
glVertex3f(1.0f, 1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, 1.0f);
glVertex3f(1.0f, -1.0f, -1.0f);

glEnd();                              // of the color cube
```

2.4.1 Indexed Vertices

- Primitives mostly share vertices. Instead of repeatedly specifying the vertices, it is more effective to create an index list of vertices, and use the indexes in specifying the primitives.
- For example, the following code fragment specifies a pyramid, which is formed by 5 vertices. first define 5 vertices in an index array, followed by their respective color. For each of the 5 faces, we simply provide the vertex index and color index.

```
float[ ] vertices = {          /* 5 vertices of the pyramid
                                  in (x,y,z)*/
    -1.0f, -1.0f, -1.0f,       // 0. left-bottom-back
     1.0f, -1.0f, -1.0f,       // 1. right-bottom-back
     1.0f, -1.0f,  1.0f,       // 2. right-bottom-front
    -1.0f, -1.0f,  1.0f,       // 3. left-bottom-front
     0.0f,  1.0f,  0.0f        // 4. top
};

float[ ] colors = {            /* Colors of the 5 vertices in
                                  RGBA */
    0.0f, 0.0f, 1.0f, 1.0f,    // 0. blue
    0.0f, 1.0f, 0.0f, 1.0f,    // 1. green
    0.0f, 0.0f, 1.0f, 1.0f,    // 2. blue
    0.0f, 1.0f, 0.0f, 1.0f,    // 3. green
    1.0f, 0.0f, 0.0f, 1.0f     // 4. red
};

byte[ ] indices = {            /* Vertex indices of the
                                  4 Triangles */
    2, 4, 3,                   // front face (CCW)
    1, 4, 2,                   // right face
    0, 4, 1,                   // back face
    4, 0, 3                    // left face
};

/* Transfer the arrays to vertex-buffer, color-buffer and
index-buffer. */
// Draw the primitives (triangle) from the index buffer
```

The OpenGL interface

GL (Graphics Library): 'basic' OpenGL, contains only basic primitives.

- **GLU (Graphics Utility Library) :** Contains richer set of primitives, e.g. spheres, and only calls GL.
- **GLUT (GL Utility Toolkit) :** For interfacing with the window system and input devices primitive, attribute, viewing, and transformation functions are handled by GL and GLU. Input and control functions are handled by GLUT.

2.5 3D Graphics Coordinate Systems

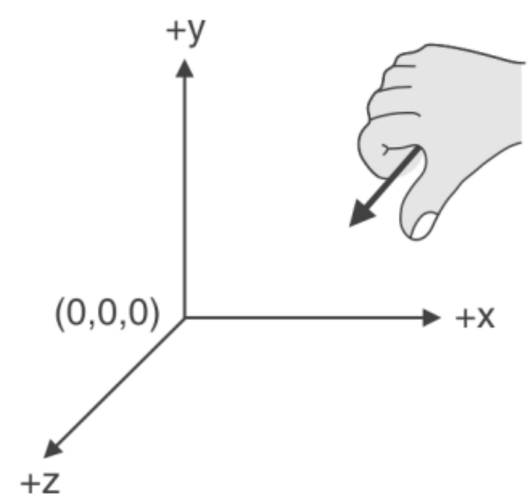


Fig. 2.5.1 : Right - hand system(RHS) or counter-clockwise(CCW) system

Technical Knowledge Publication

- OpenGL adopts the *Right-Hand Coordinate System* (RHS). In the RHS, the x-axis is points to right, y-axis is points up, and z-axis is points out of the screen. With our right-hand fingers curving from the x-axis towards the y-axis, the thumb is pointing at the z-axis. RHS is *counter-clockwise* (CCW). The 3D Cartesian Coordinates is a RHS.
- Some graphics software (such as Microsoft Direct3D) use *Left-hand System* (LHS), where the z-axis is inverted. LHS is clockwise (CW). we shall adopt the RHS.

2.5.1 Pixel vs. Fragment

- Pixels refer to the dots on the display screen, that are aligned in a 2-d grid of a certain rows and columns corresponding to the resolution of display. A pixel is 2-dimensional, with a (x, y) position and a RGB color value (there is no alpha value for pixels). The aim of the Graphics Rendering Pipeline is to produce the color-value for all the pixels to be displayed on the screen, given the input primitives.

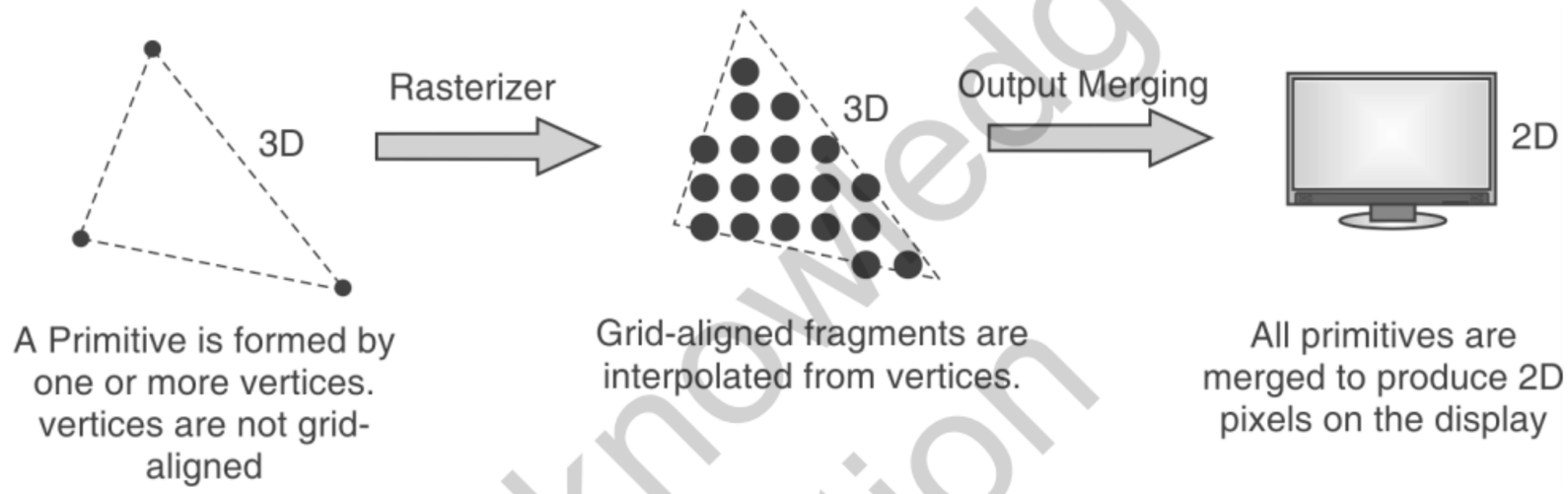


Fig. 2.5.2 : Vertex, primitives, fragments and pixels

- To produce pixels that are aligned to the display grid, the graphics rasterizer takes each input primitive and perform *raster-scan* to produce a set of grid-aligned fragments enclosed within the primitive. **A fragment** is 3-dimensional, with a (x, y, z) position. The (x, y) corresponds to the 2D pixel-grid. The value of z (not aligned with the grid) indicates its depth.
- The z-values are needed to capture the relative depth of various primitives, so that the blocked objects can be discarded (or the alpha channel of transparent objects processed) in the output-merging stage. Fragments are produced via *interpolation* of the vertices.
- Thus, a fragment has all the vertex's attributes such as color, fragment-normal and texture coordinates. In modern GPU, vertex processing and fragment processing are programmable. The programs are called vertex shader and fragment shader.

2.6 Simple Modelling and Rendering of Two and Three-Dimensional Geometric Objects

Every OpenGL geometric primitive is specified by its vertices, which are homogenous coordinates. Homogenous coordinates are of the form (x, y, z, w). Depending on how vertices are organized, OpenGL can render any of the shown primitives.

Example

```
void drawRhombus( GLfloat color[] )
{
    glBegin( GL_QUADS );
    glColor3fv( color );
    glVertex2f( 0.0, 0.0 );
```

Tech Knowledge Publications

```
    glVertex2f( 1.0, 0.0 );
    glVertex2f( 1.5, 1.118 );
    glVertex2f( 0.5, 1.118 );
    glEnd();
}
```

The drawRhombus( )function enables OpenGL to render a single quadrilateral in a single color. The rhombus is planar(2 dimmensional), since the z value is automatically set to 0.0 by glVertex2f().

Color in OpenGL

- OpenGL supports two colors models: **RGB,** or **RGBA, mode** and **color-index**
- In RGB mode, every color is represented in a triplet form of red, green, and blue values. The eye blends these primary colors, forming the color that we see. If real numbers are used to specify colors, 0.0 is none of a primary, and 1.0 represents the maximum amount of that primary color.
- Example(1.0, 0.0, 0.0) is a bright red, (0.5, 0.5, 0.0) is a dark yellow, (1.0, 1.0, 1.0) is white, and (0.0, 0.0, 0.0) is black.
- In RGBA mode, a fourth color component is used, A or alpha, that is an opacity. An opacity of 1.0 means that the color is opaque and cannot be "seen through," whereas a value of 0.0 means that a color is transparent. If an integer value is used to specify a color, the range is from 0 to the maximum value of the type. Example, we use unsigned bytes, the color values are from 0 to 255.
- In color-index mode, colors are specified as indices into a table of red, green, and blue values. In this mode, a table is formed of the allowed colors, usually with 256 possible colors. This mode is not the common one used at present and also requires more detailed interaction with the windowing system than does RGB color. Hence, generally RGB or RGBA color mode is used.
- In OpenGL, colors become part of the state . The present drawing color, is set by glColor*( ), and a present clear color, set by glClearColor( ). These colors remain the same until it is changed in the application program. Thus, colors are not attached to objects but rather to the internal state of OpenGL. The color used to render an object is the present color. While coding, it may appear that colors are associated with objects and their vertices. But in fact, OpenGL uses the present state to find the color at the time the program defines a vertex. With glColor*( ), we can set either RGB or RGBA colors, using the standard C data types. OpenGL has only one internal form for the present color, which is in RGBA form. Using glColor3*( ) is the same as using RGBA color with the alpha value set to 1.0. The clear color specified by glClearColor( ) must be stated as an RGBA color, using floats in the range (0.0, 1.0) and values of type GLclampf.

Differences in Coordinate System of GLUT and OpenGL

- OpenGL uses a variety of coordinates systems. Generally, users describe their geometry in terms of world coordinates. For two-dimensional applications, this coordinate system has the positive x values increasing to the right and the positive y values increasing as we go up. Thus, if the origin is at the bottom-left corner of the page, all the locations the page would have positive x and y values. Most windowing systems use a system in which the values of y increase as we move downwards. In such a system, if we require all x and y values to be positive, the origin is put in the top-left corner.
- In most windowing systems, the screen is displayed from top to bottom, and the counting of rows and columns starts from the top left corner. As it interacts with the window system, GLUT uses the second form, and we think of the origin of the screen as being in its top-left corner and the locations of the pixels as numbered from (0, 0) going down and to the right. For such functions as glutInitWindowPosition(), is used.

Techknowledge Publication

2.6.1 Transformations in OpenGL

The Viewing Transformation Pipeline

- Viewing Transformation is the mapping of coordinates of points and lines that form the picture or an object into appropriate coordinates on the display device.
- The conversion of 3D points into a 2D coordinates on the screen is done using matrix transformations.

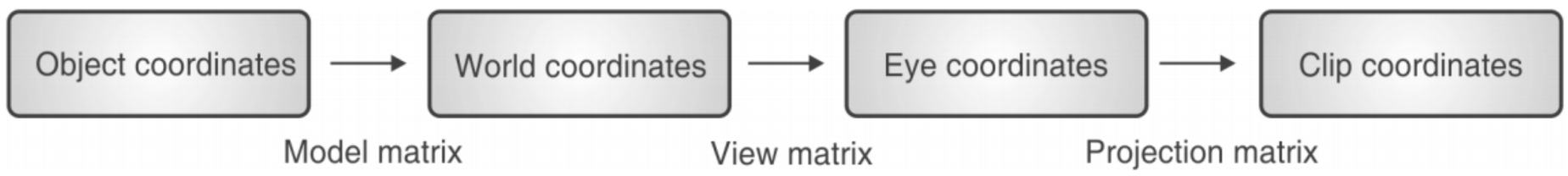


Fig. 2.6.1 : Viewing transformation pipeline

- Each transformation transforms a vector into a new coordinate system, thus moving to the next step.

Model Matrix

- The matrix that contains every translations, rotations or scaling, applied to an object is named as the model matrix in OpenGL
- The model matrix transforms a position in a model to the position in the world coordinates. This position is affected by the position, scale and rotation of the model that is being drawn. It is generally a combination of the simple transformations. If vertices are specified in world coordinates (common when drawing a simple test scene), then this matrix can simply be set to the identity matrix.

View Matrix

- In real life we are used to moving the camera to change the view of a certain scene, in OpenGL it's the other way around. The camera in OpenGL cannot move and is defined to be located at(0,0,0) facing the negative Z direction. It means that instead of moving and rotating the camera, the world is moved and rotated around the camera to construct the appropriate view.
- The view matrix in OpenGL controls the way we look at a scene. It is uniquely determined by 3 parameters :
  - The eye, or the position of the viewer;
  - The center, or the point where we the camera aims;
  - The up, which defines the direction of the up for the viewer. The up direction is parallel to the y axis and in the positive sense of y.
- The defaults in OpenGL are: the eye at (0, 0, -1); the center at (0, 0, 0) and the up is given by the positive direction of the y axis (0, 1, 0).

Projection Matrix

- By default, in OpenGL, an object appears to have the same size irrespective of camera position. This is against our day to day experience, where an object closer to us (to the camera) appears larger than an object that is at a greater distance. Imagine the way a ball approaches us, the ball will appear bigger and bigger as it gets closer to your eyes.
- Another drawback with the defaults from OpenGL is that in order to actually see (draw) something on the screen, the objects that we want to draw must be inside the unit cube. Any portion of the scene (or object) that is outside the unit cube will be clipped.

Technknowledge Publications

– The clipping volume can be virtually increased or decreased with the projection matrix. when the clipping volume has the shape of a parallelepiped, or a box, we say that we use an orthographic projection. We know that the orthographic projection does not modify the size of the objects no matter where the camera is positioned.

The orthographic projection matrix :

$$P = \begin{bmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -\frac{\text{right} + \text{left}}{\text{right} - \text{left}} \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -\frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\ 0 & 0 & -\frac{2}{\text{far} - \text{near}} & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– Another projection matrix, that can enhance the feeling of real world is the perspective projection matrix, in this case the volume is a frustum and not a parallelepiped.

The Fig. 2.6.2 shows perspective viewing volume and the Frustum( ) parameters

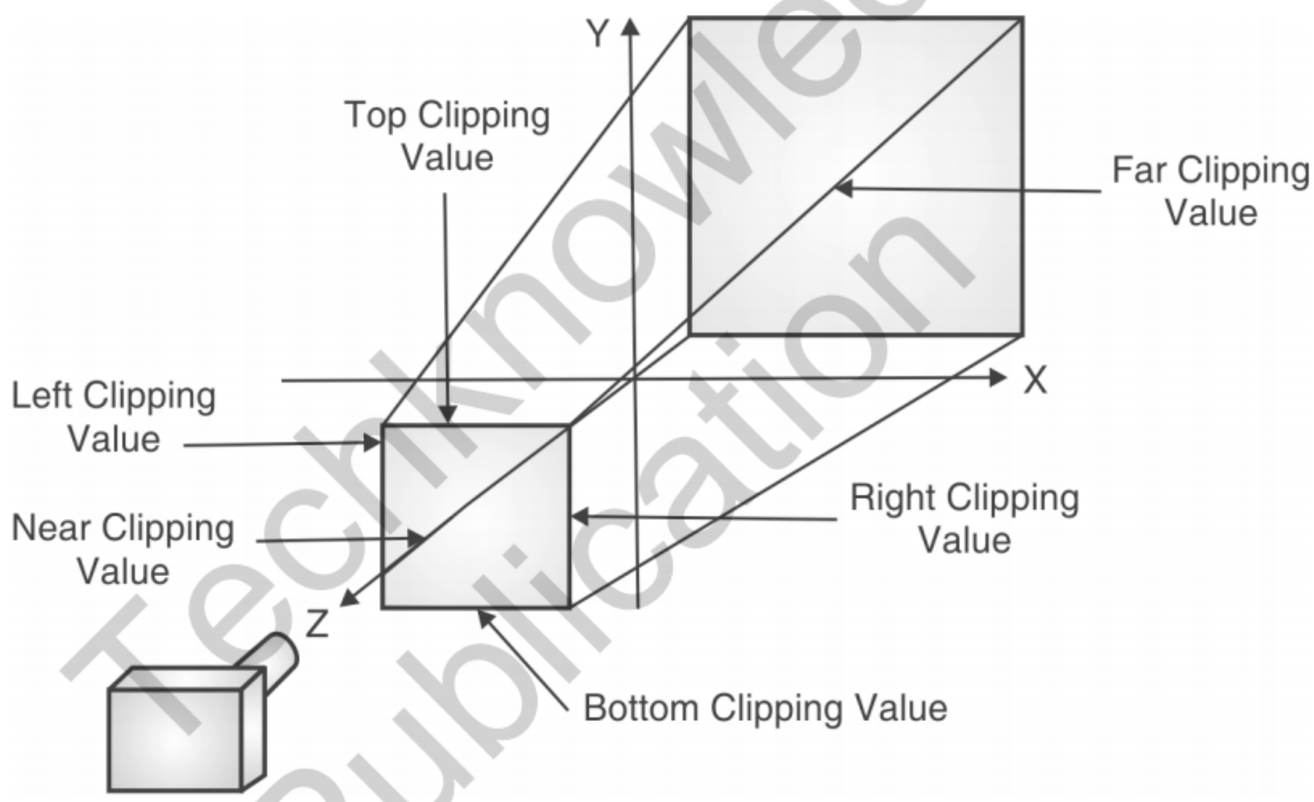


Fig. 2.6.2 : Perspective viewing volume

The perspective projection matrix

$$P = \begin{bmatrix} \frac{2\ \text{near}}{\text{right} - \text{left}} & 0 & \frac{\text{right} + \text{left}}{\text{right} + \text{left}} & 0 \\ 0 & \frac{2\ \text{near}}{\text{top} - \text{bottom}} & \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} & 0 \\ 0 & 0 & -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & -\frac{2\ \text{far near}}{\text{far} - \text{near}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

– The perspective projection matrix is usually specified through four parameters :
  - viewing angle or field of view (usually abbreviated as FOV);
  - aspect
  - near
  - far

- Where near and far represent the positions of the near and the far clipping planes.
  - top = near · tan(π180 · FOV/2)
  - bottom = –top
  - right = top · aspect
  - left = – right

Field-Of-View(FOV)

- The Field-of-view(FOV) is the angle between the top and bottom of the 2D surface on which the world will be projected. Zooming in games is often accomplished by decreasing this angle rather than moving the camera closer.

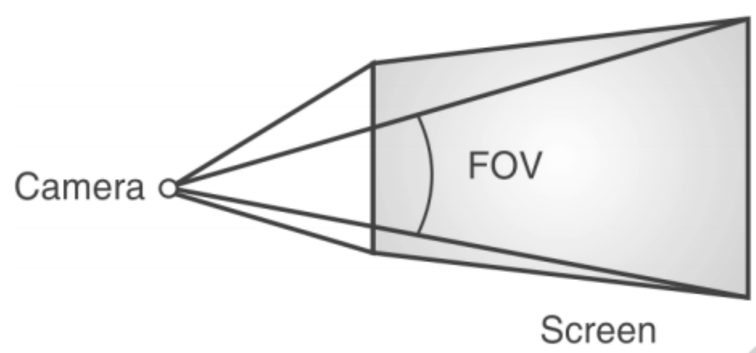


Fig. 2.6.3 : FOV

- The near and far planes are known as the clipping planes. Any vertex closer to the camera than the near clipping plane and any vertex farther away than the far clipping plane is clipped.

2.6.2 2-Dimmensinal Viewing

- OpenGL has two types of things that it can render: geometric primitives and image primitives. Geometric primitives are points, lines and polygons. Image primitives are bitmaps and graphics images.
- In OpenGL, programs written in two-dimension are a special case of three-dimensional programs. Two dimensional objects have a coordinates in the of form (x, y), but from OpenGL view, these are three values (x, y, z) values, with z set to 0.
- The basic model used in viewing is called the synthetic camera model. It creates a match between the viewer-observer, the photographer creates an image of the things and things we do on the computer to produce an image. In two dimensions the objects can be defined either by specifying or calculating a set of vertices using combination of glVertex*( ), glBegin( ), and glEnd( ) in our program.
- Assuming our code as a description of objects put up on a piece of paper. The view step determines which part of that paper is seen by synthetic camera and the object appears on the screen. If we assume that the camera is aligned with the x and y axes, we need to only specify a rectangular region through maximum and minimum values of x and y. This can be done using gluOrtho2D( ).

2.6.3 3D Geometric Transformations in OpenGL

- The transformations are an important part of 2D and 3D graphics. There are three basic types of transformation scaling, rotation, and translation in 2D as well as in 3D with few changes. Translation allows to change the position by adding some integers to each coordinate. Translation in 3D is similar to translation in 2D. In case of 3D we have an additional z coordinate. to specify the amount of motion in the direction of each of the coordinate axes. A translation by (tx, ty, tz) transforms the point (x, y, z) to (x+tx, y+ty, z+tz). (tx, ty, tz) are translation factors in respective(x,y,z) direction.
- Similarly 3D scaling transformation has scaling factors (sx, sy, sz) which specifies by how much to increase/decrease the size of object with respect to the coordinate axes. To define the direction of rotation in 3D, we use the right-hand rule: Point the extended thumb of your right hand in the direction from (0,0,0) to (dx, dy, dz), where (dx, dy, dz) is the point that specifies the axis of rotation. Then the direction of rotation for positive angles is given by the direction in which your fingers curl.

Technknowledge Publication

Geometric Transformations Functions in OpenGL

– The functions for performing geometric transformations in OpenGL are glScalef( ), glRotatef( ), and glTranslatef( ). The "f" indicates that the parameters are of type float. There are also versions ending in "d" to indicate parameters of type double. As you would expect, 3D scaling and translation require three parameters. Rotation requires four parameters.

– The first is the angle of rotation measured in degrees (not radians), and the other three parameters specify the axis of rotation. In the command glRotatef(angle, dx, dy, dz), the rotation axis is the line through the points (0, 0, 0) and (dx, dy, dz). At least one of dx, dy, and dz should be non-zero. Here are a few examples :

Table 2.6.1

Commands	Description
glTranslatef(1, 0, 0);	Move 1 unit in the positive x-direction
glTranslatef(3, 5, –7.5);	Move (x, y, z) to (x + 3, y + 5, z – 7.5)
glScalef(2, 2, 2);	Uniform scaling by a factor of 2(Increse in size).
glScalef(1, 0.5, 1);	(Decrease in size) Shrink by half in the y-direction only.
glScalef(0, 0, –1);	Reflect through the xy-plane.
glRotatef(90, 1, 0, 0);	Rotate 90 degrees about the x-axis. Moves the +y axis onto the +z axis and the +z axis onto the -y axis.
glRotatef(90, 0, 1, 0);	Rotate 90 degrees about the y-axis. Moves the +z axis onto the +x axis, Moves the +z axis onto the +x axis
glRotatef(30, 1.5, 2, –3);	Rotate 30 degrees about the line through the points (0,0,0) and (1.5,2,–3).

– **glLoadIdentity( ) :** The function glLoadIdentity() can be called to discard any transformations that have been applied. It takes no parameters. OpenGL has direct support for matrix stacks.

– **glPushMatrix( ) :** This function saves a copy of the current transformation on the matrix stack.

– **glPopMatrix( ) :** This function restores a transformation that was saved by glPushMatrix( ) by removing the top transformation from the stack and installing it as the current transformation.

Projection Transformation

Projection transformation maps the 3D world on a 2D viewing surface. OpenGL keeps the projection transformation separate from the modelview transformation. The projection transformation is always applied at the last. It is done using glMatrixMode() function.

```
glMatrixMode( GL_PROJECTION );
```

– To go back to working on the modelview transformation, call

```
glMatrixMode( GL_MODELVIEW );
```

– It's common to leave the matrix mode set to GL_MODELVIEW, except when actually working in another mode. The code shown can be used to set up the projection:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho( xmin, xmax, ymin, ymax, zmin, zmax );
glMatrixMode(GL_MODELVIEW);
```

– The glOrtho command does the work. The parameters specify the ranges of x, y, and z that will be visible in the drawing area, and the projection is done by discarding the z-coordinate.

Technknowledge Publication

2.7 GLUT

- GLUT stands for OpenGL Utility Toolkit. Mark J. Kilgard implemented it to enable the construction of OpenGL applications that are truly window system independent.
- Most of the system dependent actions required to display a window, put OpenGL graphics in it, and accept mouse and keyboard input are handled by the OpenGL Utility Toolkit (GLUT).
- GLUT provides advanced utilities to simplify OpenGL programming, specifically in interacting with the Operating System (such as creating a window, handling key and mouse inputs)
- **glutInit :** initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main( ).

void **glutInit**(int **argc*, char ***argv*)

- **glutCreateWindow :** creates a window with the given title.

int **glutCreateWindow**(char **title*)

- **glutInitWindowSize :** specifies the initial window width and height, in pixels.

void **glutInitWindowSize**(int *width*, int *height*)

- **glutInitWindowPosition :** positions the top-left corner of the initial window at (x, y). The coordinates (x, y), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.

void **glutInitWindowPosition**(int *x*, int *y*)

- **glutDisplayFunc :** registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display( ) as the handler.

void **glutDisplayFunc**(void (**func*)(void))

- **glutMainLoop :** enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as re-paint), and trigger respective event handlers (such as display( )).

void **glutMainLoop**()

2.8 Interaction

2.8.1 Reading the Mouse Position

```
// Get mouse position
int xpos, ypos;
glfwGetMousePos(&xpos, &ypos);
```

- Care should be taken to put the cursor back to the center of the screen, or it might go outside the window.

```
//Reset mouse position for next frame
glfwSetMousePos(1024/2, 768/2);
```

- Notice that the above code assumes that the window is 1024 × 768, or You can use glfwGetWindowSizetoo.

- Now to compute our viewing angles :

```
// Compute new orientation
horizontalAngle += mouseSpeed * deltaTime * float(1024/2 – xpos );
verticalAngle   += mouseSpeed * deltaTime * float( 768/2 – ypos );
```

- 1024/2 – xpos means how far is the mouse from the center of the window. The bigger this value, the more we want to turn.

Technknowledge Publication

- float(…) converts it to a floating-point number so that the multiplication goes well.
- mouseSpeed is just there to speed up or slow down the rotations. Fine-tune this at will, or let the user choose it.
- += : If you didn't move the mouse, 1024/2 – xpos will be 0, and horizontalAngle += 0 doesn't change horizontalAngle. If you had a "=" instead, you would be forced back to your original orientation each frame.

2.8.2 Handling Keyboard Inputs with GLUT

- Callback functions to handle keyboard inputs for normal and special keys, respectively.
- **glutKeyboardFunc :** registers callback handler for keyboard event.

```
void glutKeyboardFunc (void (*func)(unsigned char key, int x, int y)
  // key is the char pressed, e.g., 'a' or 27 for ESC
  // (x, y) is the mouse location in Windows' coordinates
```

- **glutSpecialFunc :** registers callback handler for special key (such as arrow keys and function keys).

```
void glutSpecialFunc (void (*func)(int specialKey, int x, int y)
  /* specialKey: GLUT_KEY_* (* for LEFT, RIGHT, UP, DOWN, HOME, END, PAGE_UP,
    PAGE_DOWN,F1,...F12).*/
  // (x, y) is the mouse location in Windows' coordinates
```

2.9 Callback Function

- A callback function is a function which the library (GLUT) calls when it needs to know how to proccess something. Examle, when glut gets a key down event it uses the glutKeyboardFunc callback routine to find out what to do with a key press. glutKeyboardFunc and glutSpecialFunc are both callback function.
- Number of callbacks is supported by GLUT that respond to events. The three types of callbacks : window, menu, and global.
- **Window callbacks :** It indicates when to redisplay or reshape a window, when the visibility of the window changes, and when input is available for the window.
- **Menu callback :** It is set by the glutCreateMenu.
- The calling order of callbacks between different windows is undefined.
- Callbacks for input events should be delivered to the window the event occurs in. Events should not propagate to parent windows.

Techknowledge Publication

Callback Handler display()

- The function display( ) is known as a callback event handler. An event handler provides the response to a particular event (such as key-press, mouse-click, window-paint). The function display( ) is meant to be the handler for window-paint event. The OpenGL graphics system calls back display( ) in response to a window-paint request to re-paint the window (e.g., window first appears, window is restored after minimized, and window is resized). Callback means that the function is invoked by the system, instead of called by the user program.
- The display( ) runs when the window first appears and once per subsequent re-paint request. Observe that we included OpenGL graphics rendering code inside the display( ) function, so as to re-draw the entire window when the window first appears and upon each re-paint request.

```
glutReshapeFunc(change_viewport);        // when window is resized
glutDisplayFunc(render);                 // when window needs to be drawn
glutIdleFunc(animate);                   // when there is nothing else to do
glutMouseFunc(callback_mouse_button);    // when mouse is clicked
glutKeyboardFunc(keyboard_down);         // when a key is down
glutKeyboardUpFunc(keyboard_up);         // when the key goes up
glutPassiveMotionFunc(look);             // when mouse moves
glutMotionFunc(look);                    // when mouse drags around
```

Technical Knowledge Publication

Review Questions

Q. 1 Enlist the command to set the color of an OpenGL display window to dark gray and white color?

Q. 2 Explain what is meant by the term “OpenGL display callback function”.

Q. 3 Describe OpenGL architecture with block diagram.

Q. 4 Write short note on “Handling Keyboard Inputs with GLUT”.

Q. 5 What are the advantages of OpenGL over other API’s

Q. 6 Explain Rendering Pipeline with respect to OpenGL.

Q. 7 What do you understand by Texture mapping and what are the basic steps involved in it.

Q. 8 State the difference between glColor3f() and glClearColor().

Q. 9 What is the difference between color index mode and RGBA mode.

Q. 10 How to transform the objects around a fixed coordinate system rather than the object\'s local coordinate system?

❑❑❑

CHAPTER 3

UNIT I

Scan Conversion

Syllabus

Scan conversion : Line drawing algorithms : Digital Differential Analyzer (DDA), Bresenham. Circle drawing algorithms : DDA, Bresenham, and Midpoint.

3.1 Scan Conversion

3.1.1 Introduction

- **Scan conversion** is the process of representing continuous graphics as a collection of discrete pixels. Each pixel on the monitor screen is either turned *on* or *off* according to the picture definition stored in a frame buffer.
- **Scan conversion** or **scan converting** rate is a video processing technique for changing the vertical / horizontal **scan** frequency of video signal for different purposes and applications. The smallest element to be displayed on the screen is the pixel.
- **Example :**

  To display a point A(100,200) on the screen we use

  putpixel(100,200,15)

  putpixel(100,200,RED)

  where 100 represents x coordinate and 200 represents y coordinate and 15 is the Color. In C++ the syntax for plotting a single point on the screen is shown in above example.
- Scan conversion is widely used in CRT, Flat-panel, TV set and other types of display devices to display.
- Implementation of such algorithm varies from one computer system to others. Some algorithms are implemented in software, some are implemented in firmware, and some are the combination of hardware, software and firmware.
- Algorithms implemented in hardware scan converts the objects quicker than software.
- Application program takes these parameters as input and generates all the pixels representing the shape. This process is called scan conversion.
- All the points on a line can be generated using DDA or Bresenham's algorithm for given endpoints.
- Most of the monitors and television sets support raster scan display. In raster scan devices, picture definition is stored as a discrete pixel value. The entire scene is defined by a set of intensity values for each pixel on the screen.
- Computed pixel values may be a fraction, so it needs to round off to show it on a monitor screen.
- As shown in Fig. 3.1.1, computed pixel values (10.8, 10.8), (12.6, 11.7) and (13.3, 10.4) are rounded to nearest integer values (11, 11), (13, 12) and (13, 10).

TechKnowledge Publication

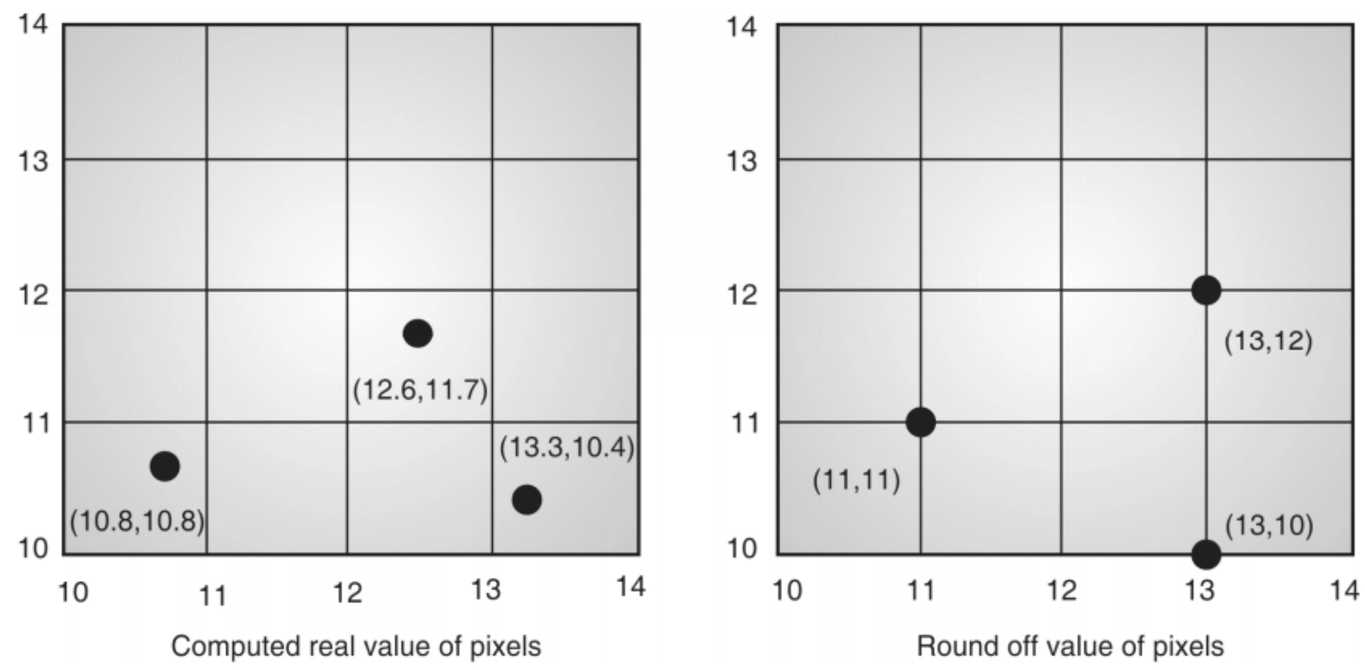


Fig. 3.1.1 : Conversion from actual value to rounded value

3.1.2 Line and Line Segment

– The line is a collection of points along a straight path. Typically, the line is described by its two endpoints (x_1, y_1) and (x_2, y_2). Intermediate points on the line are computed by the line generation algorithm.

– Geometric interpretation of line joining endpoints P_1 and P_2 are depicted in Fig. 3.1.2

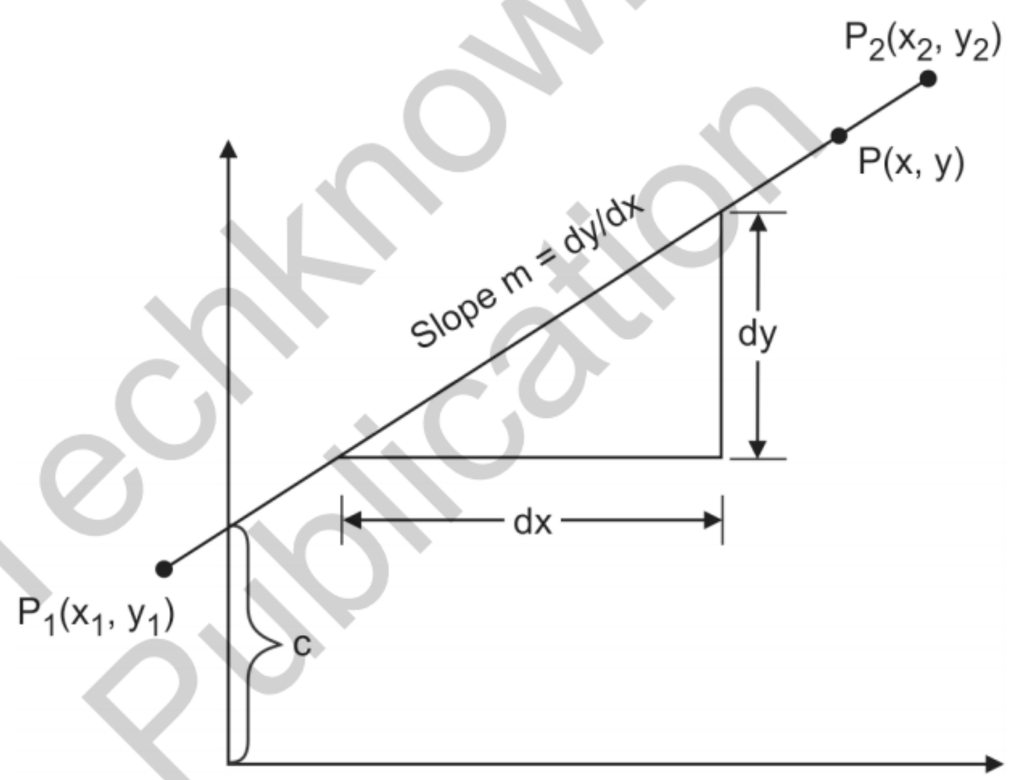


Fig. 3.1.2 : Representation of line

– Given the endpoints $P_1(x_1, y_1)$ and $P_2(x_2, y_2)$, any point P(x, y) on the line can be computed using the slope-intercept formula as follow.

– It is evident from Fig. 3.1.2 that the slope of the line is,

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

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

$$y = mx + c$$

Techknowledge Publication

- A horizontal line has slope 0. The line bisecting the first quadrant has slope 1. As the angle of line with X-axis increases, the slope increases. The slope of the vertical line is ∞.
- Lines with different slopes are depicted in Fig. 3.1.3.

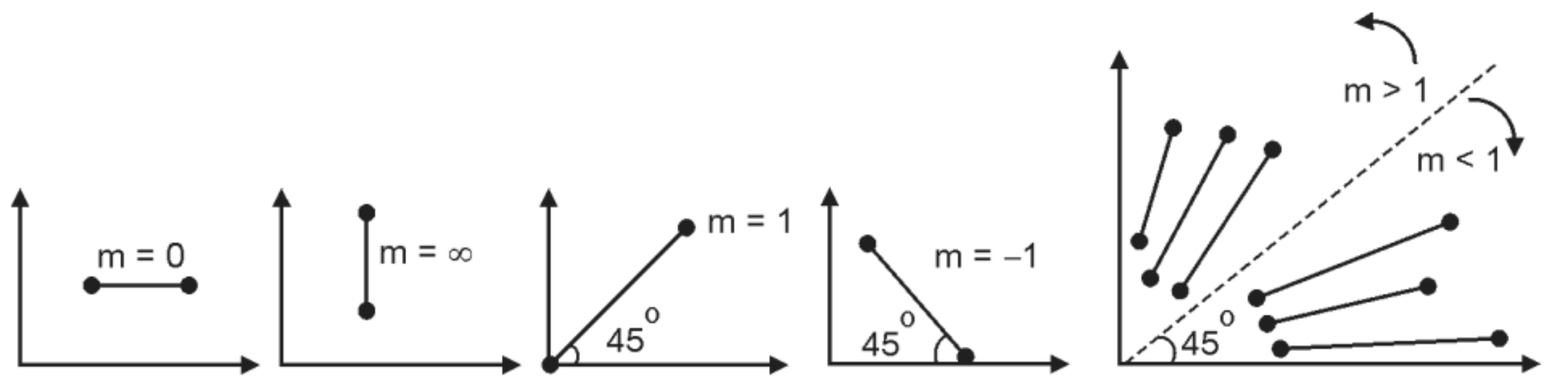


Fig. 3.1.3 : Lines with different slope

- A line segment is the part of the line. Often curves are approximated using small line segments. Almost any shape can be approximated by line segments.
- Like line, the line segment is also bounded by some start and endpoints.

3.1.3 Qualities of Good Line Drawing Algorithms

- When we draw a line on paper, it looks smooth and continuous. But due to the grid structure of monitor, when the line is rasterized, it looks zig-zag except horizontal (m = 0), vertical (m = ∞) and diagonal (m = 1) line.
- Distance between pixels also changes with orientation, and hence the brightness.
- An ideal line should have the following properties :
  - It should interpolate both the endpoints.
  - The brightness of the line should be independent of the orientation of the line.
  - It should appear straight and smooth.
  - It should be drawn quickly.
- All properties are satisfied when we draw lines on paper. But when we rasterize it on the monitor, most of the properties do not hold.

- On the monitor screen, vertical and horizontal lines are displayed with maximum intensity; the brightness of inclined line changes with orientation.

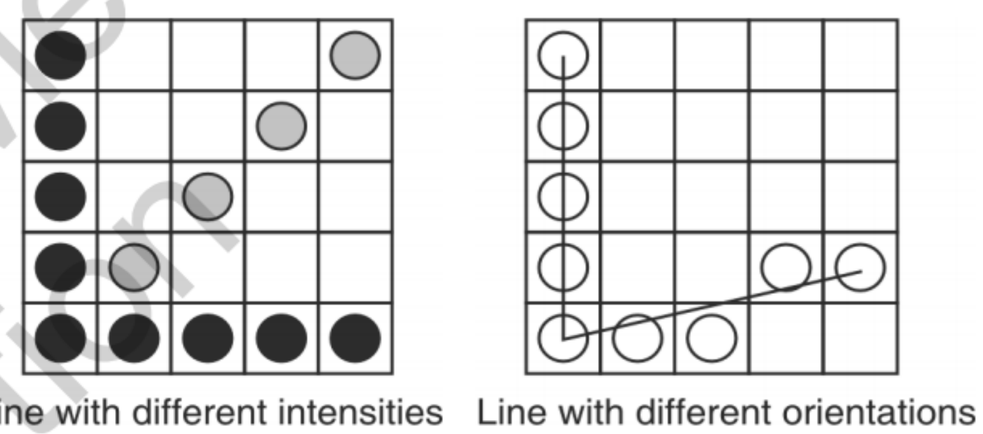


Fig. 3.1.4 : Line with different intensities and orientations

- All properties are satisfied when we draw them on paper. But when we rasterize it on the monitor, most of the properties do not hold.

3.2 Line Drawing Algorithms

- **Line generation** or **scan conversion of the line** is the process of turning '*on*' or '*off*' the pixel along the line path.
- In this section, we will discuss the following line drawing algorithms.
  1. Digital Differential Analyzer
  2. Bresenham's Line Drawing Algorithm

3.2.1 Digital Differential Analyzer (DDA)

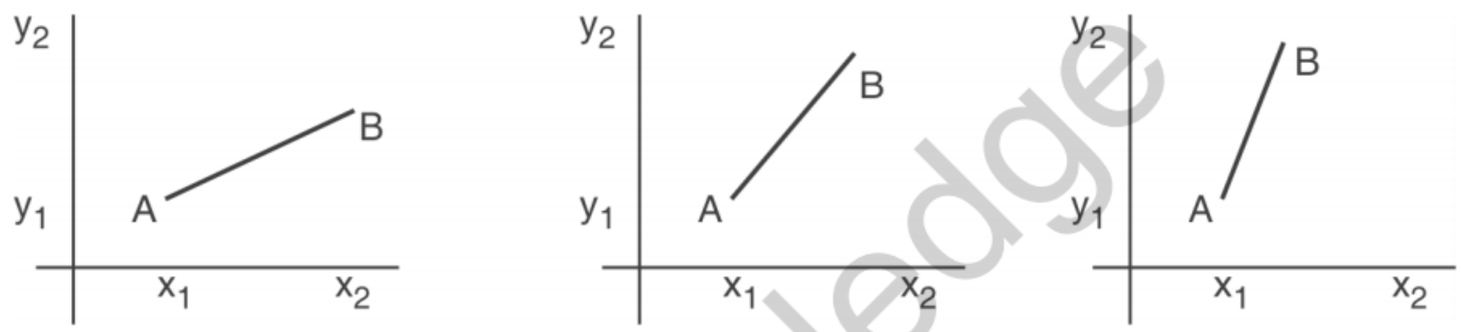
Q. Explain DDA line drawing algorithm with example.

SPPU - Dec. 15, May 17, 6 Marks

TechKnowledge Publications

3.2.1.1 Working Mechanism

- Digital Differential Analyzer (DDA) is a simple, incremental line scan converting algorithm.
- In the DDA algorithm, either horizontal or vertical displacement is set to unit interval and the corresponding displacement for other direction is calculated using the slope.
- If the line makes an angle less than 45° with X-axis (i.e. m < 1), increment in the X-direction is set to 1 and corresponding Y is computed.
- And if line makes an angle greater than 45° with X-axis (i.e. m > 1), increment in the Y-direction is set to 1 and corresponding X is computed.
- For line with slope m = 1, increment in both the directions is set to 1.
- Fig. 3.2.1 shows the lines with different orientations.



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

Fig. 3.2.1 : Lines with different orientations

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

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

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

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

We will discuss three cases.

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

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

Fig. 3.2.2 shows the actual line and the rounded of locations of a pixel on the monitor grid.

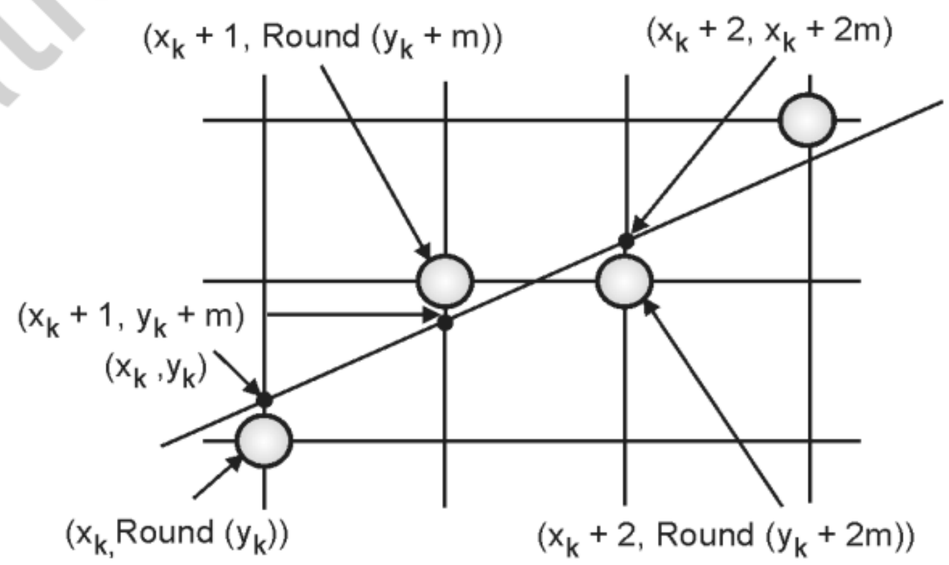


Fig. 3.2.2 : Line with |m| < 1

From equation of slope,

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

$$\therefore \quad \Delta y = m$$

So coordinate value for next pixel is given as,

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

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

Value of m is real number, so y_{k+1} may be real number. Hence, every time after adding m to y_k, answer is rounded to nearest integer.

TechKnowledge Publications

If the line is processed from right to left, the value of x and y decreases in every iteration. So, Δx is set to -1 and Δy is set to $-m$.

In that case, coordinate values for next pixel is given as,

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

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

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

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

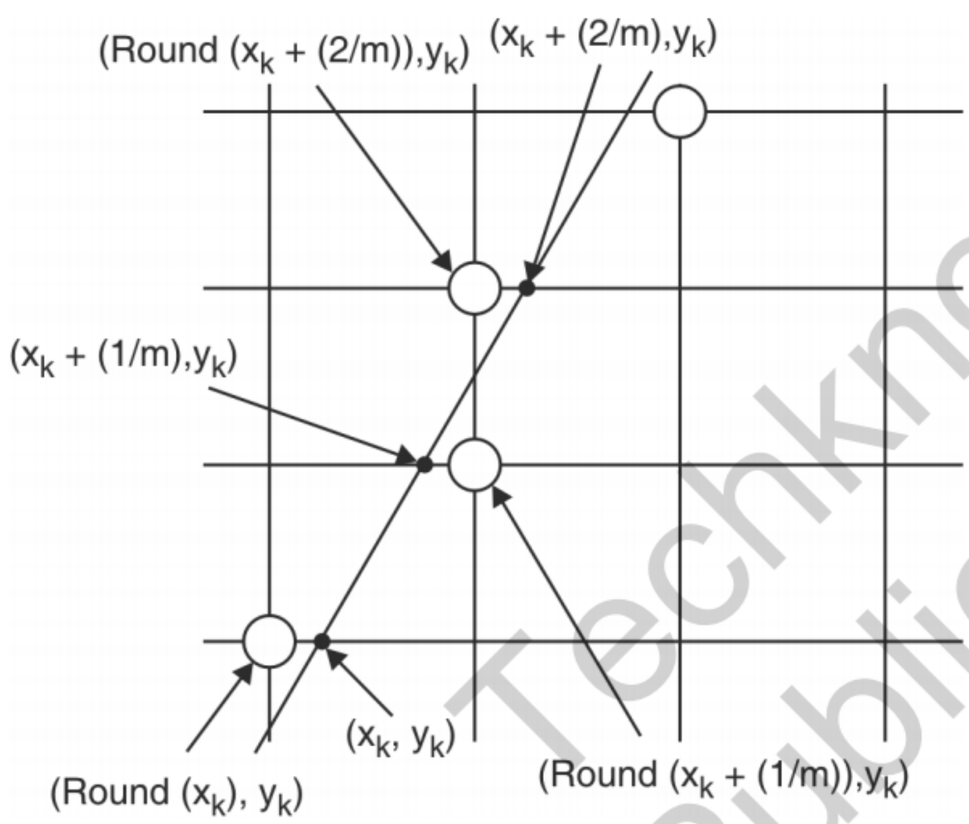


Fig. 3.2.3 : Line with |m| > 1

From equation of slope,

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

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

Coordinate value for next pixel is given as,

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

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

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

In that case, coordinate values for next pixel is given as,

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

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

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

When slope of line is 1, i.e. line makes an angle of 45° with X-axis, we increment both coordinate value by 1.

So, $\Delta x = \Delta y = 1$.

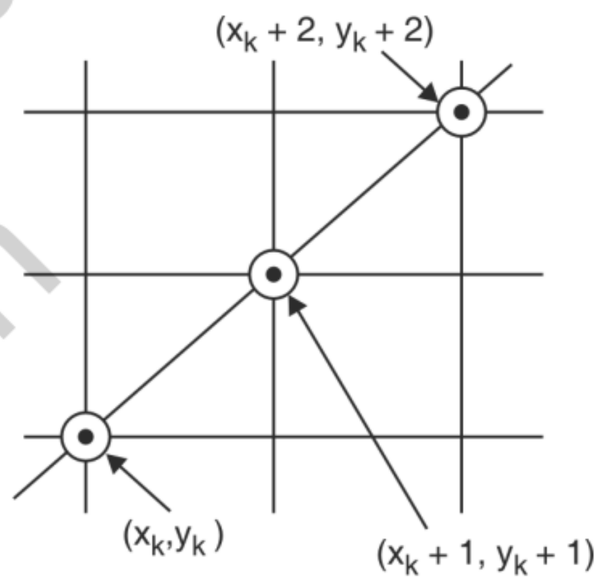


Fig. 3.2.4 : Line with |m| = 1

Coordinate value for next pixel is given as,

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

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

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

In that case coordinate values for next pixel is given as,

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

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

Table 3.2.1 shows the sign and value of x and y increments to draw a line in any quadrant.

Techknowledge Publications

Table 3.2.1 : Generalization of DDA line drawing algorithm

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

3.2.1.2 Algorithm

Steps of DDA Line Drawing Algorithm

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

Step 2 : Compute horizontal and vertical differences as,

$$dx = x_2 - x_1$$
$$dy = y_2 - y_1$$

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

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

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

Step 6 : If |dx| > |dy| and $x_1 \geq x_2$, set increments in X and Y-direction to (–1) and (–m), respectively.

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

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

Note : These steps are to draw a line in the first quadrant – growing from left to right or right to left only. Generalized line algorithm can be derived with the help of increment and sign as shown in Table 3.2.1.

DDA line drawing algorithm

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

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

TechKnowledge Publications

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

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

Program 3.2.1 : Program to implement DDA line drawing algorithm.

```
#include<graphics.h>
#include<iostream.h>
#include<math.h>
#include<dos.h>
#include<conio.h>
void main( )
{
    float x, y, x1, y1, x2, y2, dx, dy, step;
    int i, gd = DETECT, gm;
    initgraph(&gd, &gm, "NULL");
    cout << "Enter the coordinates of first point ( x1
            and y1) : ";
    cin >> x1 >> y1;
    cout << Enter the coordinates of second  point (x2
            and y2): ";
    cin >> x2 >> y2;
    dx = (x2 - x1);
    dy = (y2 - y1);
    if (abs(dx) >= abs(dy))
      step = abs(dx);
    else
      step = abs(dy);
    dx = dx / step;
    dy = dy / step;
    x = x1;
    y = y1;
    i = 1;
    while (i <= step)
    {
    putpixel(x, y, 15);
    x = x + dx;
    y = y + dy;
    i = i + 1;
    delay(200);
    }
    getch();
    closegraph();
}
```

3.2.1.3 Pros and Cons

Advantages and disadvantages of DDA line drawing algorithm are stated here :

Advantages

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

Techknowledge Publications

Disadvantages

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

3.2.1.4 Examples

Ex. 3.2.1 : Calculate the pixel coordinates of line AB using the DDA algorithm. Where, A = (0, 0) and B = (4, 6).

Soln. :

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

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

$$= \frac{6-0}{4-0} = \frac{6}{4} = 1.5$$

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

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

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

Table P. 3.2.1 shows the calculation for required pixels.

Table P. 3.2.1 : Calculation for required pixels

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

In Table P. 3.2.1, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.1.

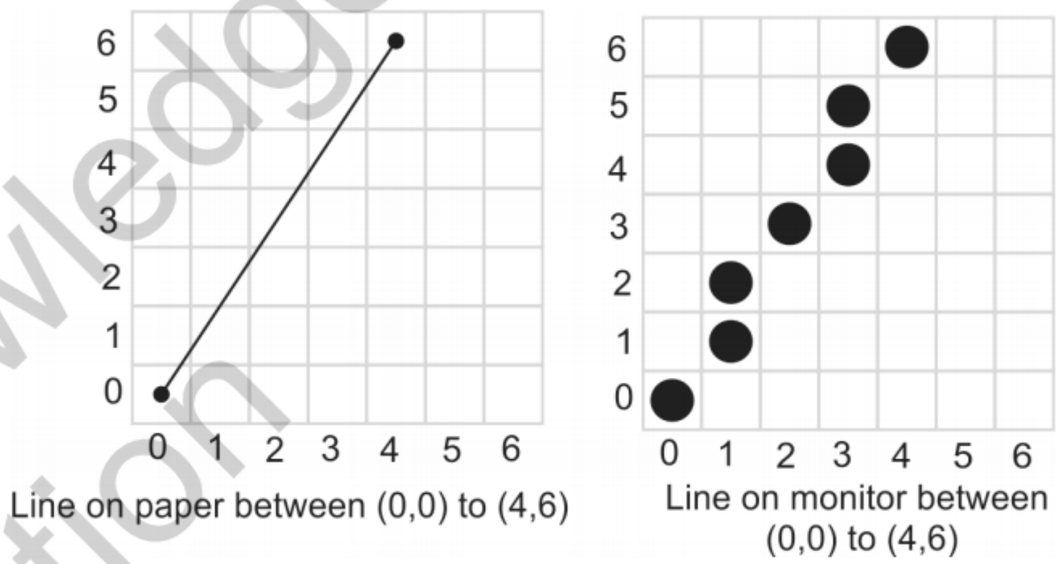


Fig. P. 3.2.1 : DDA line between points (0, 0) and (4, 6)

Ex. 3.2.2 : Draw a line from point (2, 2) to (10, 7) using DDA line drawing algorithm.

Soln. :

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

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

$$= \frac{7-2}{10-2} = \frac{5}{8} = 0.625$$

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

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

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

Technknowledge Publication

Table P. 3.2.2 shows the calculation for required pixels.

Table P. 3.2.2 : Calculation for required pixels

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

In Table P. 3.2.2, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.2.

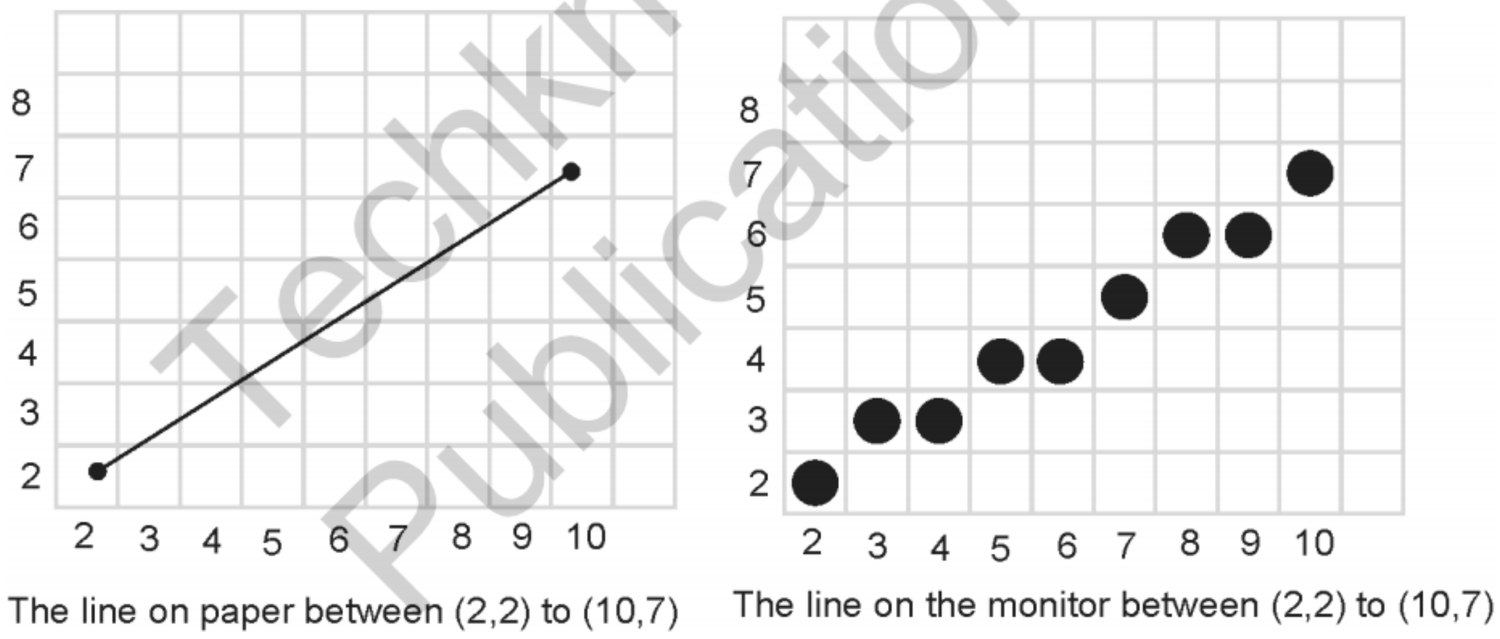


Fig. P. 3.2.2 : DDA line between points (2, 2) and (10, 7)

Ex. 3.2.3 : Draw a line from A(10, 2) to B(6, 8) using DDA line drawing algorithm.

Soln. :

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

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

$$= \frac{8-2}{6-10} = \frac{6}{-4} = -1.5$$

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

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

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

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

Table P. 3.2.3 shows the calculation for required pixels.

Table P. 3.2.3 : Calculation for required pixels

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

In Table P. 3.2.3, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.3.

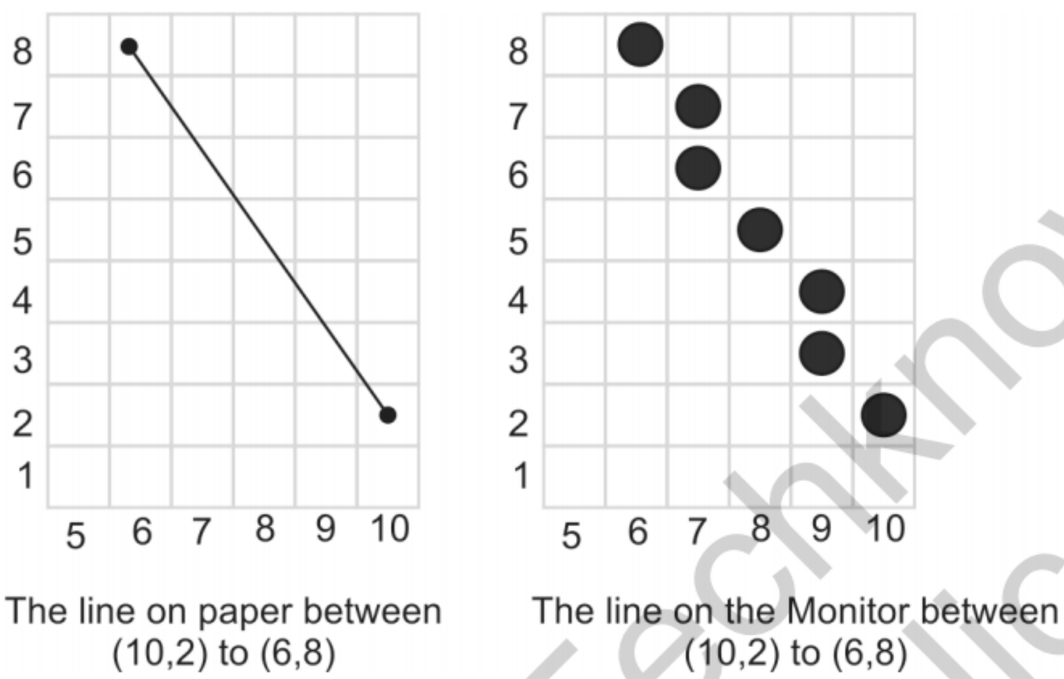


Fig. P. 3.2.3 : DDA line between points (10, 2) and (6, 8)

Ex. 3.2.4 : Draw a line from A(20, 10) to B(30, 18) using DDA line drawing algorithm.

Soln. :

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

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

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

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

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

Table P. 3.2.4 : Calculation for required pixels

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

In Table P. 3.2.4, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.4.

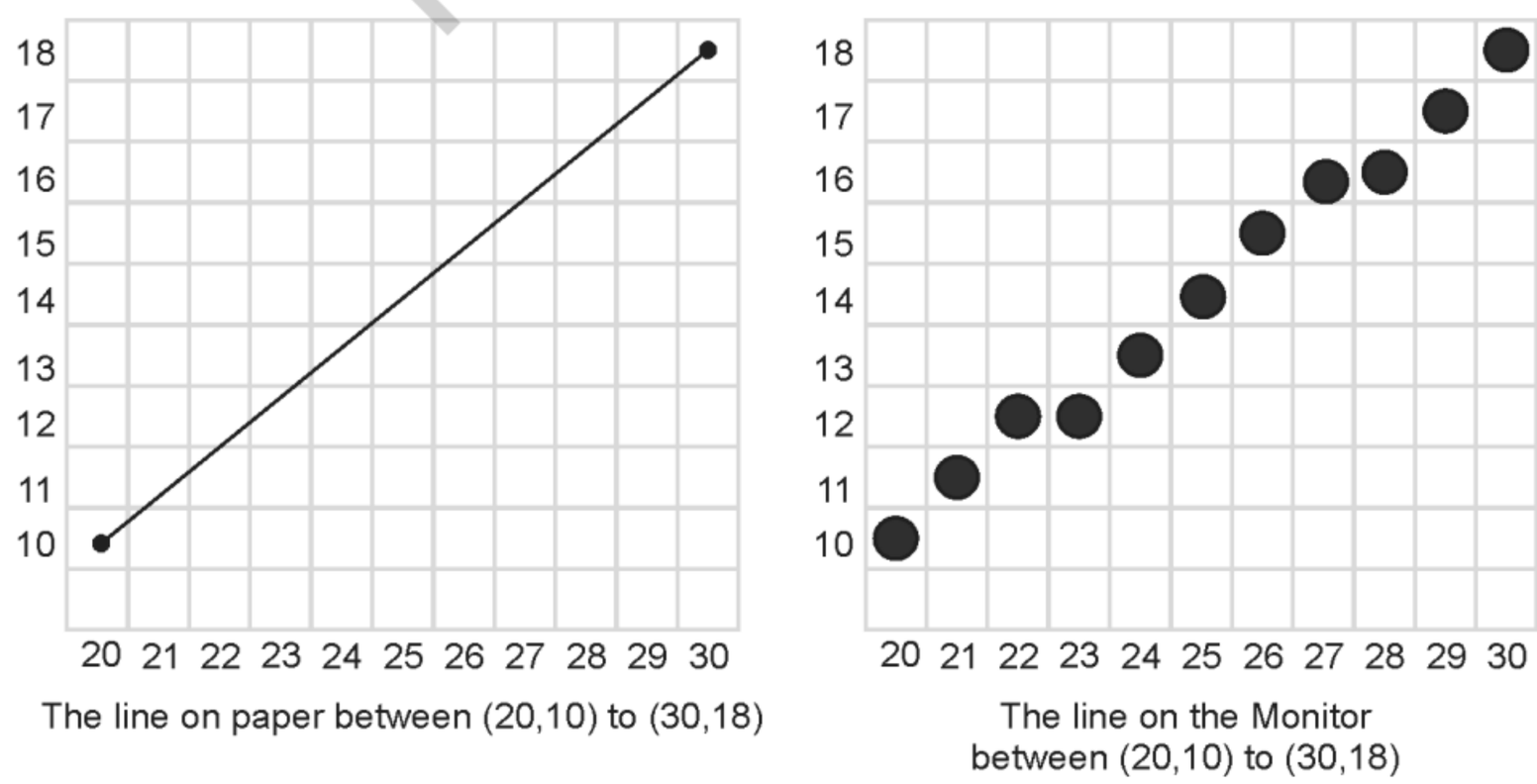


Fig. P. 3.2.4 : DDA line between points (20, 10) and (30, 18)

TechKnowledge Publications

Ex. 3.2.5 : Draw a line from A(0, 0) to B(–5, –5) using the DDA line drawing algorithm.

Soln. :

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

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

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

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

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

Table P. 3.2.5 : Calculation for required pixels

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

In Table P. 3.2.5, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.5.

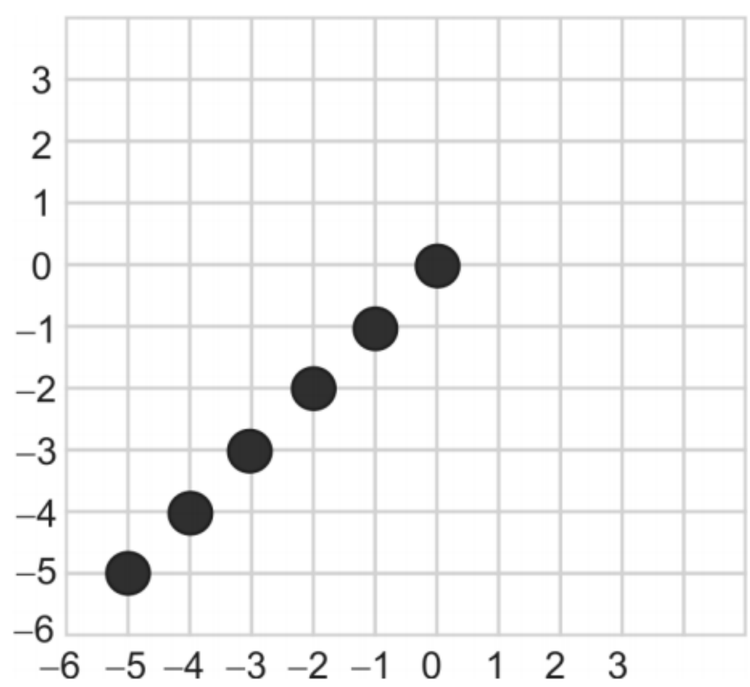


Fig. P. 3.2.5

Ex. 3.2.6 : Scan convert a line with end points (10, 5) and (16, 10) using DDA line drawing algorithm.

SPPU - May 18, 4 Marks

Soln. :

Let's take, $(x_1, y_1) = (10, 5)$ and $(x_2, y_2) = (16, 10)$

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{10-5}{16-10} = \frac{5}{6} = 0.833$$

Line is in first quadrant, it is growing from left to right and slope $|m| < 1$, so set $\Delta x = 1$ and $\Delta y = m = 0.833$. So,

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

$$y_{k+1} = y_k + \Delta y = y_k + 0.833$$

Table P. 3.2.6 shows the calculation for required pixels.

Table P. 3.2.6 : Calculation for required pixels

K	(x_k, y_k)	$x_{k+1} = x_k + 1$	$y_{k+1} = y_k + 0.833$	Round (y_{k+1})
0	(10, 5)	10 + 1 = 11	5 + 0.833 = 5.833	6
1	(11, 6)	11 + 1 = 12	5.833 + 0.833 = 6.666	7
2	(12, 7)	12 + 1 = 13	6.666 + 0.833 = 7.499	7
3	(13, 7)	13 + 1 = 14	7.499 + 0.833 = 8.332	8
4	(14, 8)	14 + 1 = 15	8.325 + 0.833 = 9.165	9
5	(15, 9)	15 + 1 = 16	9.165 + 0.833 = 9.998	10
6	(16, 10)	–	–	–

In Table P. 3.2.6, columns 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.6.

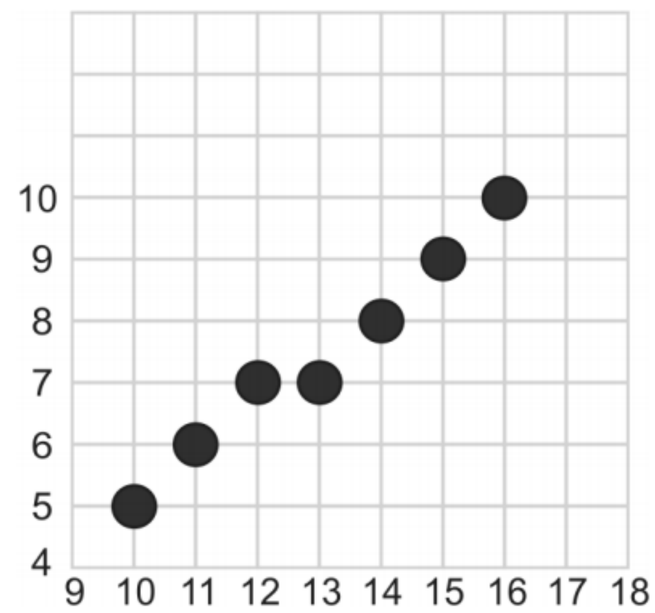


Fig. P. 3.2.6 : DDA line between points (10, 5) and (16, 10)

TechKnowledge Publications

Ex. 3.2.7 : Explain DDA line drawing algorithm. Consider line segment from A(– 2, – 1) to B(6, 3) use DDA line drawing algorithm to rasterize this line.

SPPU - May 19, 6 Marks

Soln :

Let's take, $(x_1, y_1) = (-2, -1)$ and $(x_2, y_2) = (6, 3)$

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

$$= \frac{3-(-1)}{6-(-2)} = \frac{4}{8} = 0.5$$

Line is in third and first quadrant, it is growing from left to right and slope $|m| < 1$, so set $\Delta x = 1$ and $\Delta y = m = 0.5$. So,

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

$$y_{k+1} = y_k + \Delta y = y_k + 0.5$$

Table P. 3.2.7 shows the calculation for required pixels.

Table P. 3.2.7 : Calculation for required pixels

k	(x_k, y_k)	$x_{k+1} = x_k + 1$	$y_{k+1} = y_k + 0.5$	Round (y_{k+1})
0	(– 2, – 1)	– 2 + 1 = – 1	– 1 + 0.5 = – 0.5	0
1	(– 1, 0)	– 1 + 1 = 0	– 0.5 + 0.5 = 0	0
2	(0, 0)	0 + 1 = 1	0 + 0.5 = 0.5	1
3	(1, 1)	1 + 1 = 2	0.5 + 0.5 = 1	1
4	(2, 1)	2 + 1 = 3	1 + 0.5 = 1.5	2
5	(3, 2)	3 + 1 = 4	1.5 + 0.5 = 2	2
6	(4, 2)	4 + 1 = 5	2 + 0.5 = 2.5	3
7	(5, 3)	5 + 1 = 6	2.5 + 0.5 = 3	3
8	(6, 3)	–	–	–

In Table P. 3.2.7, columns 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.7.

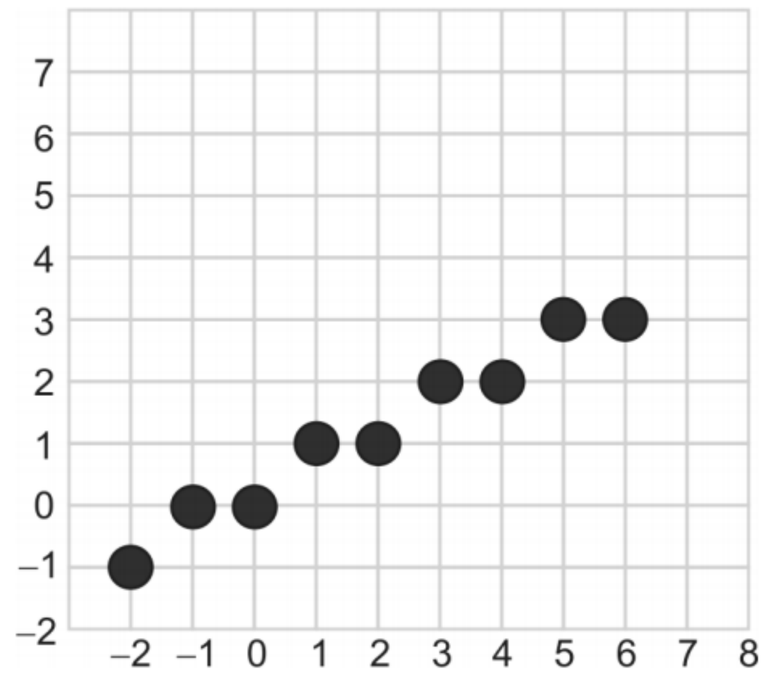


Fig. P. 3.2.7 : DDA line between points (–2, –1) and (6, 3)

3.2.2 Bresenham Line Drawing Algorithm

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

SPPU - May 15, May 16, 6 Marks

Q. Explain the Bresenham algorithm for line drawing. Write the advantages and disadvantages of it over the DDA line drawing algorithm.

SPPU - Dec. 17, 6 Marks

Q. Write Bresenham's line drawing algorithm to draw a dotted line. Also, explain any two advantages of Bresenham's line drawing algorithm over other line drawing algorithm. **SPPU - Dec. 18, 6 Marks**

Q. Write a Bresenham line drawing algorithm. Also, explain the mathematical foundation of it.

SPPU - May 19, 6 Marks

3.2.2.1 Working Mechanism

- DDA algorithm is simple but not efficient. It involves floating-point calculations.
- This algorithm was introduced by "Jack Elton Bresenham" in 1962 at IBM. It is a scan conversion algorithm. It is a powerful, useful, and accurate algorithm. An incremental integer calculations are used to draw a line. The integer calculations include addition, subtraction, and multiplication.
- Bresenham has proposed integer calculation based on an incremental approach to computing the next pixel position along a straight line.

Techknowledge Publication

- Given the endpoints of the line (x_0, y_0) and (x_1, y_1), we will derive the incremental formula for Bresenham's line drawing algorithm.
- This approach selects the pixel which is close to the ideal line. It uses only integer addition and subtraction operations.
- **Assumption :** Line is in the firstquadrant, growing from left to right and $|m| < 1$.

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

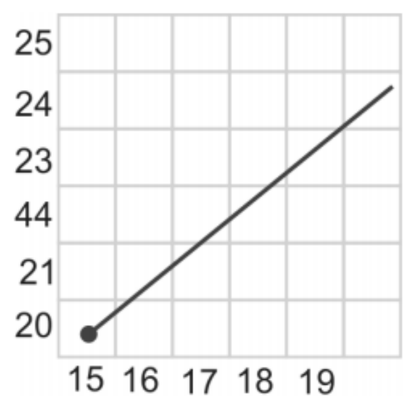


Fig. 3.2.5 : Segment of the line

- Fig. 3.2.5 shows the segment of the line. The line has a slope of less than 1. By sampling in the x-direction, we shall compute y-coordinate that best matches with the ideal line. In Fig. 3.2.5, (15, 20) is the known pixel value, next pixel can be drawn on (16, 20) or (16, 21).
- Thus, for the line with slope $|m| < 1$, if the current pixel is (x_k, y_k), then next pixel on line would be at $(x_k + 1, y_k)$ or $(x_k + 1, y_k + 1)$.
- For each $(x_k + 1)$ position, algorithm determines on which scan line pixel should be plotted, i.e. y_k or y_{k+1}?
- The computed y-coordinate for $(x_k + 1)$ position is let's say y. Distance between y and y_k is d_1 and distance between $(y_k + 1)$ and y is d_2 (Refer Fig. 3.2.6).
- From Fig. 3.2.6, it is clear that $d_2 < d_1$ implies line is close to scan line $(y_k + 1)$, so we shall plot pixel at $(x_k + 1, y_k + 1)$ location.
- And if $d_1 < d_2$, the line is close to scan line $y_k$and hence next pixel remains on the same scan line, and we shall plot pixel at $(x_k + 1, y_k)$.
- Thus, the choice of the next pixel depends on the sign of $(d_1 - d_2)$. If it is positive, plot $(x_k + 1, y_k)$ i.e. pixel in East (E) direction, else plot $(x_k + 1, y_k + 1)$ i.e. Pixel in North-East (NE) direction. This computation needs to be performed recursively till the next endpoint of the line.

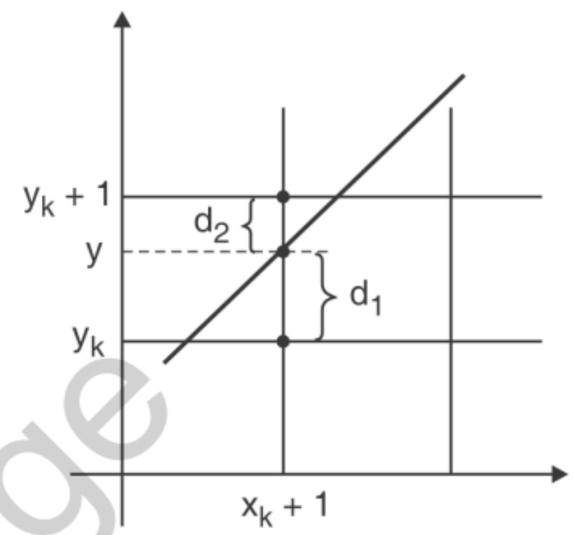


Fig. 3.2.6

- Implicit representation of line is $y = mx + c$. At sampling position $x_k + 1$, corresponding y is computed as,

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

From Fig. 3.2.6,
$$d_1 = y - y_k = (m \cdot x_k + m + c) - y_k$$
$$d_2 = y_{k+1} - y$$
$$= y_k + 1 - (m \cdot x_k + m + c)$$

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

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

- To simplify the computation, multiply it by Δx,

Decision parameter at step k,

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

Put, $m = \Delta y / \Delta x$ in above equation and simplify,

$$= 2\Delta y \cdot x_k + 2\Delta y - 2\Delta x \cdot y_k + 2\Delta x \cdot c - \Delta x \quad \text{...(3.1.1)}$$

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

Where, $B = 2\Delta y + \Delta x (2c - 1)$.

TechKnowledge Publications

Value of 'B' does not depend on pixel position, so in subsequent computation of p_k, 'B' would be eliminated.

Decision parameter at step k + 1 would be,

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

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

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

Increment is x direction is always 1,

$$\text{so } x_{k+1} = x_k + 1.$$

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

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

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

if $p_k \geq 0$, then select NE pixel and hence set $p_{k+1} = p_k + 2\Delta y - 2\Delta x$, otherwise select E pixel and set $p_{k+1} = p_k + 2\Delta y$

Initial Decision Parameter

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

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

Put this value in equation (3.1.1).

Initial decision parameter,

$$p_0 = 2\Delta y \cdot x_0 + 2\Delta y - 2\Delta x \cdot y_0 + 2\Delta x \cdot (y_0 - (\Delta y / \Delta x) \cdot x_0) - \Delta x$$

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

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

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

If decision parameter at any point is positive, then select NE pixel otherwise select E pixel as the next pixel on line.

3.2.2.2 Algorithm

Steps for Bresenham's line drawing algorithm to draw a line in the first quadrant are described here :

Steps of Bresenham's Line Drawing Algorithm

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

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

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

Step 3 : Compute the constants.

$$\Delta x = x_1 - x_0, \ \Delta y = y_1 - y_0,$$

initial decision parameter

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

Step 4 : if $p \geq 0$ then

$$x = x + 1$$

$$y = y + 1$$

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

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

else

$$x = x + 1$$

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

//Select pixel in East (E) direction

Step 5 : Repeat step 4 Δx times

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

Algorithm for Bresenham's Line Drawing Method

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

Algorithm BRESENHAM_LINE(x_0, y_0, x_1, y_1)

$\Delta x \leftarrow x_1 - x_0$

$\Delta y \leftarrow y_1 - y_0$

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

for $k \leftarrow 0$ to Δx

if $p_k \geq 0$ **then**

putPixel(x_{k+1}, y_{k+1})

// Select pixel in North-East direction

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

else

putPixel(x_{k+1}, y_k)

// Select pixel in East direction

$p_{k+1} \leftarrow p_k + 2\Delta y$

end

end

Generalization

Above discussed approach is valid for the line $0 < m < 1$ and $x_1 < x_2$. To generalize this approach, perform following steps.

– If $x_1 > x_2$, swap them

– For lines, $m > 1$, interchange x with y.

Program 3.2.2 : Program for Bresenham line drawing algorithm.

```
#include<graphics.h>
#include<math.h>
#include<iostream.h>
void main()
{
    int x1,x2,y1,y2,i,e,x,y,dx,dy;
    int gd = DETECT, gm;
    initgraph(&gd,&gm,"NULL");
    cout<<"Enter co-ordinates of point 1: ";
    cin>>x1>>y1;
    cout<<"Enter co-ordinates of point 2: ";
    cin>>x2>>y2;
    dx = abs(x2 – x1);
    dy = abs(y2 – y1);
    x=x1;
    y=y1;
    e = 2*dy – dx;
```

TechKnowledge Publications

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

3.2.2.3 Pros and Cons

Advantages and disadvantages of Bresenham's line drawing algorithm are stated here :

Advantages

- It involves only integer calculation.
- It is faster than DDA.
- In decision parameter, multiplication by 2 can be implemented in hardware using shift register.
- Involves cheaper operations like addition and subtraction.
- More accurate.
- Bresenham's algorithm does not perform rounding operation.

Disadvantages

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

3.2.2.4 Examples

Ex. 3.2.8 : Calculate the pixel coordinates of line PQ using Bresneham's algorithm. Where, P = (20, 20) and Q = (10, 12).

Soln. :

Let us assume,

$$(x_1, y_1) = (20, 20) \text{ and } (x_2, y_2) = (10, 12)$$

$$\Delta x = dx = |20 - 10| = 10,$$

$$\Delta y = dy = |20 - 12| = 8$$

Initial decision parameter,

$$p_0 = 2dy - dx = 16 - 10 = 6$$

$$\Delta E = 2dy = 16$$

(Increment if E pixel is selected)

$$\Delta NE = 2(dy - dx) = -4$$

(Increment if NE pixel is selected)

Table P. 3.2.8

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	–	–	10	12	$p_k = p_0 = 2dy - dx = 6$
1	(10, 12)	$p_k \geq 0$, so select NE	10 + 1 = 11	12 + 1 = 13	$p_k = p_k + \Delta NE = 6 - 4 = 2$
2	(11, 13)	$p_k \geq 0$, so select NE	11 + 1 = 12	13 + 1 = 14	$p_k = p_k + \Delta NE = 2 - 4 = -2$
3	(12, 14)	$p_k < 0$, so select E	12 + 1 = 13	14 + 0 = 14	$p_k = p_k + \Delta E = -2 + 16 = 14$

Tech Knowledge Publications

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

In Table P. 3.2.8, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.8.

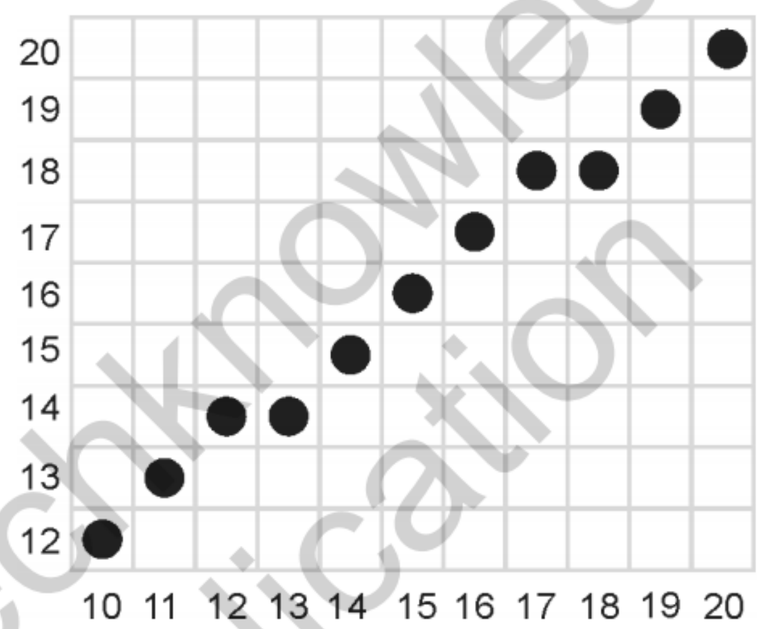


Fig. P. 3.2.8 : Line between (20, 20) to (10, 12) using Bresenham's line drawing approach

Ex. 3.2.9 : Consider line from (4,4) to (12,9). Use Bresenham's algorithm to rasterize this line.

Soln. :

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

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

Initial decision parameter,

$$p_0 = 2dy - dx = 10 - 8 = 2$$

$$\Delta E = 2dy = 10 \quad \text{(Increment if E pixel is selected)}$$

$$\Delta NE = 2(dy - dx) = 2(5 - 8) = -6 \quad \text{(Increment if NE pixel is selected)}$$

Table P. 3.2.9

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	p_k
0	–	–	4	4	$p_k = p_0 = 2dy - dx = 2$
1	(4, 4)	$p_k \geq 0$, so select NE	4 + 1 = 5	4 + 1 = 5	$p_k = p_k + \Delta NE = 2 - 6 = -4$

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

In Table P. 3.2.9, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.9.

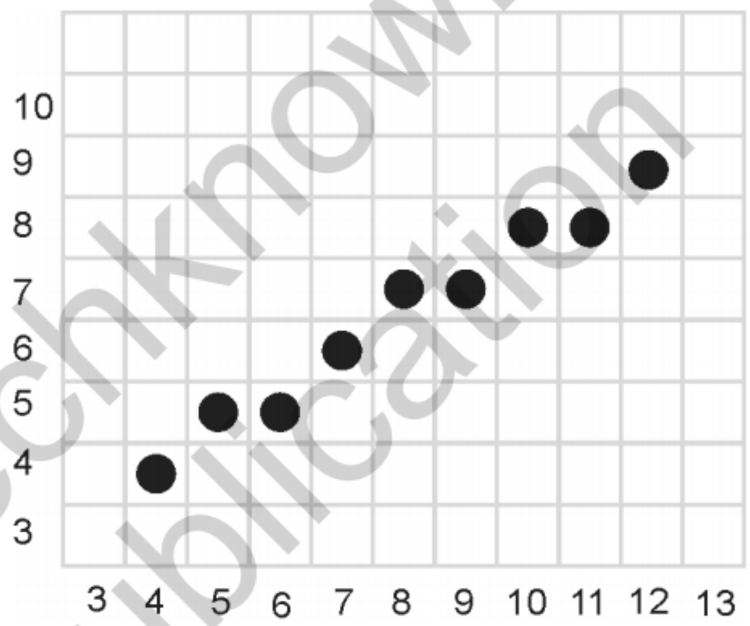


Fig. P. 3.2.9 : Line between (4, 4) to (12, 9) using Bresenham's line drawing approach

Ex. 3.2.10 : Indicate which raster locations would be chosen by Bresenham's algorithm when scan-converting a line from pixel coordinate (1, 1) to pixel coordinate (8, 5).

Soln. : Let us assume, $(x_1, y_1) = (1, 1)$ and $(x_2, y_2) = (8, 5)$

$$dx = |8 - 1| = 7, \quad dy = |5 - 1| = 4$$

$$m = dy / dx = 4 / 7 < 1,$$

$$\text{Initial decision parameter, } d = 2dy - dx = 8 - 7 = 1$$

If d = 0, we can select any pixel, E or NE. We will select NE here.

$$\Delta E = 2dy = 8$$

$$\Delta NE = 2(dy - dx) = -6$$

Table P. 3.2.10

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	–	–	1	1	d = 2dy – dx = 1
1	(1, 1)	d > 0, so select NE	1 + 1 = 2	1 + 1 = 2	d + ΔNE = 1 – 6 = – 5
2	(2, 2)	d < 0, so select E	2 + 1 = 3	2 + 0 = 2	d + ΔE = – 5 + 8 = 3
3	(3, 2)	d > 0, so select NE	3 + 1 = 4	2 + 1 = 3	d + ΔNE = 3 – 6 = – 3
4	(4, 3)	d < 0, so select E	4 + 1 = 5	3 + 0 = 3	d + ΔE = – 3 + 8 = 5
5	(5, 3)	d > 0, so select NE	5 + 1 = 6	3 + 1 = 4	d + ΔNE = 5 – 6 = 1
6	(6, 4)	d > 0, so select NE	6 + 1 = 7	5 + 1 = 5	d + ΔNE = 1 – 6 = – 5
7	(7, 5)	d < 0, so select E	7 + 1 = 8	5 + 0 = 5	d + ΔE = – 5 + 8 = 3
8	(8, 5)	–	–	–	–

In Table P. 3.2.10, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.10.

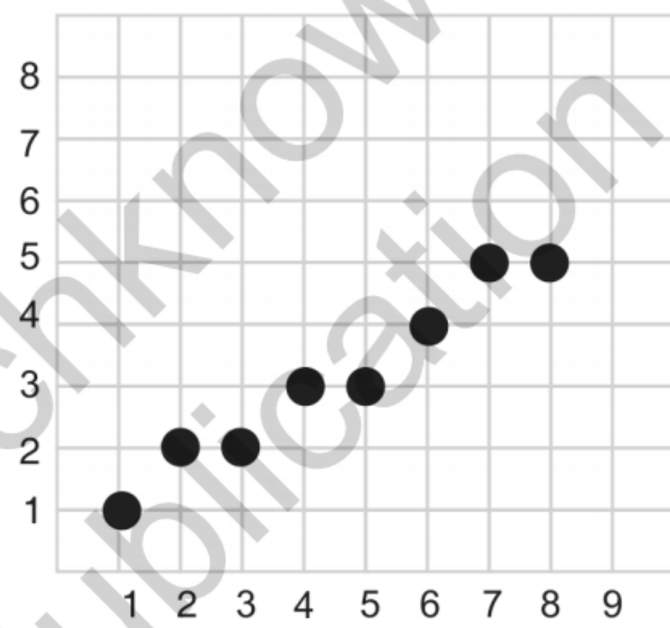


Fig. P. 3.2.10

Techknowledge Publication

Ex. 3.2.11 : What are the steps involved in Bresenham's line drawing algorithm for the line (0, 0) to (– 8, – 5)?

Soln. :

To draw a line from left to right, let us assume,

$$(x_1, y_1) = (-8, -5) \text{ and } (x_2, y_2) = (0, 0)$$

$$dx = 0 - (-8) = 8, \quad dy = 0 - (-5) = 5$$

Initial decision parameter,

$$d = 2dy - dx = 10 - 16 = -6$$

As d < 0, we select E pixel.

$$\Delta E = 2dy = 10$$

$$\Delta NE = 2\,(dy - dx) = -6$$

Table P. 3.2.11

K	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	–	–	– 8	– 5	d = 2dy – dx = – 6
1	(– 8, – 5)	d < 0, so select E	– 8 + 1 = – 7	– 5 + 0 = – 5	d + ΔE = – 6 + 10 = 4
2	(– 7, – 5)	d > 0, so select NE	– 7 + 1 = – 6	– 5 + 1 = – 4	d + ΔNE = 4 – 6 = – 2
3	(– 6, – 4)	d < 0, so select E	– 6 + 1 = – 5	– 4 + 0 = – 4	d + ΔE = – 2 + 10 = 8
4	(– 5, – 4)	d > 0, so select NE	– 5 + 1 = – 4	– 4 + 1 = – 3	d + ΔNE = 8 – 6 = 2
5	(– 4, – 3)	d > 0, so select NE	– 4 + 1 = – 3	– 3 + 1 = – 2	d + ΔNE = 2 – 6 = – 4
6	(– 3, – 2)	d < 0, so select E	– 3 + 1 = – 2	– 2 + 0 = – 2	d + ΔE = – 4 + 10 = 6
7	(– 2, – 2)	d > 0, so select NE	– 2 + 1 = – 1	– 2 + 1 = – 1	d + ΔNE = 6 – 6 = 0
8	(– 1, – 1)	d ≥ 0, so select NE	– 1 + 1 = 0	– 1 + 1 = 0	–
9	(0, 0)	–	–	–	–

In Table P. 3.2.11, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.11.

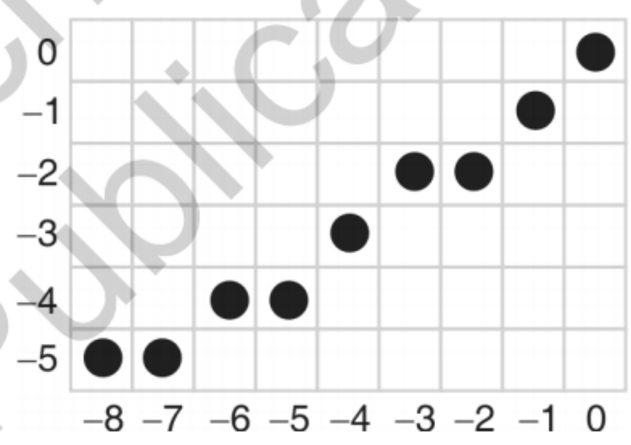


Fig. P. 3.2.11 : Line between (– 8, – 5) to (0, 0) using the Midpoint line-drawing approach.

Ex. 3.2.12 : Write and explain Bresenham's line algorithm and find out which pixel would be turned on for the line with end points (3, 2) to (7, 4) using the same. **SPPU - May 17, 7 Marks**

Soln. :

Let us assume,

$$(x_1, y_1) = (3, 2) \text{ and } (x_2, y_2) = (7, 4)$$

$$dx = |7 - 3| = 4, \quad dy = |4 - 2| = 2$$

$$m = dy / dx = 2 / 4 = 0.5, \text{ slope is less than 1}$$

Initial decision parameter,

$$d = 2dy - dx = 4 - 4 = 0$$

If d = 0, we can select any pixel, E or NE. We will select NE here.

$$\Delta E = 2dy = 4$$

$$\Delta NE = 2(dy - dx) = -4$$

Table P. 3.2.12

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	–	–	3	2	d = 2dy – dx = 0
1	(3, 2)	d = 0, so select NE	3 + 1 = 4	2 + 1 = 3	d + ΔNE = 0 – 4 = – 4
2	(4, 3)	d < 0, so select E	4 + 1 = 5	3 + 0 = 3	d + ΔE = – 4 + 4 = 0
3	(5, 3)	d = 0, so select NE	5 + 1 = 6	3 + 1 = 4	d + ΔNE = 0 – 4 = – 4
4	(6, 4)	d < 0, so select E	6 + 1 = 7	4 + 0 = 4	d + ΔE = – 4 + 4 = 0
8	(7, 4)	–	–	–	–

In Table P. 3.2.12, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.12.

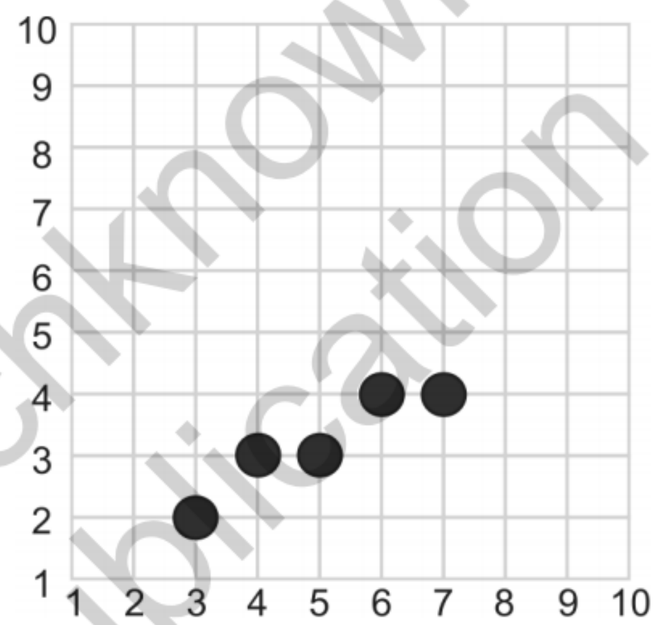


Fig. P. 3.2.12

Ex. 3.2.13 : Write a function Bresenham's line $(x_1, y_1, x_2, y_2$, type) so it draws a line with the specified type. If type = 0, solid line, type = 1 dashed line and type = 2 dotted line.

Soln. :

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

Algorithm BRESENHAM_LINE $(x_1, y_1, x_2, y_2$, type)

$\Delta x \leftarrow x_2 - x_1$

$\Delta y \leftarrow y_2 - y_1$

$d \leftarrow 2\Delta y - \Delta x$ // Initial decision parameter

```
ΔE ← 2Δy
ΔNE ← 2(Δy – Δx)
x ← x1
y ← y1
Count ← 0
flag ← 1
putPixel (x, y)
while (x < x2) do
    if d < 0 then
      d ← d + ΔE
    else
      d ← d + ΔNE
      y ← y + 1
    end
    x ← x + 1
    if type = = 0 then                    // Solid Line
      putPixel (x, y)
    end
    if type = = 1 then                    // Dashed Line
      if (Count % 5) ≠ 0 then             // Draw five pixels and skip one to draw a dotted line
          putPixel (x, y)
          Count ← Count + 1
      end
    end

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

Technknowledge Publications

Ex. 3.2.14 : Write Bresenham's line drawing algorithm to draw dotted line. Also explain any two advantages of Bresenham's line drawing algorithm over other line drawing algorithm. **SPPU - Dec. 18, 6 Marks**

Soln. :

Algorithm BRESENHAM_LINE(x_0, y_0, x_1, y_1)

$\Delta x \leftarrow x_1 - x_0$

$\Delta y \leftarrow y_1 - y_0$

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

for $k \leftarrow 0$ to Δx **do**

$x_{k+1} = x_k + 1$

if $p_k < 0$ **then**

$p_{k+1} \leftarrow p_k + 2\Delta y$

else

$y_{k+1} = y_k + 1$

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

end

if $k \% 2 = = 0$ **then**

putPixel (x_k, y_k)

end

end

Ex. 3.2.15 : Consider a line from (2, 5) to (8, 8). Use Bresenham's line drawing algorithm rasterize this line. **SPPU - Dec. 19, 6 Marks**

Soln. : Let us assume $(x_1, y_1) = (2, 5)$

And $(x_2, y_2) = (8, 8)$

$\Delta x = dx = |x_2 - x_1| = |8 - 2| = 6, \Delta y = dy$

$= |y_2 - y_1| = |8 - 5| = 3$

Initial decision parameter, $p_0 = 2dy - dx = 6 - 6 = 0$

$\Delta E = 2dy = 6$ (Increment if E pixel is selected)

$\Delta NE = 2(dy - dx)$

$= 2(3 - 6) = -6$ (Increment if NE pixel is selected)

Table P. 3.2.15

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

Techknowledge Publication

In Table P. 3.2.15, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P. 3.2.15.

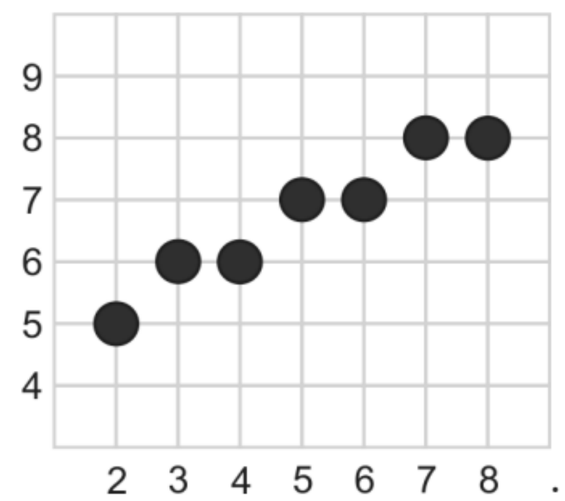


Fig. 3.2.15

Advantages

- Involves only integer calculation.
- It is faster than DDA.
- Multiplication by two can be implemented in hardware using shift register.
- Involves cheaper operations like addition and subtraction.
- More accurate.
- Bresenham's algorithm does not perform rounding operation.

3.2.3 DDA vs. Bresenham Line Drawing Algorithm

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

3.3 Circle Drawing Algorithms

- The circle is another very useful and important output primitive. Almost all graphics packages provide the functionality to plot and transform the circle.
- Implicit equation of a circle centered at the origin is given by $x^2 + y^2 = r^2$, where r is the radius and (x, y) is any pair of the coordinate on circle boundary which satisfies the above equation.
- If the circle is centered at the point (x_c, y_c), it should be translated to origin to plot the circle. The equation of the circle, in this case, would be $(x - x_c)^2 + (y - y_c)^2 = r^2$.

TechKnowledge Publications

3.3.1 DDA Circle Drawing Algorithm

3.3.1.1 Working Mechanism

- This is an incremental algorithm.
- The circle is defined by the equation, $x^2 + y^2 = r^2$
- The algorithm starts drawing the circle from the initial point and computes ε (epsilon), by which the x and y-coordinates should be increased or decreased.
- The increment ε is derived as follow. Find the n such that,

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

Where r is the radius of the circle.

$$\varepsilon = \frac{1}{2^n}$$

- Start drawing the circle from point (0, r).Using ε, we can compute the points along circle perifery as follow :

$$x_2 = x_1 + \varepsilon \cdot y_1$$

$$y_2 = y_1 - \varepsilon \cdot x_1$$

- Computation of successive points halts when $((y_1 - y_0) < \varepsilon$ OR $(x_1 - x_0) > \varepsilon)$

3.3.1.2 Algorithm

- Algorithm for DDA circle drawing method is depicted here,

Algorithm DDA_CIRCLE(r) // r : radius of circle

Find the value of n such that $2^{n-1} \leq r \leq 2^n$

Compute value of $\varepsilon : \varepsilon = \frac{1}{2^n}$

Set initial points of circle : $x_1 = x_0, y_1 = y_0$

while ($(y_1 - y_0) < \varepsilon$ OR $(x_1 - x_0) > \varepsilon$) **do**

$x_2 = x_1 + \varepsilon \cdot y_1$

$y_2 = y_1 - \varepsilon \cdot x_1$

Plot (floor(x_2), floor(y_2))

Set, $x_1 = x_2$ and $y_1 = y_2$

end

3.3.1.3 Pros and Cons

Advantages

- It is simple
- It is easy to understand

Disadvantages

- Involves floating-point operations
- Use of floor operation increases the complexity of the algorithm
- Points generated by this algorithm are not accurate

Tech Knowledge Publications

3.3.2 Bresenham Circle Drawing Algorithm

Q. Explain Bresenham's circle drawing algorithm in detail. **SPPU - Dec. 15, 8 Marks**

Q. Explain significance of error term in Bresenham's circle drawing algorithm. Explain its mathematical derivations. **SPPU - May 16, 8 Marks**

Q. Write Bresenham circle drawing algorithm. Also explain mathematical foundation of it. **SPPU - Dec. 17, May 18, 6 Marks**

3.3.2.1 Working Mechanism

- Bresenham has proposed integer calculation based circle drawing algorithm. This approach is simple and faster.
- The circle is an eight-way symmetrical shape, so we will derive the formula for only one octant, circle in remaining 7 octants is produced by its symmetry property.
- Let (x_k, y_k) be the current pixel, as shown in Fig. 3.3.1 there are two choices for the next pixel :
  1. East pixel E, i.e. (x_{k+1}, y_k)
  2. South East pixel SE, i.e. (x_{k+1}, y_{k-1})

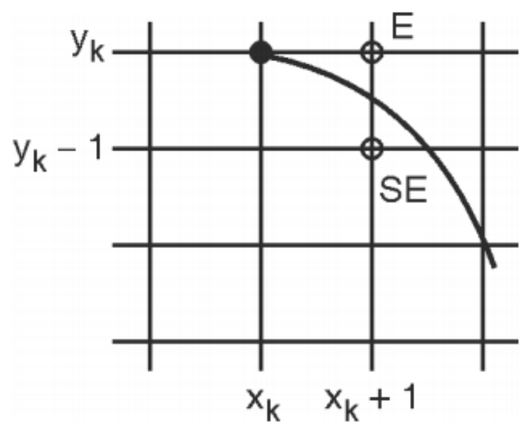


Fig. 3.3.1

- Explicit representation of the circle is,

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

  - If (x, y) is on circle, f (x, y) = 0
  - If (x, y) is an outside circle, f (x, y) > 0
  - If (x, y) is inside the circle, f (x, y) < 0
- For point E and SE, equations of the circle would be,

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

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

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

All five cases are depicted in Fig. 3.3.2.

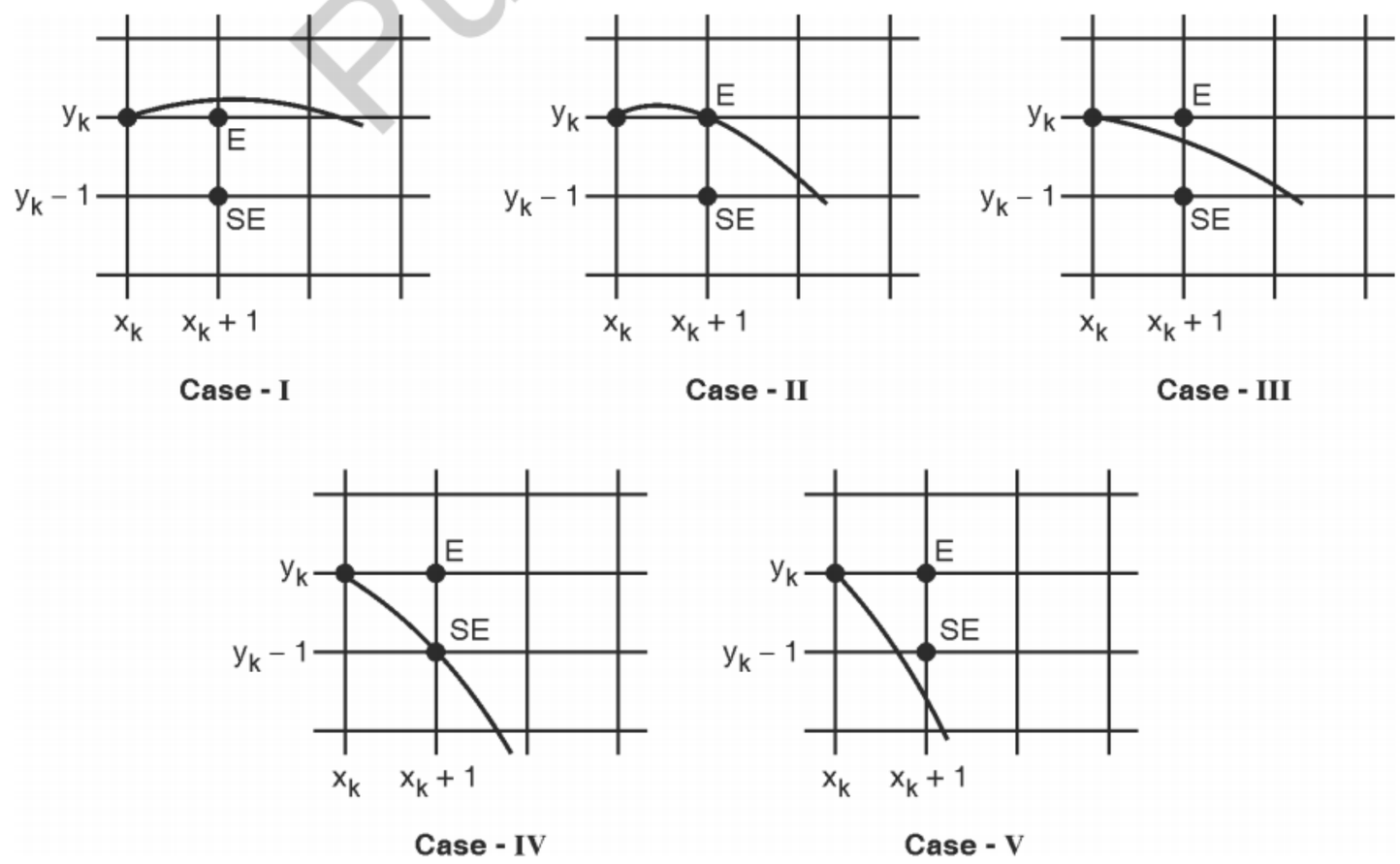


Fig. 3.3.2

Tech Knowledge Publications

According to the relation between arc E and SE, f(E) and f(SE) would be positive, zero or negative. Table 3.3.1 sign for all five cases.

Table 3.3.1

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

- Case I and Case V are trivial. Both points are inside or outside the circle in respective cases.
  - If both points are inside the circle (Case - I), the next pixel would be E (x_{k+1}, y_k), because arc is close to E.
  - If both points are outside the circle(Case - V), next pixel would be SE (x_{k+1}, y_{k-1}), because arc is close to SE.
- Cases II and IV are also straightforward. In case - II, f(E) = 0. So circle passes through the point (x_{k+1}, y_k). And in case IV, f(SE) = 0, so circle passes through point (x_{k+1}, y_{k-1}).
- From Table 3.3.1 and above discussion, it is simple to note that if the sum is negative, select East pixel, if the sum is positive, then select South East pixel.
- In case III, the sum can take any value. So according to sign of sum, we shall select E or SE as the next pixel. Let us derive the decision parameter for subsequent pixel choice.
- If we start plotting circle from initial point $(x_0, y_0) = (0, r)$ then

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

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

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

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

Initial decision parameter,

$$S = f(E) + f(SE) = 1 + 2 - 2r$$

$$S = 3 - 2r$$

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

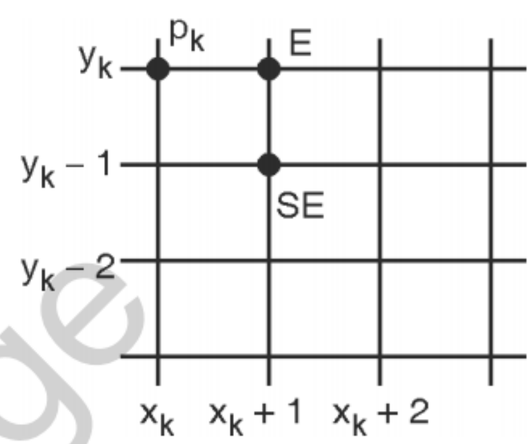


Fig. 3.3.3

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

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

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

1. If E pixel is selected as next pixel

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

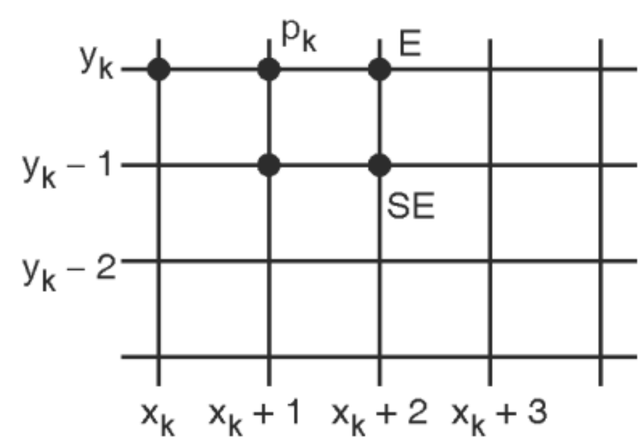


Fig. 3.3.4

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

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

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

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

TechKnowledge Publications

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

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

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

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

2. If SE is selected as next pixel

In Fig. 3.3.5, if $S_{old} > 0$, we shall select SE as next pixel. In next iteration, consider SE as current pixel, we have two choices from current pixel $(x_k + 1, y_k - 1)$ i.e. $(x_k + 2, y_k - 1)$ or $(x_k + 2, y_k - 2)$.

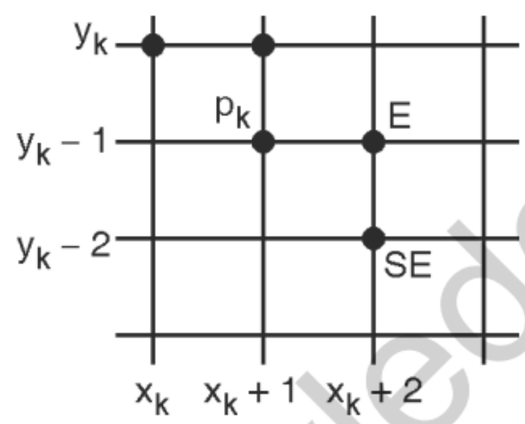


Fig. 3.3.5

From Fig. 3.3.5

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

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

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

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

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

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

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

3.3.2.2 Algorithm

Steps for Bresenham's circle drawing algorithm are stated below :

Steps of Bresenham's Circle Generation Algorithm

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

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

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

Tech Knowledge Publications

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

If $S < 0$ then

Set $S = S + 4x + 6$

else

$x = x + 1$

$y = y - 1$

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

Step 5 : Compute remaining seven symmetry points

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

Bresenham's Circle Drawing Algorithm

Bresenham's line drawing algorithm is described below.

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

  EightWaySymmetry(x, y)
  PutPixel(x, y)
  PutPixel(x, – y)
  PutPixel(– x, y)
  PutPixel(– x, – y)
  PutPixel(y, x)
  PutPixel(y, – x)
  PutPixel(– y, x)
  PutPixel(–y, –x)
```

Program 3.3.1 : Write C++/Java program to draw circle using Bresenham's algorithm. Inherit pixel class.

```
#include<graphics.h>
#include<iostream>
#include<math.h>
using namespace std;

class pixel
{
    public:
    float dp;
    int x1,y1,rad;
    void circle(int x, int y, int rad)
    {
      x1=0;
      y1=rad;
      dp=3-2*rad;
      while(x1<=y1)
      {
          if(dp<=0)
              dp+=(4*x1)+6;
          else
          {
              dp+=4*(x1-y1)+10;
              y1--;
          }
          x1++;
          display(x1,y1,x,y);
```

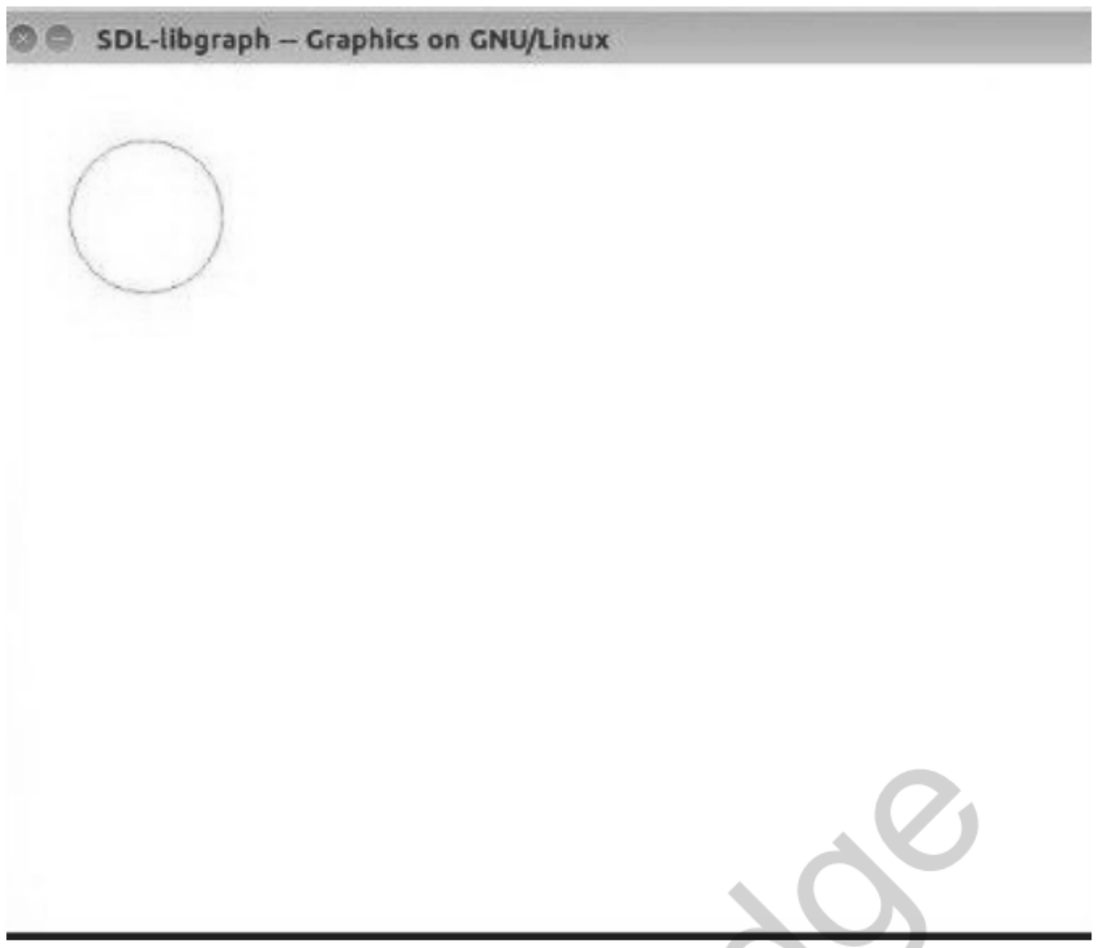
Techknowledge Publications

```
    }
  }
  void display(int x1,int y1,int x,int y)
  {
    putpixel(x1+x,y1+y,5);
    putpixel(x1+x,y-y1,5);
    putpixel(x-x1,y1+y,5);
    putpixel(x-x1,y-y1,5);
    putpixel(x+y1,y+x1,5);
    putpixel(x+y1,y-x1,5);
    putpixel(x-y1,y+x1,5);
    putpixel(x-y1,y-x1,5);
  }
};

class derive:public pixel
{
  public:
  int get()
  {
    cout<<"Enter x coordinate";
    cin>>x1;
    cout<<"Enter y coordinate";
    cin>>y1;
    cout<<"Enter radius";
    cin>>rad;
  }
};

int main()
{
  int gd=DETECT,gm;
  derive d;
  d.get();
  initgraph(&gd,&gm,0);
  d.circle(d.x1,d.y1,d.rad);
  getch();
  closegraph();
  return 0;
}
```

Output :

```
sndcompmml10@sndcompmml10-OptiPlex-330: ~
sndcompmml10@sndcompmml10-OptiPlex-330:~$ g++ brc.cpp -o b -lgraph
sndcompmml10@sndcompmml10-OptiPlex-330:~$ ./b
Enter x coordinate80
Enter y coordinate80
Enter radius40
[xcb] Unknown sequence number while processing queue
[xcb] Most likely this is a multi-threaded client and XInitThreads has not been
called
[xcb] Aborting, sorry about that.
b: ../../src/xcb_io.c:274: poll_for_event: Assertion `!xcb_xlib_threads_sequenc
_lost' failed.
```

Techknowledge Publications



3.3.2.3 Pros and Cons

Advantages

- Derivation of decision parameters is based on simple circle drawing formula, i.e. $x^2 + y^2 = r^2$
- It is easy to implement

Disadvantages

- It is time-consuming
- We won't get smooth circle due to uneven distance between pixels

3.3.2.4 Examples

Ex. 3.3.1 : Plot a circle using Bresenham's algorithm for r = 3 and center at (0, 0).

Soln. :

$$\text{Here, } r = 3, xc = 0, yc = 0$$

$$\text{First point to plot is } (0, r) = (0, 3)$$

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

Let's compute the remaining points on circle.

$$\text{Initial decision parameter, } r = 3 - 2r = 3 - 6 = -3$$

Table P. 3.3.1

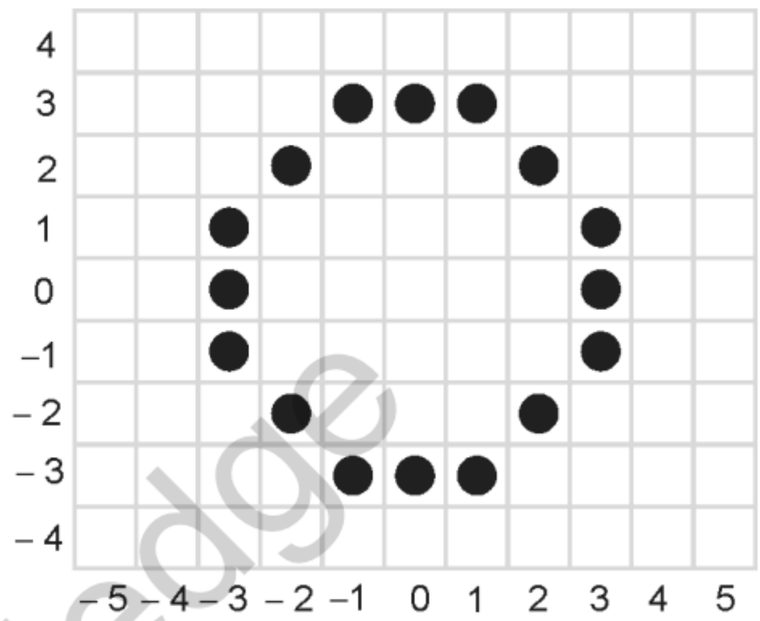
k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	–	–	0	3	$S = 3 - 2r = 3 - 6 = -3$
1	(0, 3)	$S < 0$, so select E	$0 + 1 = 1$	$3 + 0 = 3$	$S = S + 4x_k + 6 = -3 + 0 + 6 = 3$

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

Fig. P. 3.3.1 shows the final points on the circle.

X	Y
0	3
1	3
2	2

(a) Points computed in one octant



(b) Points generated using 9-way symmetry

Fig. P. 3.3.1 : Circle generation for given data

3.3.3 Midpoint Circle Drawing Algorithms

3.3.3.1 Working Mechanism

- Principle of midpoint circle drawing approach is identical to the midpoint line-drawing approach.
- Moving down from current pixel (x_k, y_k), possible point to be plotted is either East pixel E or South-East pixel (SE) as shown in Fig. 3.3.6. Coordinates of midpoint M are $\left(x_k + 1, y_k - \frac{1}{2}\right)$.

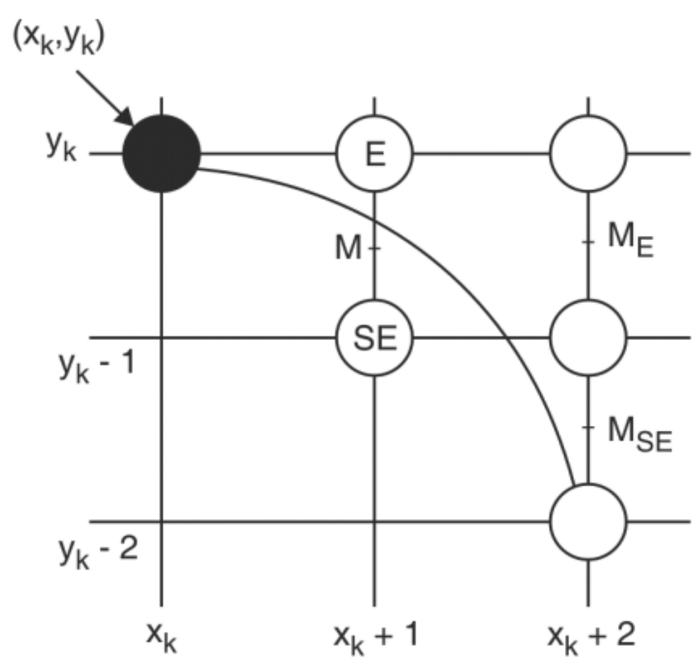


Fig. 3.3.6 : Midpoint circle drawing approach

- We can write circle equation in explicit form $f(x, y) = x^2 + y^2 - r^2$. We will put the value of midpoint coordinates in this function and evaluate its sign. If the function returns 0, means midpoint is on the circle and we can select E or SE as the next pixel. If the function returns positive value, implies midpoint is inside circle and the arc is closer to E pixel so select E as next pixel. And if function returns negative value, implies midpoint is outside circle and the arc is closer to SE pixel so select SE as next pixel.
- Like line drawing algorithm, we will compute decision parameter d to decide the next pixel.

$$d = d_{old} = f(M) = f\left(x_k + 1, y_k - \frac{1}{2}\right)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2,$$

if $d = f(M) = 0$, then select E or SE as a next pixel

$d = f(M) > 0$, then select SE as a next pixel

$d = f(M) < 0$, then select E as a next pixel

TechKnowledge Publications

– As shown in Fig. 3.3.6, if E is selected then, next mid point would be $M_E\left(x_k + 2, y_k - \frac{1}{2}\right)$ and if SE is selected, next mid point would be $M_{SE}\left(x_k + 2, y_k - \frac{3}{2}\right)$.

If E is chosen :

$$d_{new} = f(M) = f(M_E) = f\left(x_k + 2, y_k - \frac{1}{2}\right)$$

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

$$= \left(x_k^2 + 4x_k + 4\right) + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$= \left(x_k^2 + 2x_k + 1\right) + \left(y_k - \frac{1}{2}\right)^2 - r^2 + 2x_k + 3$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 + 2x_k + 3$$

$$\text{But } d_{old} = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$\therefore d_{new} = d_{old} + 2x_k + 3$$

We define this increment as ΔE

$$\therefore \Delta E = d_{new} - d_{old} = 2x_k + 3$$

This implies, if E is selected, we can compute the value of the next decision parameter just by adding $2x_k + 3$ to the old decision parameter.

If SE is chosen :

$$d_{new} = f(M) = f(M_{SE}) = f\left(x_k + 2, y_k - \frac{3}{2}\right)$$

$$= (x_k + 2)^2 + \left(y_k - \frac{3}{2}\right)^2 - r^2$$

$$= \left(x_k^2 + 4x_k + 4\right) + \left(y_k^2 - 3y_k + \frac{9}{4}\right) - r^2$$

$$= \left(x_k^2 + 2x_k + 1\right) + \left(y_k^2 - y_k + \frac{1}{4}\right) - r^2 + (2x_k + 3) + (-2y_k + 2)$$

$$= (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2 + 2x_k - 2y_k + 5$$

$$\text{But } d_{old} = (x_k + 1)^2 + \left(y_k - \frac{1}{2}\right)^2 - r^2$$

$$d_{new} = d_{old} + 2x_k - 2y_k + 5$$

We define this increment as ΔSE

$$\therefore \Delta SE = d_{new} - d_{old} = 2x_k - 2y_k + 5$$

This implies, if SE is selected, we can compute the value of the next decision parameter just by adding $2x_k - 2y_k + 5$ to the old decision parameter.

The initial point of the circle is always known, it is (0, r). The first midpoint would be at $\left(1, r - \frac{1}{2}\right)$ and an initial decision parameter is,

$$d_{initial} = f\left(1, r - \frac{1}{2}\right)$$

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

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

$$\therefore d_{initial} = \frac{5}{4} - r$$

Here, $d_{initial}$ introduces the floating-point arithmetic. We are interested in sign, not in magnitude. So we will accept slight modification $d_{initial} = 1 - r$.

3.3.3.2 Algorithm

```
Algorithm MIDPOINT_CIRCLE(r)
x ← 0
y ← r
d ← 1 – r
EightWaySummetry (x, y)
while x < y do
    if d < 0 then
      d ← d + 2x + 3
    else
      d ← d + 2x – 2y + 5
      y ← y – 1
    end
    x ← x + 1
    EightWaySymmetry (x, y)
end
```

Techknowledge Publications

Routine for eight way symmetry is shown below :

```
EightWaySymmetry (x, y)
putPixel (x, y)
putPixel (x, – y)
putPixel ( – x, y)
putPixel ( – x, – y)
putPixel (y, x)
putPixel (y, – x)
putPixel ( – y, x)
putPixel ( – y, – x)
```

3.3.3.3 Pros and Cons

Advantages

- It is powerful and efficient
- No special programming skill is needed
- This algorithm is used to generate complex graphics on the raster system
- It involves integer calculations only

Disadvantages

- It is time-consuming
- Circle generated using this algorithm is not so smooth due to uneven distance between pixels

3.3.3.4 Example

Ex. 3.3.2 : With the help of the midpoint circle drawing algorithm, find out the coordinates of points that lie on the circle with radius 10 and (15, 5) as center.

Soln. :

First, we will derive the pixels for the circle centred at the origin, and then we will translate them by (15, 5).

Circle equation $x^2 + y^2 - r^2 = 0$

So, $x^2 + y^2 - 100 = 0$

We start with $x = 0$ and $y = r = 10$

Initial decision parameter, $d = 1 - r$

$= 1 - 10 = -9$

If $d < 0$, select E

and $\Delta E = 2x_k + 3$

If $d > 0$,

select SE and $\Delta SE = 2x_k - 2y_k + 5$

Table P. 3.3.2

K	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	-	-	0	10	$d = 1 - r = 1 - 10 = -9$
1	(0, 10)	$d < 0$, so select E	$0 + 1 = 1$	$10 + 0 = 10$	$d = d + \Delta E = -9 + 3 = -6$
2	(1, 10)	$d < 0$, so select E	$1 + 1 = 2$	$10 + 0 = 10$	$d = d + \Delta E = -1 + 7 = 6$
3	(2, 10)	$d < 0$, so select E	$2 + 1 = 3$	$10 + 0 = 10$	$d = d + \Delta E = -1 + 7 = 6$
4	(3, 10)	$d > 0$, so select SE	$3 + 1 = 4$	$10 - 1 = 9$	$d = d + \Delta SE = 6 - 9 = -3$
5	(4, 9)	$d < 0$, so select E	$4 + 1 = 5$	$9 + 0 = 9$	$d = d + \Delta E = -3 + 11 = 8$
6	(5, 9)	$d > 0$, so select SE	$5 + 1 = 6$	$9 - 1 = 8$	$d = d + \Delta SE = 8 - 3 = 5$
7	(6, 8)	$d > 0$, so select SE	$6 + 1 = 7$	$8 - 1 = 7$	$8 >= y$, so stop
8	(7, 7)	-	-	-	-

Techknowledge Publication

After translation by vector [15 5], we get the points on circle boundary in the first octant as shown below :

Points for a circle centered at origin	Points for circle centered at (15, 5)
(0, 10)	(15, 15)
(1, 10)	(16, 15)
(2, 10)	(17, 15)
(3, 10)	(18, 15)
(4, 9)	(19, 14)
(5, 9)	(20, 14)
(6, 8)	(21, 13)
(7, 7)	(22, 12)

By using 8-way symmetry, we can generate pixels in the remaining seven octants.

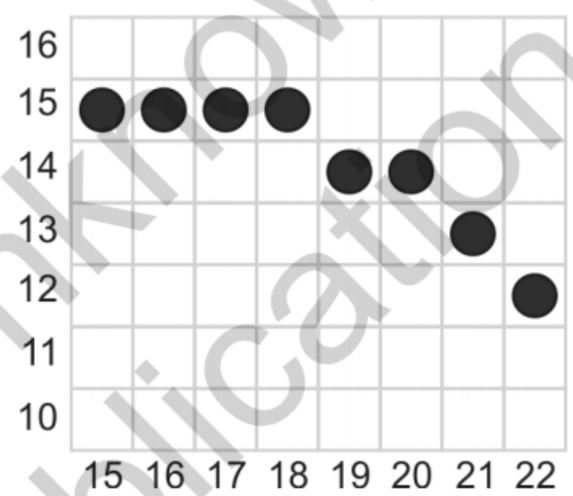


Fig. P. 3.3.2 : Circle in the second octant from (15, 15) to (22, 12) using Midpoint circle drawing approach

Tech Knowledge Publication

Ex. 3.3.3 : Apply algorithm and find out points for a circle with radius 8 and centre (0, 0) for one octant only.

Soln. :

$$\text{We start with } x = 0 \text{ and } y = r = 8$$

$$\text{Initial decision parameter } d = 1 - r = 1 - 8 = -7$$

$$\text{If } d < 0, \text{ select E and } \Delta E = 2x_k + 3$$

$$\text{If } d > 0, \text{ select SE and } \Delta SE = 2x_k - 2y_k + 5$$

Table P.3.3.3

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	-	-	0	8	$d = 1 - r = 1 - 8 = -7$
1	(0, 8)	$d < 0$, so select E	$0 + 1 = 1$	$8 + 0 = 8$	$d = d + \Delta E = -7 + 3 = -4$

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
2	(1, 8)	d < 0, so select E	1 + 1 = 2	8 + 0 = 8	d = d + ΔE = – 4 + 5 = 1
3	(2, 8)	d > 0, so select SE	2 + 1 = 3	8 – 1 = 7	d = d + ΔSE = 1 – 7 = – 6
4	(3, 7)	d < 0, so select E	3 + 1 = 4	7 + 0 = 7	d = d + ΔE = – 6 + 9 = 3
5	(4, 7)	d > 0, so select SE	4 + 1 = 5	7 – 1 = 6	d = d + ΔSE = 3 – 1 = 2
6	(5, 6)	d > 0, so select SE	5 + 1 = 6	6 – 1 = 5	-
7	(6, 6)	-	-	-	-

In Table P.3.3.3, column 2, i.e. (x_k, y_k) are the points to be displayed. They are shown in Fig. P.3.3.3.

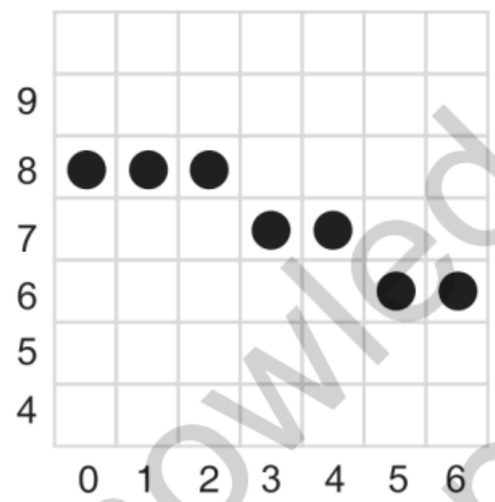


Fig. P.3.3.3 : Circle in the second octant from (0, 8) to (6, 6) using Midpoint circle drawing approach

Ex. 3.3.4 : Specify the midpoint circle algorithm, using the same, plot the circle whose radius is 8 units and center is at (10,10)

Soln. : First, we will derive the pixels for the circle centered at the origin, and then we will translate them by (10, 10).

We start with x = 0 and y = r = 8

Initial decision parameter d = 1 – r = 1 – 8 = – 7

If d < 0, select E and $\Delta E = 2x_k + 3$

If d > 0, select SE and $\Delta SE = 2x_k - 2y_k + 5$

Table P. 3.3.4(a)

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
0	-	-	0	8	d = 1 – r = 1 – 8 = – 7
1	(0, 8)	d < 0, so select E	0 + 1 = 1	8 + 0 = 8	d = d + ΔE = – 7 + 3 = – 4
2	(1, 8)	d < 0, so select E	1 + 1 = 2	8 + 0 = 8	d = d + ΔE = – 4 + 5 = 1
3	(2, 8)	d > 0, so select SE	2 + 1 = 3	8 – 1 = 7	d = d + ΔSE = 1 – 7 = – 6
4	(3, 7)	d < 0, so select E	3 + 1 = 4	7 + 0 = 7	d = d + ΔE = – 6 + 9 = 3
5	(4, 7)	d > 0, so select SE	4 + 1 = 5	7 – 1 = 6	d = d + ΔSE = 3 – 1 = 2

k	(x_k, y_k)	Decision	$x_k + 1$	$y_k + 1$	d_{new}
6	(5, 6)	d > 0, so select SE	5 + 1 = 6	6 – 1 = 5	-
7	(6, 6)	-	-	-	-

After translation by vector [10 10], we get the points on circle boundary in the first octant as shown in Table P. 3.3.4(b)

Table P. 3.3.4(b)

Points for a circle centered at the origin	Points for a circle centered at (10, 10)
(0, 8)	(10, 18)
(1, 8)	(11, 18)
(2, 8)	(12, 18)
(3, 7)	(13, 17)
(4, 7)	(14, 17)
(5, 6)	(15, 16)
(6, 6)	(16, 16)

By using 8-way symmetry, we can generate pixels in the remaining seven octants.

3.4 Aliasing

The process by which smooth curves and lines gives a jagged line appearance is called aliasing.

Staircase

One of the example of aliasing effects is staircase or jagged appearance. This effect becomes visible during scan converting a line or circle. This is because the line represented in the raster node is approximated by a sequence of pixels staircase or jagged appearance occur due to many reasons, the most common is the display device does not have enough resolution to display a smooth object. It also occurs when a bit mapped image is converted to a different resolution.

Brightness

Inspite of the same intensity level the slanted line appears dime than the horizontal or vertical lines. The reason is as the pixels on the horizontal / vertical lines are placed one unit apart whereas those on diagonal line are approximately 1.4114 units apart. This difference in the density produces the observed difference in brightness.

Global aliasing

The overall length of the picket tence is approximately correct by the spacing between pickets will be distorted.

3.4.1 Antialiasing

Antialiasing is a term used for techniques that removes the effects of aliasing. The easiest solution for removal of aliasing is to increase the resolution of the image. But sometimes this would not be very effective solution. These are techniques that can reduce this effect and improve the quality of images.

Tech Knowledge Publications

Antialiasing Methods

1. Prefiltering

The prefiltering methods treat a pixel as an area. It computes the color of the pixel based on the backup of the scenes object and pixel area. It involves longer calculations and approximation.

2. Super Sampling

The aliasing effect can be reduced by taking more than one sample per pixel. Binary pixel is divided in nine equal parts which are the sampling points. Few of the nine sub pixels may be colored in background and remaining may be colored as the color of object.

3. Post filtering

Super sampling and post filtering are very much similar. Here a more weightage is given to the center point and remaining is distributed among 8 samples.

4. Pixel phasing

It is a hardware based anti-aliasing method. The graphics system is capable of shifting the individual pixel from its original position in the pixel grid by a fraction of unit distance between pixels.

Gray Level

While displaying an image on the screen, many display system has only two states i.e On (1)and OFF(0). The object or as in our case when line is displayed a jagged like appearance or a staircase like effect can be seen. So to reduce this gray level antialiasing technique can be used. In this method the pixels are set to gray in between black and white.

Review Questions

Q. 1 Explain the way of computing pixel positions using the slope-intercept formula.

Q. 2 Explain the explicit formula of line drawing.

Q. 3 Define the line segment.

Q. 4 Discuss the characteristics of the ideal line.

Q. 5 Explain and write steps for DDA line drawing algorithm.

Q. 6 Enlist the properties of an ideal line.

Q. 7 What are the advantages of DDA line drawing algorithm?

Q. 8 What are the disadvantages of DDA line drawing algorithm?

Q. 9 Explain with the help of illustration, Bresenham's line drawing algorithm.

Q. 10 Derive the decision parameter expressions for Bresenham's line drawing algorithm.

Q. 11 Write the Bresenham's line drawing algorithm and explain how it is better than the DDA algorithm for line generation.

Q. 12 Mention the steps for Bresenham's line drawing algorithm.

Q. 13 Enlist advantages and disadvantages of Bresenham's line drawing algorithm.

Q. 14 State merits and demerits of Bresenham's line drawing algorithm.

Q. 15 Write an algorithm for Bresenham's Line drawing method.

Q. 16 Compare DDA and Bresenham's line drawing algorithm.

Q. 17 Derive the expression for decision parameter used in Bresenham's circle drawing algorithm.

Q. 18 Mention the steps for Bresenham's circle generation algorithm.

Q. 19 Write an algorithm for Bresenham's circle drawing algorithm.

❑❑❑

Technical Knowledge Publication

CHAPTER 4

UNIT II

Polygon Processing

Syllabus

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

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

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

4.1 Polygons

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

4.1.1 Types of Polygon

- **Polygon** is a closed figure made up of connected lines. Polygon has many sides and is represented by line segments. Line segments forming the boundary of the polygon are called **edges** of the polygon and the endpoints of edges are known as **vertices** of the polygon.
- Polygon is always closed figure; hence less than three sides cannot form the polygon. Triangle is the polygon with a minimum number of edges. The circle is closed figure but it does not have edges, so it is not a polygon. Few examples of polygons are shown in Fig. 4.1.1.

Fig. 4.1.1 : Polygon with different shapes and sides

- We can classify polygon in one of the following categories :
  - Convex polygon
  - Concave polygon
  - Complex polygon

4.1.1.1 Convex Polygon

Q. Define the term Convex polygon.

SPPU - Dec. 18, 1 Mark

Technical Knowledge Publication

- In a convex polygon, the angle between any consecutive pair of edges is always less than 180°.
- For a convex polygon, if we select any random two points inside the polygon and if we join them through a line; all the points on the line are certainly inside the polygon.
- Examples of the convex polygon are shown in Fig. 4.1.2.

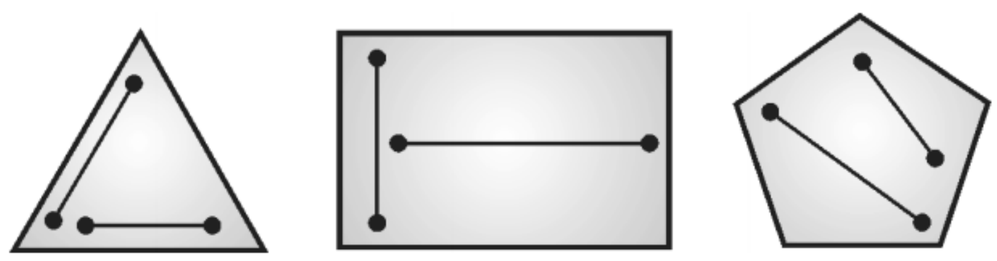


Fig. 4.1.2 : Example of convex polygons

4.1.1.2 Concave Polygon

Q. Define the term Concave polygon.

SPPU - Dec.18, 1 Mark

- In a concave polygon, the angle between at least one pair of consecutive edges is greater than 180°.
- For a concave polygon, there always exists a pair of vertices inside the polygon, when joined through a line segment, part of line segment falls outside the polygon.
- Examples of the concave polygon are shown in Fig. 4.1.3.

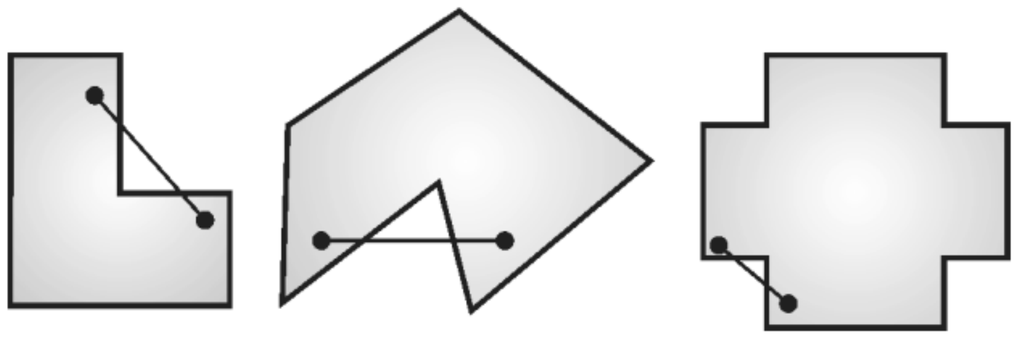


Fig. 4.1.3 : Examples of concave polygons

4.1.1.3 Complex Polygon

- Polygons which are neither convex nor concave are called a complex polygon. Complex polygons are self-intersecting or overlapping.
- Examples of complex polygon are shown in Fig. 4.1.4.

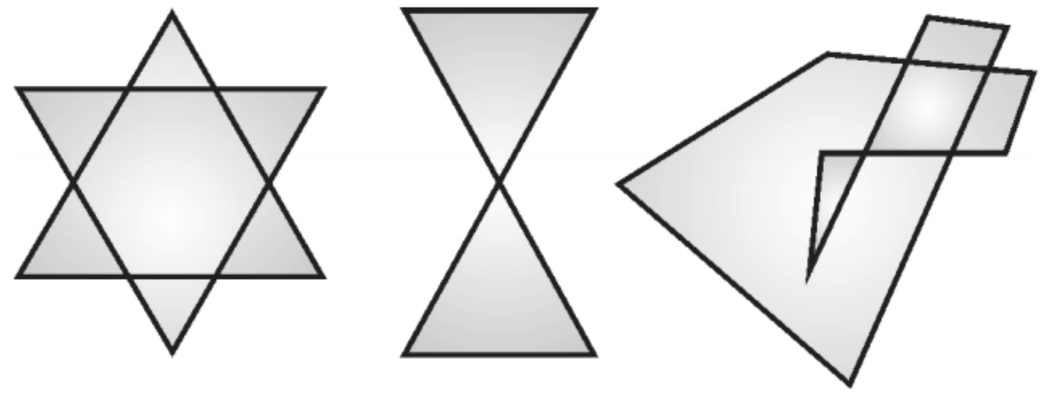


Fig. 4.1.4 : Examples of complex polygons

4.2 Inside Test

Q. What is inside test ? **SPPU - May 16, 1 Mark**

Q. Write and explain any one inside test algorithm.

SPPU - Dec. 18, 4 Marks

Q. Explain any one inside test algorithm.

SPPU - May 19, 2 Marks

During polygon filling, only those pixels should be filled which are inside the polygon. Deciding membership of pixel is very important in polygon filling. Accurate detection of whether the point is inside or outside the polygon is crucial. We will discuss two methods to decide if a pixel is inside the polygon.

- Even-odd method
- Winding number method

4.2.1 Even-Odd Test

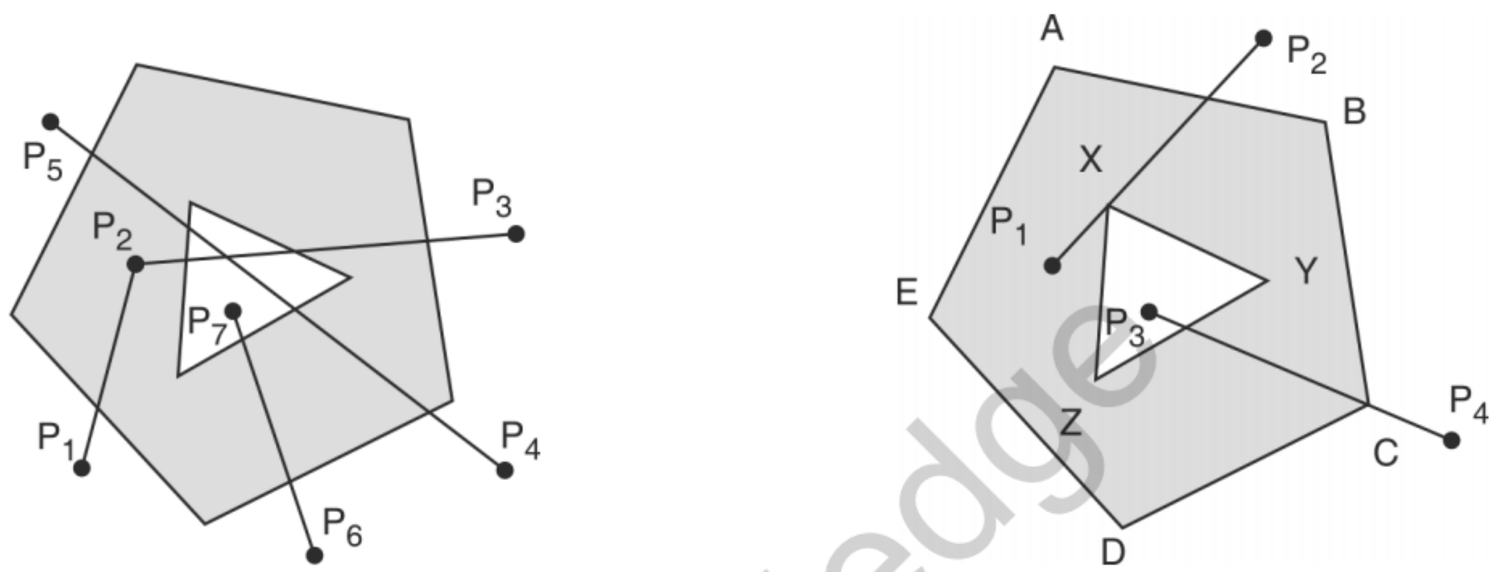
Q. Explain even-odd inside test with example.

SPPU - Dec. 15, May 16, 4 Marks

- This method constructs a line from a given point to the known point outside the polygon and checks the intersection of a line with polygon boundary.
- Let's check the visibility of point P_2 (Refer Fig. 4.2.1(a)). Draw a line from P_2 to known exterior point to a polygon in any direction and count number of intersections of line with polygon edges.

TechKnowledge Publications

- If count is odd, then the point is inside polygon otherwise it is outside of the polygon.
- Line P_2P_1 intersects polygon edges only once, while line P_2P_3 intersects polygon edge thrice, in both the cases count is odd, means the point is inside the polygon.
- Consider the point P_4, draw a line to known exterior point P_5, it intersects polygon boundary four times, so P_4 is considered as an exterior point. Same is true for point P_7.



(a) Simple case (b) Complex case

Fig. 4.2.1 : Odd-even test

- This rule is very simple, but the problem arises when the line crosses the common vertex shared by two adjacent edges.
- Consider Fig. 4.2.1(b), line P_3P_4 intersects the outer boundary of the polygon at common vertex shared by edges BC and CD. So even though P_3P_4 intersects three polygon edges (YZ, BC and CD), P_3 is an exterior point.
- The solution to this problem is to count intersection as a single intersection if both edges are on the opposite side of the line. For example, CD and BC in case of line P_3 P_4.
- If both edges are on the same side of the line segment, then count it as two intersections.
- For example, XY and XZ are on the same side of line P_1P_2, so count it as two intersection points.
- It is easy to understand from Fig. 4.2.2, where to count once (odd count) and where to count twice (even count).

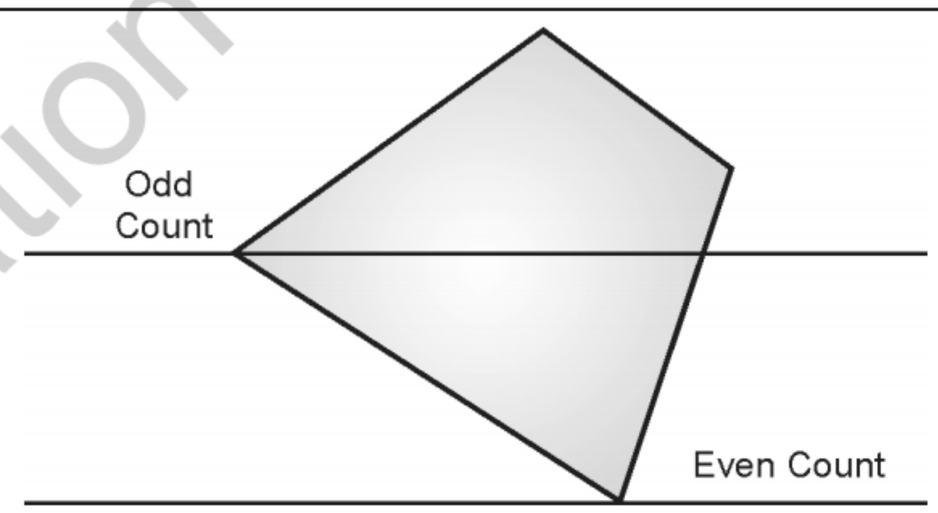


Fig. 4.2.2 : Remedy to shared edges

4.2.2 Winding Number Rule

Q. Explain the term with example - Winding Method.

SPPU - Dec. 17, 2 Marks

- Winding number method is another way to define the visibility of pixel.
- Like the even-odd method, draw a straight line from point to be examined to any known exterior point.
- Visit outermost boundary of the polygon in an anti-clockwise direction and inner boundary (if exists) in a clockwise direction.

Technknowledge Publications

- Find the intersection points of line with polygon boundaries and check how many times polygon edge crosses the line while traversing from left to right and right to left.

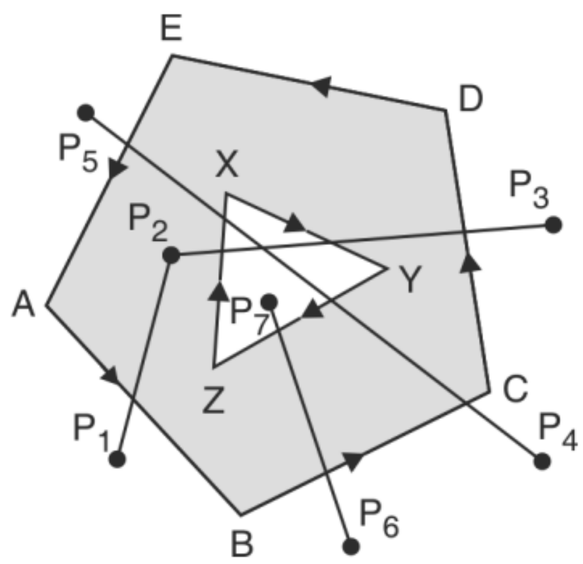


Fig. 4.2.3 : Winding number method

- Take the difference between these two counts, if the difference is zero, the point is outside the polygon, otherwise, it is considered as an inner point.
- Consider the point P_2, (Refer Fig. 4.2.3) which is joined with exterior point P_1.
- Polygon edge AB intersects P_1P_2 while traversing from left to right. So point P_2 is the interior point.
- Edge YZ intersect line P_6P_7 from right to left and edge BC intersects it while traversing from left to right. The difference of intersection count is zero and hence P_7 is considered as an outside pixel.
- Similarly edges EA and YZ intersect line P_4P_5 while traversing from right to left and edges ZX and BC intersects P_4P_5 while traversing from left to right. The difference is zero, so point P_4 is outside the polygon.

4.3 Polygon Filling

4.3.1 Introduction

Q. Explain the term with example - Polygon filling with pattern. **SPPU - Dec. 17, 2 Marks**

Q. What is polygon filling?

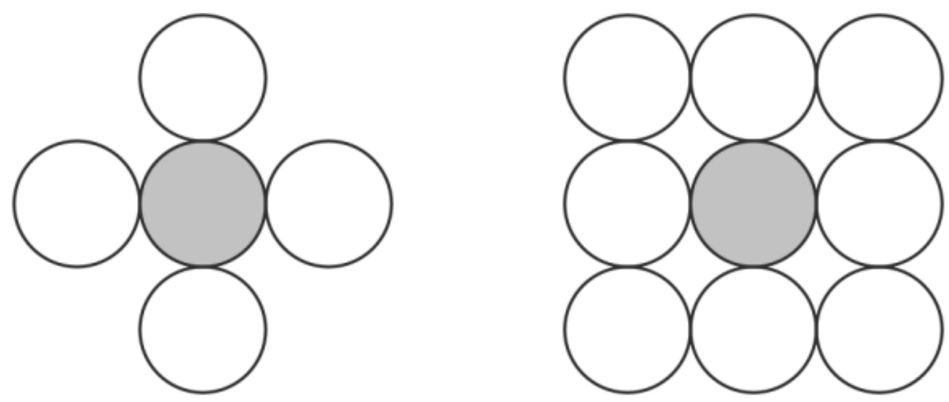
SPPU - May 18, May 19, Dec. 19, 2 Marks

- Most of the graphics packages provide area filling primitives. There are two common approaches for area filling in a raster system. One way is to determine the visible span of each scan line of the polygon. This method is used to fill simple shapes like ellipse, circle, rectangle etc. Scan line polygon filling approach is the example of this kind of area filing method.
- Another approach is to start with any interior pixel and keep filling region outwards until it hits the boundary. This approach is applicable for filling shapes with complex boundaries and interactive painting system. Flood fill and boundary fill fall in the second category.
- Polygon filling algorithms fill the background of polygons. Most popular algorithms of polygon filling are listed and discussed in the following section.
- Pixel level algorithms
  - Seed fill algorithms
    - (i) Boundary fill algorithm
    - (ii) Flood fill algorithm
  - Edge fill algorithm
  - Fence fill algorithm
- Geometry level algorithms
- Scan line polygon filling algorithm

4.3.2 Pixel Connectivity

- Connectivity defines the relationship between pixel under consideration and its neighbor pixels.
- Most common and useful types of connectivity are shown in Fig. 4.3.1.
- Pixel is called four connected if it has four neighbors and it is called 8-connected if it has eight neighbors.
- Fig. 4.3.1 defines the 4-connected and 8-connected component.

Tech Knowledge Publications



(a) 4 Connected (b) 8 Connected

Fig. 4.3.1 : Pixel connectivity

4.3.3 Boundary Fill

Q. Explain polygon fill with seed fill algorithm.

SPPU - May 18, 4 Marks

Q. Explain Boundary fill algorithm.

SPPU - May 19, Dec. 19, 4 Marks

Working Mechanism

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

Algorithm

- Algorithm for boundary fill method is described here :

```
Algorithm    BOUNDARY_FILL(x, y, fillColor, boundaryColor)
                                  // filColor: Color to be filled in polygon
                                  //boundaryColor: Color of polygon boundary
  Current ← getColor (x, y)       // Retrieve the color of current pixel
  if current ≠ fillColor && current ≠ boundaryColor then
    putPixel(x, y, fillColor)
    BOUNDARY_FILL (x + 1, y, fillColor, boundaryColor)
    BOUNDARY_FILL (x – 1, y, fillColor, boundaryColor)
    BOUNDARY_FILL (x, y + 1, fillColor, boundaryColor)
    BOUNDARY_FILL (x, y – 1, fillColor, boundaryColor)
  end
```

Tech Knowledge Publications

- This algorithm may not work properly if some of the pixels are already filled, because algorithm returns when the current pixel is either boundary pixel or it is filled.
- To avoid this, we shall set the color of all interior pixels to background color.

Advantages

- Simple.
- Easy to implement.
- Works for any type of polygons.

Disadvantages

- Require extensive stacking due to recursion.
- Does not work if the boundary is specified with multiple colors.
- Need to specify seed point contained within the polygon.
- Polygon should be filled with the background color.
- Require more memory.
- Not suitable for the large polygon.

Filling Region with 4 and 8 Connectivity

- Filling of polygon depends on types of connectivity employed. The 4-connected approach does four recursive calls for each of its four nearest neighbors.
- The 8-connected approach makes eight calls. The 4-connected approach requires less memory but sometimes fails to fill certain parts of the polygon.
- In Fig. 4.3.2, if the 4-connected approach is used then it terminates after reaching to pixel marked 'x'. It cannot jump into another part of the polygon. 8-connected approach can jump diagonally too. Hence it would be able to fill both the regions of the polygon.

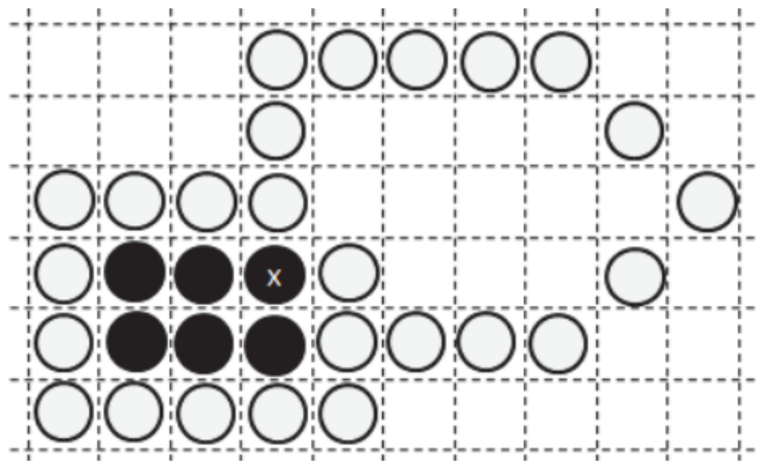


Fig. 4.3.2

Example

Ex. 4.3.1 : Which algorithm cannot be used to fill region R2 which is bounded by Blue color and Red color boundary, justify.

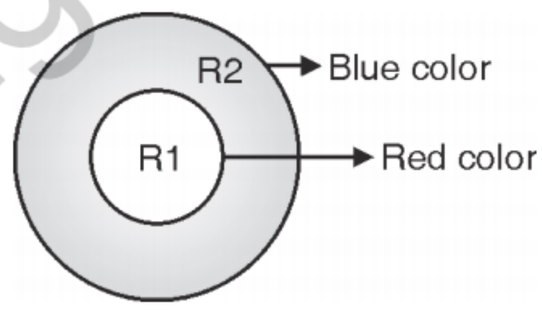


Fig. P. 4.3.1

Soln. :

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

4.3.4 Flood Fill

Q. Write algorithm to fill the polygon area using flood fill method. **SPPU - May 15, Dec. 15, 4 Marks**

Working Mechanism

- Many realistic objects have a different colored boundaries. Boundary fill algorithm cannot fill the polygon with multi-colored boundary.
- Flood fill algorithm can handle this issue.

Tech Knowledge Publications

- The algorithm starts with the interior pixel, fill color and old color. If the color of the current pixel is the same as old color then it fills the pixel with fill color and examines its neighbors recursively.
- Like boundary fill algorithm, if the interior pixel is already filled with some other color, flood fill algorithm fails. To handle this problem, we shall first paint all interior pixels with background (old) color.

Algorithm

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

Advantages

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

Disadvantages

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

4.3.5 Optimization

- Boundary fill and flood fill, both are recursive approaches. On every call, they make other four or eight calls to the neighbours of the current pixel, this creates extensive stacking. The system runs out of memory if polygon to be filled is very large.
- We will discuss a more efficient method which will minimize the recursive calls. Instead of pushing all neighbors on the stack, just push neighbor scan line numbers on the stack and fill the entire span of the current scan line.
- A sequence of figures shown in Fig. 4.3.3 explains the concept. In Fig. 4.3.3(a), random seed pixel is selected. The span of the current scan line is filled up and addresses of its upper and lower scan line are pushed on to the stack. The procedure always pops topmost scan line number from the stack, fills it horizontally and pushes them above and below scan line numbers if they are not already processed. This approach greatly simplifies the processing and reduces memory requirement.

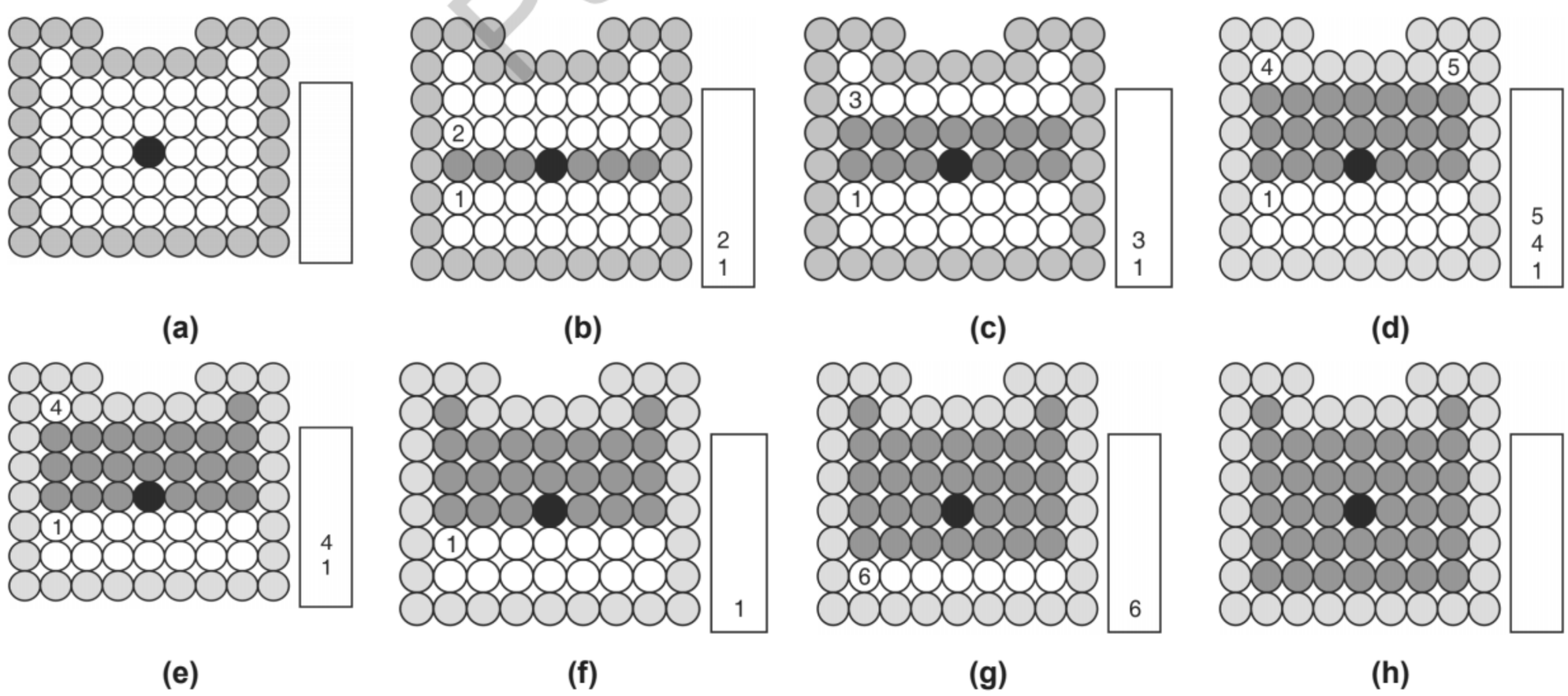


Fig. 4.3.3 : Optimized seed fill approach

Tech Knowledge Publications

4.3.6 Scan Line Fill

Q. Explain in detail scan line polygon filling algorithm.

SPPU - May 17, May 18, May 19, 5 Marks

Working Principle

- Seed fill approaches are memory and time intense due to their recursive structure. Computation can be reduced by determining the entire visible span instead of checking the visibility of individual pixel.
- Scan line fill algorithm computes the visible span of each scan line within polygon boundaries. It works with convex, concave, self-intersecting and polygon with holes.

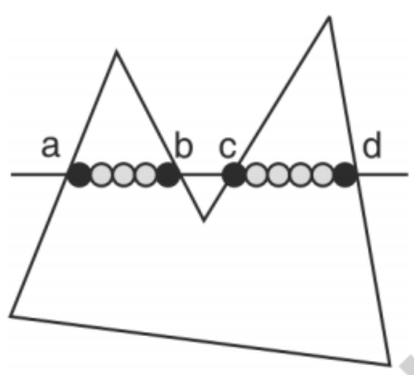


Fig. 4.3.4 : Visible span of the scan line

- The scan line intersects polygon boundary at a, b, c and d (Refer Fig. 4.3.4). We shall set appropriate values of all intersection points to fill the polygon properly.
- Midpoint line clipping algorithm can be used to determine the intersection point of the scan line with the edges.
- This approach selects the pixel which is closer to the edge, irrespective of on which side of polygon it lies. If the edge is shared between two polygons, and the outer pixel is selected for drawing, this would look odd if a neighbor polygon has a different color. So even if the exterior pixel is closer to the edge, we should reject it for better rasterization.

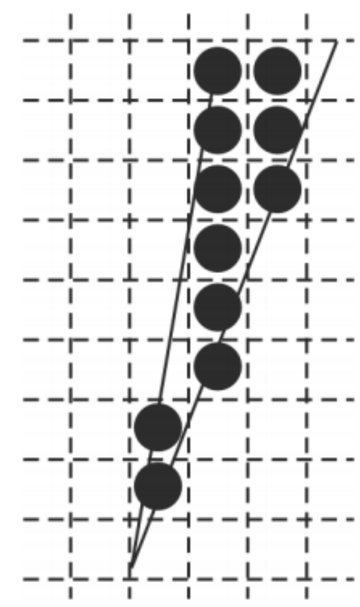


Fig. 4.3.5 : Sliver

- A simple solution to this problem is to round up the intersection pixel coordinate if scan line is entering in polygon region, and round down if it is leaving the polygon.
- Scan line polygon filling approach works as follow :
  - Find intersections of scan line with polygon edges.
  - Sort all intersections on their x-coordinate.
  - Fill the span using a pair of sorted intersection points.
- When scan line intersects the polygon boundary, parity is odd and when it intersects again parity becomes even. Fill the span between odd to even parity. Initially parity would be even.
- For example, in Fig. 4.3.4 parity is initially even. When scanline intersects the polygon edge at point a, parity becomes odd. When the scanline intersects the edge at point b, parity becomes odd. So fill the span from a to b. In similar way parity will be odd and even for points c and d so span from c to d will be filled up but span b to c will not.
- Problem arises when the scanline intersects the shared vertex. This problem can be handled in similar way to odd-even test. If shared edges are on the same side of scan line, parity is set to even and if shared edges are on opposite side of scanline then set the parity to odd.
- Horizontal edge is the special case. Parity changes on every pixel for horizontal edge and every alternative pixel is illuminated. To avoid this effect, horizontal edges are not considered for processing.

Tech Knowledge Publications

- Another problem with this approach is a sliver. When the angle between adjacent edges is very small, consecutive scan lines produce the same span, which results in aliasing effect. This is called a **sliver**. Fig. 4.3.5 demonstrates the sliver.
- Scan line approach takes advantage of edge coherence. Most of the time, edges intersecting scan line k are also intersected by scan line k + 1. If the current intersection point is (x_k, y_k), the intersection point on the next scan line is computed as,

$$y_k + 1 = y_k + 1$$

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

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

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

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

Algorithm

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

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

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

Advantages

- It takes the advantage of edge coherence, which results in to faster execution.
- It only draws visible pixels.
- It requires less memory.
- This is a common choice in software routine.

Disadvantages

- It is more complex.
- It requires all polygon should be sent to render before drawing.

Example

Ex. 4.3.2 : Determine the visible span for each scan line for given polygon using scan line polygon filling algorithm.

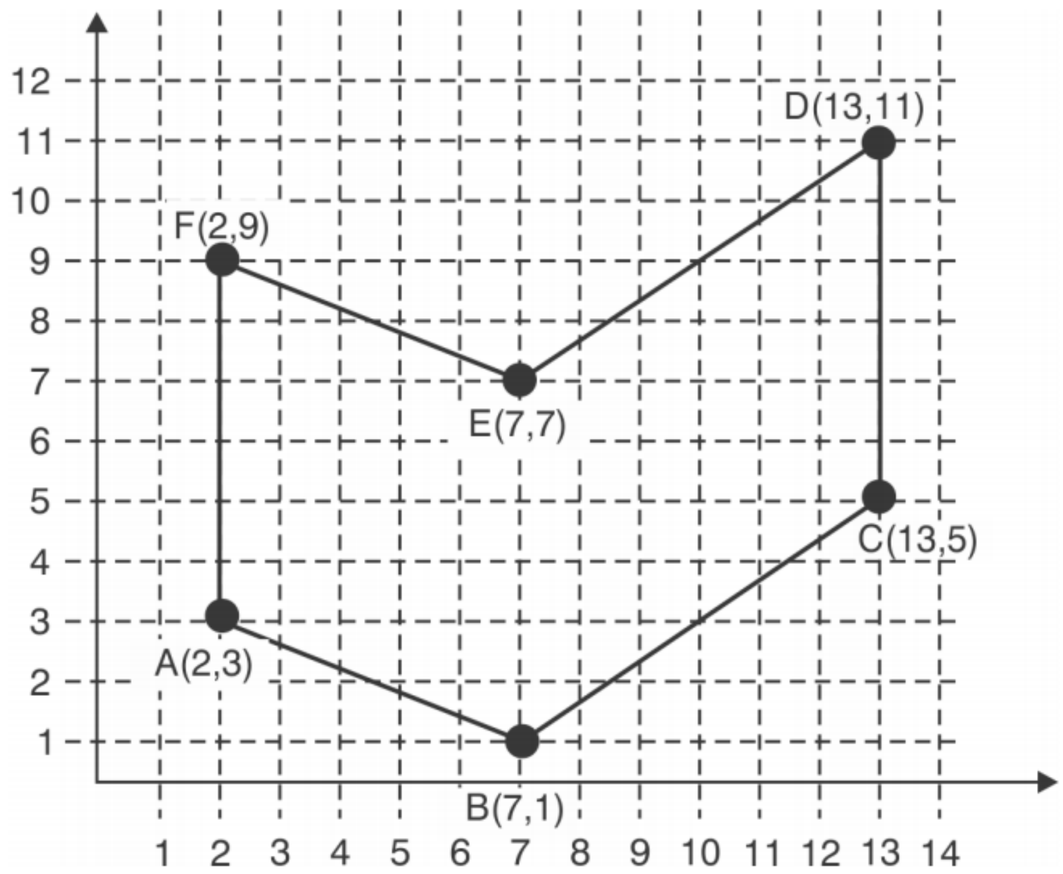


Fig. P. 4.3.2

Technknowledge Publications

Soln. :

Edge table for a given polygon is shown in Fig. P. 4.3.2(a).

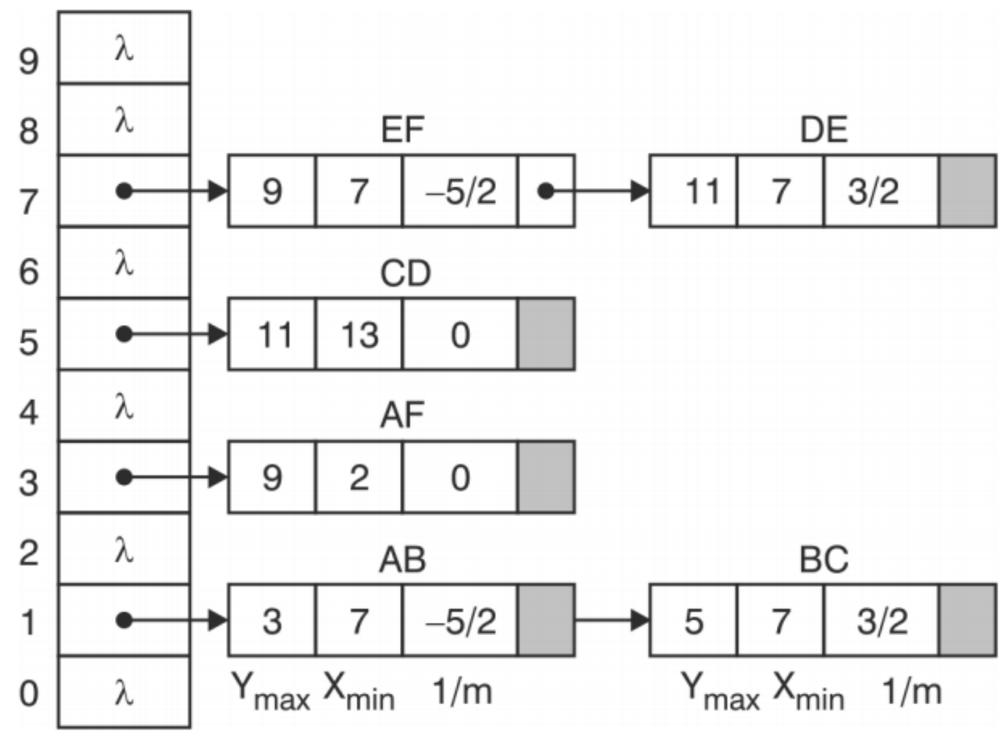


Fig. P. 4.3.2(a)

Let us compute the visible span for each scan line.

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

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

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

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

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

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

Scan line Y = 2 : No new Edges are added

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

$X_{AB} = X_{AB} + (1 / m) = 7 - 2.5 = 4.5$,

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

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

$X_{BC} = X_{BC} + (1 / m) = 7 + 1.5 = 8.5$,

So $(X_{BC}, Y_{SCAN}) = (8, 2)$

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

Scan line Y = 3 : AF Edge is added

Edge AB : $Y_{scan} \geq Y_{max(AB)}$, So remove AB

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

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

So $(X_{BC}, Y_{SCAN}) = (10, 3)$

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

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

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

Scan line Y = 4 : No New Edges are added

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

$X_{BC} = X_{BC} + (1 / m) = 10 + 1.5 = 11.5$,

So $(X_{BC}, Y_{SCAN}) = (11, 4)$

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

$X_{AF} = X_{AF} + (1 / m) = 2 + 0 = 2$,

So $(X_{AF}, Y_{SCAN}) = (2, 4)$

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

Scan line Y = 5 : CD Edge is added

Edge BC : $Y_{scan} \geq Y_{max(BC)}$, So remove BC

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

$X_{AF} = X_{AF} + (1 / m) = 2 + 0 = 2$,

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

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

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

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

Scan line Y = 6 : No New Edges are added

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

$X_{AF} = X_{AF} + (1 / m) = 2 + 0 = 2$,

So $(X_{AF}, Y_{SCAN}) = (2, 6)$

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

$X_{CD} = X_{CD} + (1 / m) = 13 + 0 = 13$,

So $(X_{CD}, Y_{SCAN}) = (13, 6)$

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

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

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

$X_{AF} = X_{AF} + (1 / m) = 2 + 0 = 2$,

So $(X_{AF}, Y_{SCAN}) = (2, 7)$

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

$X_{CD} = X_{CD} + (1 / m) = 13 + 0 = 13$,

So $(X_{CD}, Y_{SCAN}) = (13, 7)$

Technknowledge Publication

EF and DE Edges are added.

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

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

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

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

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

Scan line Y = 8 : No New Edges are added

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

$X_{AF} = X_{AF} + (1 / m) = 2 + 0 = 2$,

So $(X_{AF}, Y_{SCAN}) = (2, 8)$

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

$X_{CD} = X_{CD} + (1 / m) = 13 + 0 = 13$,

So $(X_{CD}, Y_{SCAN}) = (13, 8)$

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

$X_{EF} = X_{EF} + (1 / m) = 7 - 2.5 = 4.5$,

So $(X_{EF}, Y_{SCAN}) = (4, 8)$

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

$X_{DE} = X_{DE} + (1 / m) = 7 + 1.5 = 8.5$,

So $(X_{DE}, Y_{SCAN}) = (9, 8)$

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

Scan line Y = 9 : No New Edges are added

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

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

$X_{CD} = X_{CD} + (1 / m) = 13 + 0 = 13$,

So $(X_{CD}, Y_{SCAN}) = (13, 9)$

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

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

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

So $(X_{DE}, Y_{SCAN}) = (10, 9)$

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

Scan line Y = 10 : No New Edges are added

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

$X_{CD} = X_{CD} + (1 / m) = 13 + 0 = 13$,

So $(X_{CD}, Y_{SCAN}) = (13, 10)$

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

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

So $(X_{DE}, Y_{SCAN}) = (12, 10)$

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

Scan line Y = 11 : No New Edges are added

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

Edge DE : $Y_{scan} < Y_{max(DE)}$, So remove DE

AET is empty, so STOP

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

4.3.7 Seed Fill vs Scanline Algorithm

Sr. No.	Seed Fill Algorithm	Scan Line Algorithm
1.	Simple to implement.	Very complex.
2.	Operates in screen space.	Operates in screen and / or object space.
3.	Require system call to get pixel intensity.	It is device-independent.
4.	Need to specify seed point contained within the polygon.	No need to specify seed point.

Techknowledge Publications

Sr. No.	Seed Fill Algorithm	Scan Line Algorithm
5.	Need extensive stacking.	No need for stacking.
6.	Generally used in graphics packages.	Typically used in interactive rendering.
7.	Not suitable for Z-Buffer.	Suitable for Z-Buffer.

4.4 Windowing and Clipping

4.4.1 Viewing Transformations

Q. What is viewing transformation?

SPPU - Dec. 19, 6 Marks

- Graphics packages allow the user to specify which part of the scene is to be displayed.
- The process of selecting the part of the real-world scene to display it on some device is called **windowing.**
- **Clipping window** is the rectangular window against which visibility of scene object is tested.
- **Clipping** is the process of deciding and removing the portion of the object which is outside the clipping window.
- Part of the primitive outside the clipping window is not drawn. Clipping region may be of any arbitrary shape, but to simplify the clipping procedure, rectangle region is used.
- User can select single or clipping multiple windows simultaneously. Clipping reduces the computation by not processing the unnecessary part of the object.
- There are two approaches for clipping the primitives :

1. **Scan conversion approach :** In this approach visibility of each pixel of an object is tested, and the pixel is drawn if it is visible. In other words, we perform the clipping during scan conversion of the object. This approach is simple but does extensive calculations.
2. **Analytical approach :** In this approach, before scan converting the object, visible part of the primitives is calculated analytically. This approach does not perform the check for individual pixel. It reduces the computation but it is complex.

- The visible region of the scene inside the clipping window is displayed on some device. More formally, the world coordinate area which is selected for display is called a **window**.
- An area on the display device to which window is mapped is called **viewport**.
- Window defines *what* is to be displayed from the scene and viewport defines *where* it is to be displayed on the screen.
- In most of the graphics package window and viewport is a rectangle, with edges parallel to the principal axis.
- The process of mapping the part of world coordinate scene to device coordinate is referred to as a **viewing transformation** or **window to viewport transformation** or **windowing transformation**.
- Fig. 4.4.1 illustrates the concept and relationship of window and viewport.

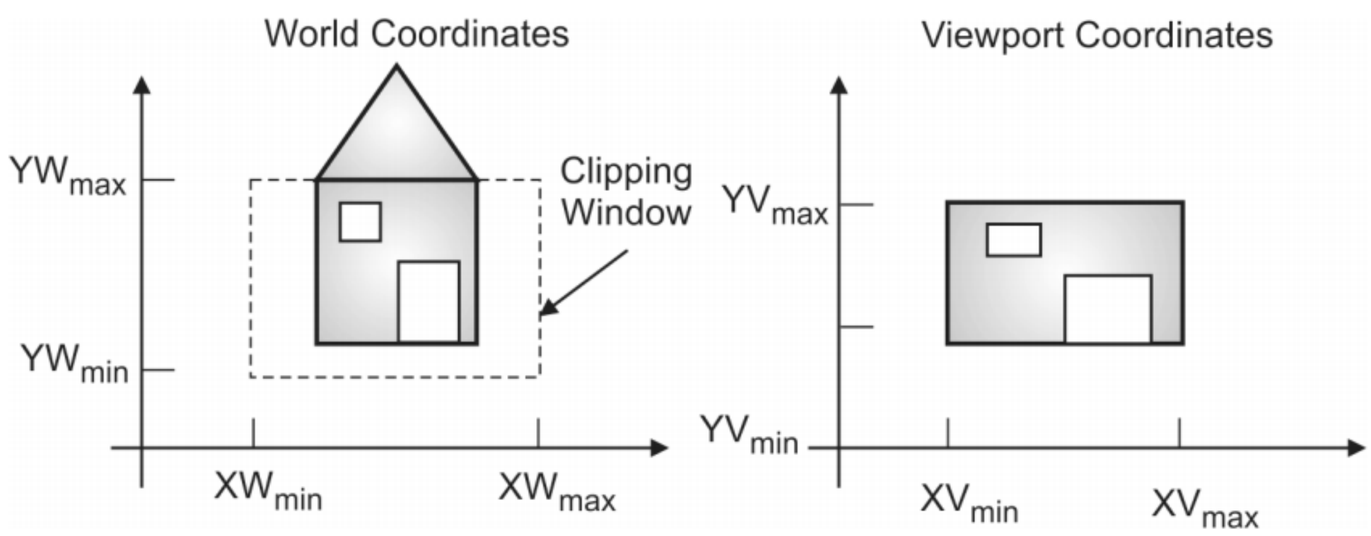


Fig. 4.4.1 : Window to viewport transformation

- We can define the scene by creating objects with their actual dimensions, called **modelling coordinates** or **local coordinates** or **master coordinates** i.e. (x_{mc}, y_{mc}).
- Once the object is designed, it is placed at an appropriate position in the scene using **world coordinates** i.e. (x_{wc}, y_{wc}).
- The scene may be displayed on different devices, so it must be mapped to the proper dimension to render it properly. World coordinates are then represented in **normalized coordinate** (x_{nc}, y_{nc}), usually between 0 and 1. This makes the system independent of the various display devices.
- These normalized coordinate can be easily scaled to fit on any size of the display by multiplying it with a proper scaling factor that is known as **device coordinate** (x_{dc}, y_{dc}). Fig. 4.4.2 shows sequence of coordinate transformations.

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

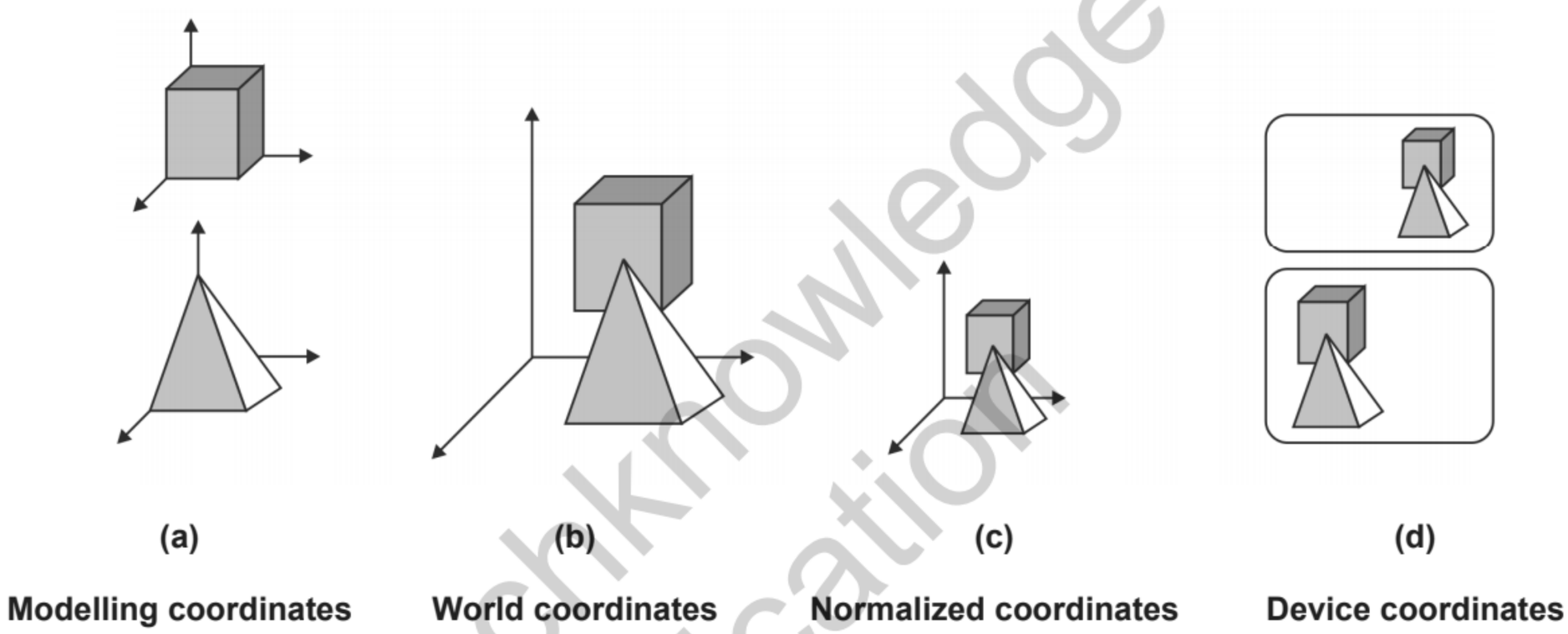


Fig. 4.4.2 : Coordinate transformation

The viewing pipeline is depicted in Fig. 4.4.3.

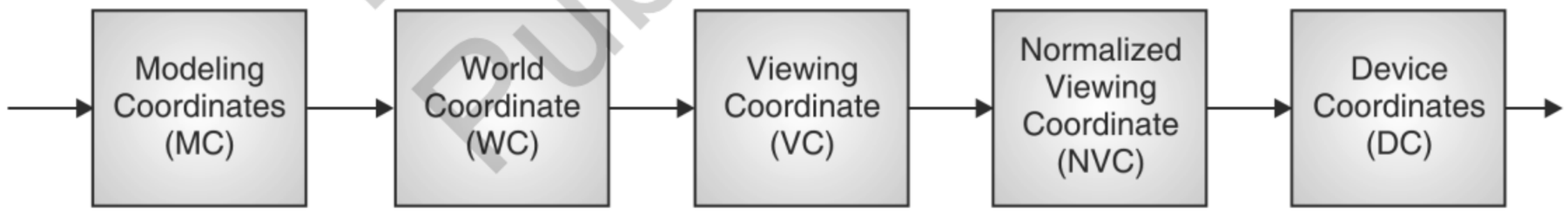


Fig. 4.4.3 : Steps for the window to viewport transformation

- As shown in Fig. 4.4.4, we can view the scene on the device at different locations by changing the position of viewport.

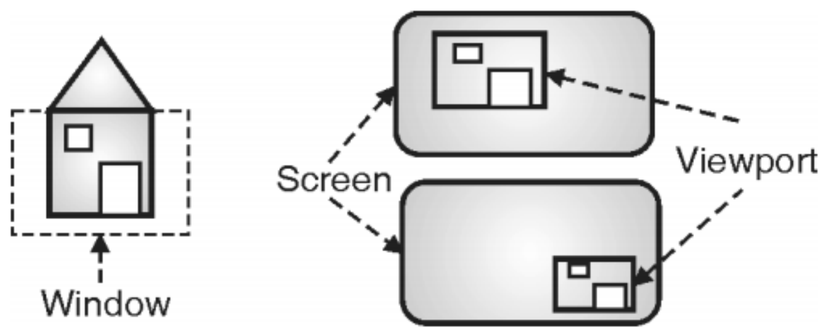


Fig. 4.4.4 : Mapping same window to different viewports

- As shown in Fig. 4.4.4, the same window is mapped to viewports with different size and location on the screen.
- To achieve the zoom-in effect, we shall map window with different size on the fixed-size viewport. If the window size is made smaller, we get a zoom-in effect and if the window size is made larger, we get a zoom out effect.

TechKnowledge Publications

– **Panning** is achieved by moving a fixed size window over various positions in the scene.

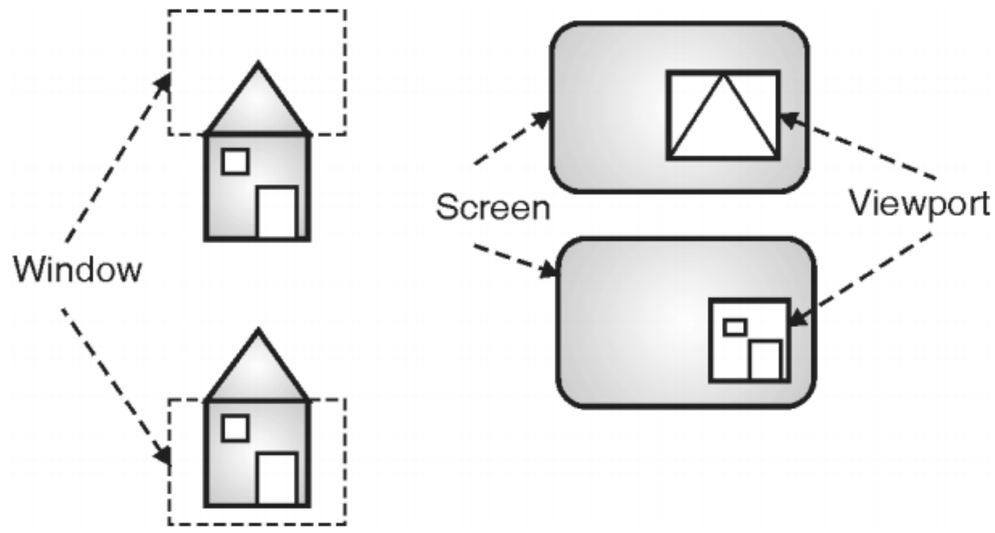


Fig. 4.4.5 : Effect of panning

4.4.2 Window to Viewport Transformation

– Physical description of the object goes through multiple sequences as shown in Fig. 4.4.6.

– Window to the viewport to transformation is necessary because the size of the window and viewport may not be the same all the time. So actual scene selected by window needs to be rescaled to fit it in the viewport.

– Let (XW_{min}, YW_{min}) and (XW_{max}, YW_{max}) represent the lower left and upper-top corner points of clipping window, respectively.

– And let (XV_{min}, YV_{min}) and (XV_{max}, YV_{max}) represent the lower left and upper-top corner points of the viewport, respectively.

– As shown in Fig. 4.4.6, point (xw, yw) in the window is to be mapped to point (xv, yv) in the viewport.

– To maintain the same relative placement in the viewport as in a window,we normalize both.

$$\frac{xv - XV_{min}}{XV_{max} - XV_{min}} = \frac{xw - XW_{min}}{XW_{max} - XW_{min}}$$

$$xv - XV_{min} = (XV_{max} - XV_{min}) \cdot \frac{xw - XW_{min}}{XW_{max} - XW_{min}}$$

$$xv = XV_{min} + (xw - XW_{min}) \cdot S_x$$

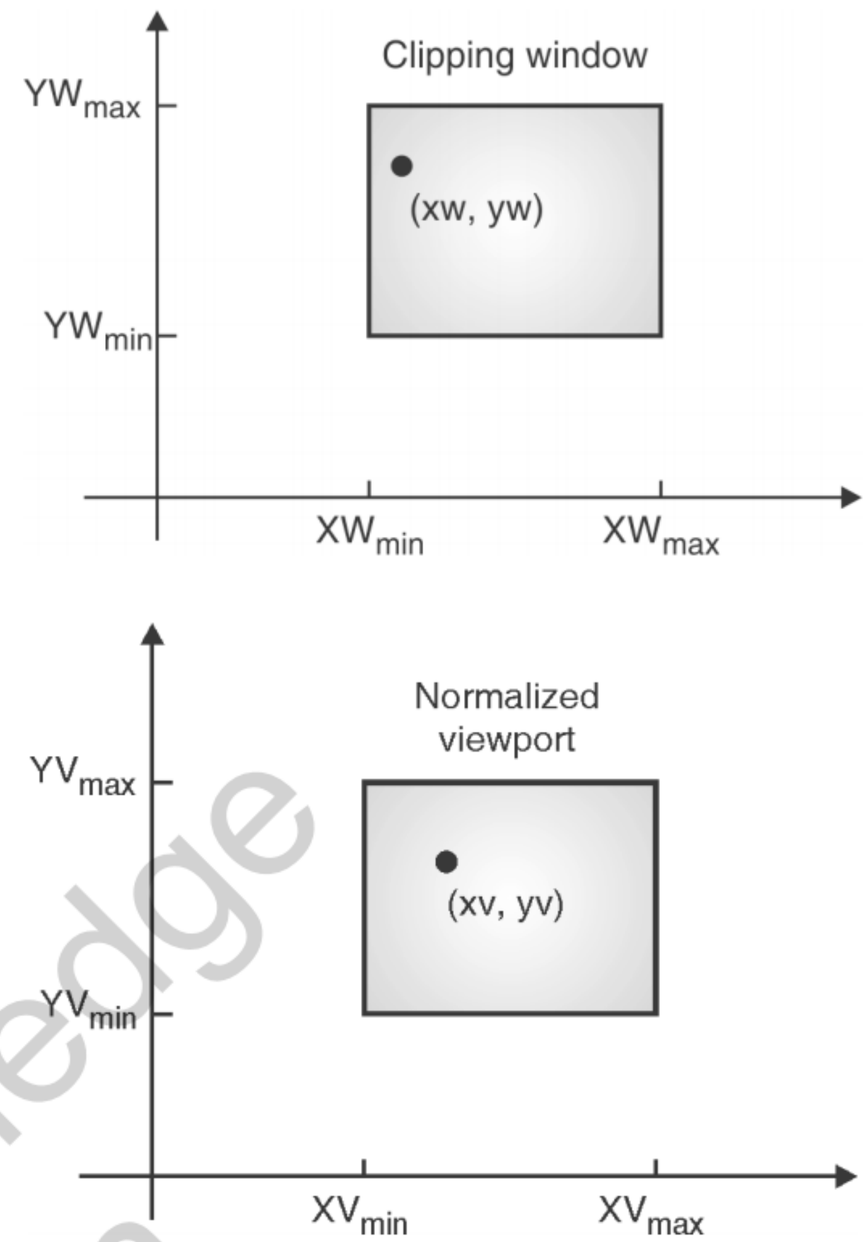


Fig. 4.4.6 : Relation between window and viewport

Similarly,

$$\frac{yv - YV_{min}}{YV_{max} - YV_{min}} = \frac{yw - YW_{min}}{YW_{max} - YW_{min}}$$

$$yv - YV_{min} = (YV_{max} - YV_{min}) \cdot \frac{yw - YW_{min}}{YW_{max} - YW_{min}}$$

$$yv = YV_{min} + (yw - YW_{min}) \cdot S_y$$

Where the scaling factors are,

$$S_x = \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}}$$

$$S_y = \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}}$$

Matrix Representation

– If the lower-left corner of the window is not at the origin, we should first translate it to the origin before we map the points in the window to the viewport.

– Window to viewport transformation is achieved by the following three steps :

Technknowledge Publications

(a) Translate the lower-left point of window to the origin.

Translation matrix is defined as,

$$T = \begin{bmatrix} 1 & 0 & -XW_{min} \\ 0 & 1 & -YW_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

(b) Apply scaling to map the scene to the viewport.

Scaling matrix is defined as,

$$S = \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & 0 \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(c) Inverse translation in the viewport.

Inverse translation matrix is defined as,

$$T^{-1} = \begin{bmatrix} 1 & 0 & XV_{min} \\ 0 & 1 & YV_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

– The composite transformation matrix for the window to viewport transformation is given as,

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

$$M = \begin{bmatrix} 1 & 0 & XV_{min} \\ 0 & 1 & YV_{min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & 0 \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -XW_{min} \\ 0 & 1 & -YW_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} & 0 & -XW_{min} \cdot \frac{XV_{max} - XV_{min}}{XW_{max} - XW_{min}} + XV_{min} \\ 0 & \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} & -YW_{min} \cdot \frac{YV_{max} - YV_{min}}{YW_{max} - YW_{min}} + YV_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

4.5 2-D Clipping

We can perform clipping operations to clip various primitives like point, line, polygon or even text. In the next few sections, we will discuss different algorithms to clip such basic primitives. Clipping operations determines the visible span of the primitive and maps it to the viewport.

The taxonomy for 2D clipping operations is described in Fig. 4.5.1.

Tech Knowledge Publications

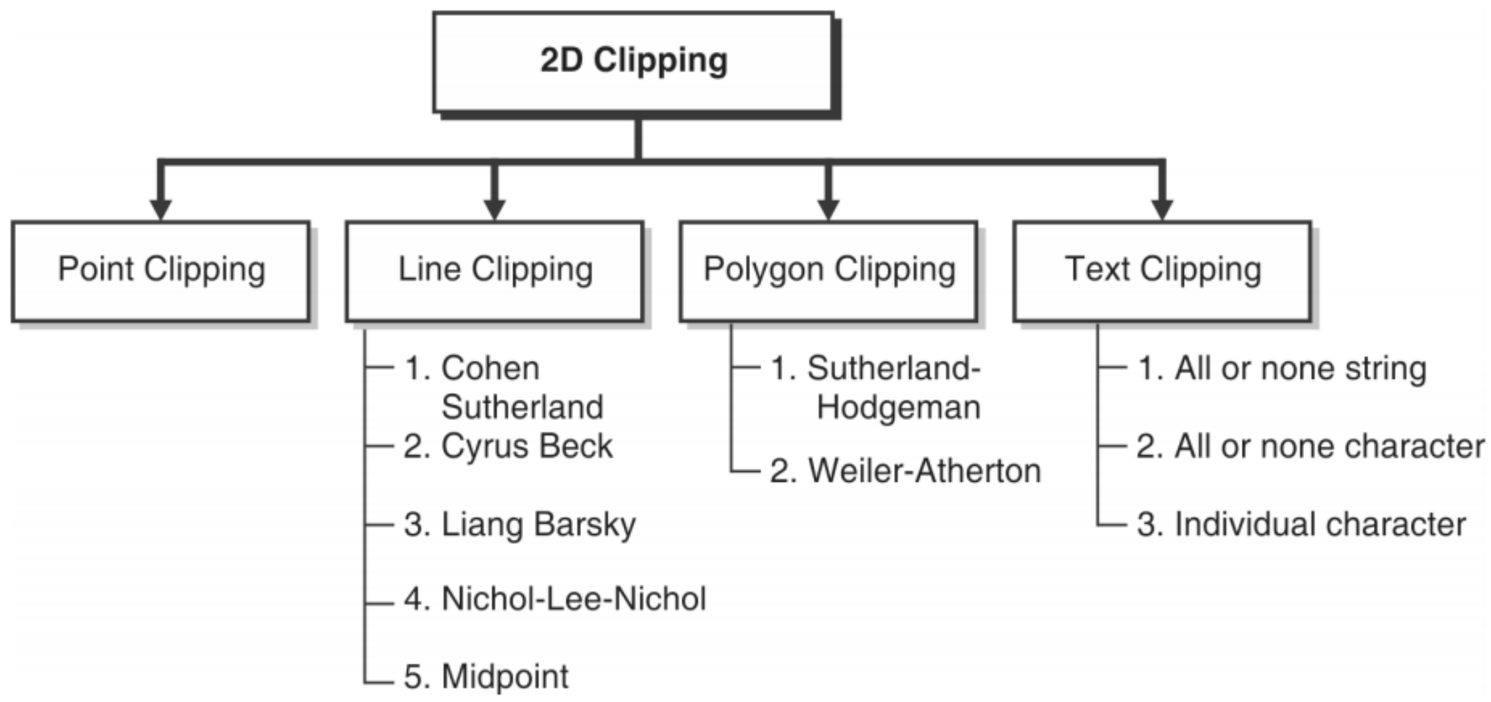


Fig. 4.5.1 : Taxonomy of 2D clipping operations

4.6 Point Clipping

- Before we discuss line clipping, let's look at the simpler problem of clipping individual point.

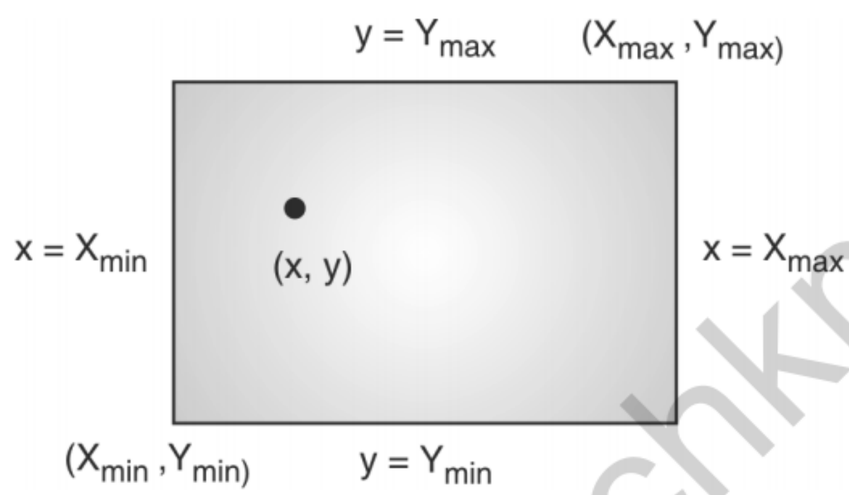


Fig. 4.6.1 : Point clipping

- Point clipping is very simple. It's a trivial case of acceptance or rejection. The point can be either fully inside or fully outside the clipping region.The point on the clipping window boundary is considered inside.
- Let, (X_{min}, Y_{min}) and (X_{max}, Y_{max}) represents the lower-left and top-right corners of the clipping window, respectively.
- Point (x, y) is inside the region only if the following four inequalities are true,

$$X_{min} \le x \le X_{max}$$

$$Y_{min} \le y \le Y_{max}$$

- For point being inside the clipping window, all four conditions must hold. If any of the four inequalities does not hold, the point is outside the clipping rectangle.

4.7 Line Clipping

- The line is widely used primitive in engineering drawing. Most of the engineering shapes contain lines.
- When we display the scene on the monitor screen, we should not render the portion of the scene which is outside the clipping window. It is essential to determine which portion of the line is inside the clipping region.
- Various cases of line membership are discussed here. Consider the clipping rectangle PQRS in Fig. 4.7.1. Scene contains four lines. If both endpoints of the line are within a rectangle, the line is completely visible. Otherwise, the line may be partially visible or completely outside the window.

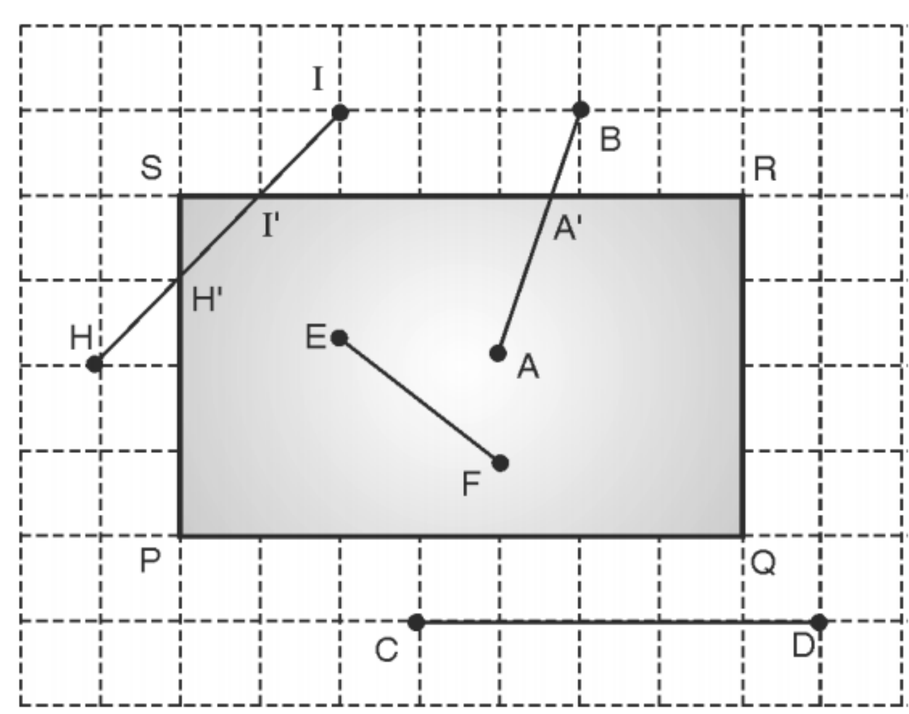


Fig. 4.7.1

TechKnowledge Publications

- Line EF is completely visible and will be displayed as it is.
- One endpoint of line AB is outside the clipping region. The intersection point of line AB with the top edge is A′. Hence segment AA′ will be displayed and A′B will be clipped.
- Line CD is completely outside the clipping window so it won't be displayed at all.
- Line HI intersects left and top border of clipping regions. Intersections with respective boundaries are H′ and I′. So only segment H′I′ will be displayed. Segments HH' and I′I would be clipped.
- Fig. 4.7.2 shows the output of the scene after the clipping procedure.

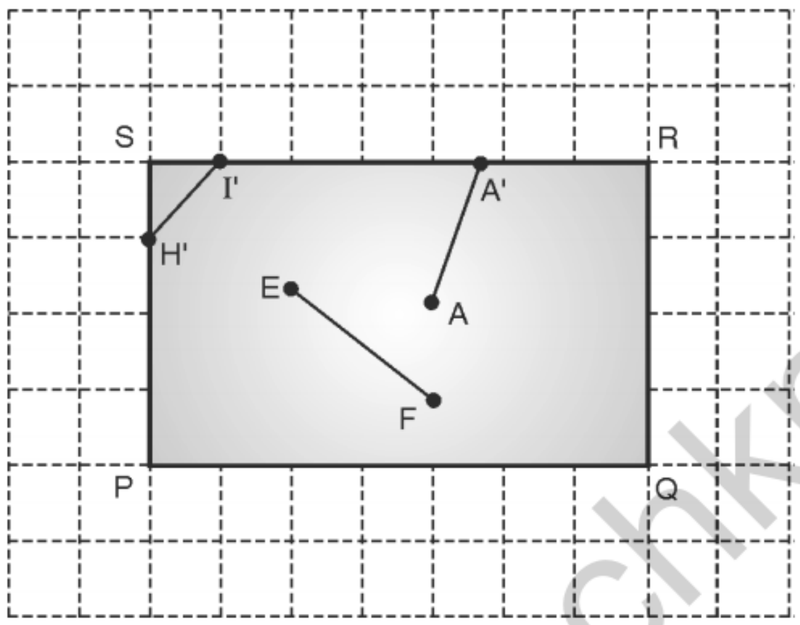


Fig. 4.7.2

4.7.1 Cohen - Sutherland Line Clipping Algorithm

Q. Explain Cohen-Sutherland line clipping algorithm with example.

SPPU - May 15, May 17, Dec. 18, May 19, 4 Marks

Q. Write and explain with an example Cohen-Sutherland line clipping algorithm.

SPPU - Dec. 15, May 16, 8 Marks

Working Mechanism

- Cohen-Sutherland is a 2D line clipping algorithm. The main advantage of the algorithm is that it vastly reduces the number of line intersections that must be calculated in scan conversion approach.
- It operates in two phases :
  1. Region code generation
  2. Clipping

1. Region Code Generation

- It divides the plane into nine regions as shown in Fig. 4.7.4 and determines the visible portion of the line using outcodes (region code) of the endpoints. Outcode is a four-bit number. Values of these bits are set 1 if conditions shown in Fig. 4.7.3 are true for a particular bit.

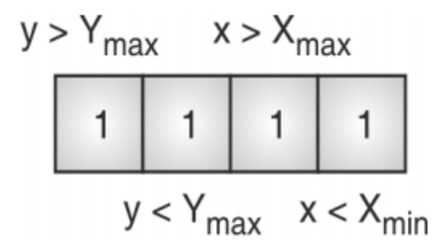


Fig. 4.7.3 : Setting the bits of outcode

- For any endpoint (x, y) of a line, the outcode is set according to the following conditions:

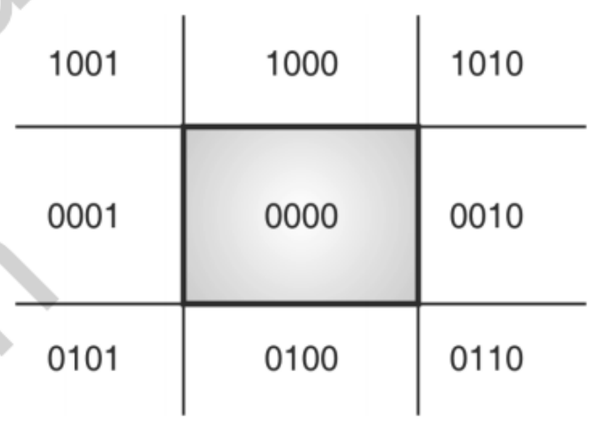


Fig. 4.7.4 : Outcodes for all nine regions

  - Set first bit (most significant bit) if a point lies above window i.e. $y > Y_{max}$.
  - Set second bit if a point lies below window i.e.$y < Y_{min}$.
  - Set third bit if a point lies to the right of window i.e. $x > X_{max}$.
  - Set fourth bit (least significant bit) if a point lies to left of window i.e. $x < X_{min}$.
- If the point is inside the clipping window, none of the above conditions would be true. So, outcode of endpoint inside the clipping region would be 0000. Outcodes of all nine regions are shown in Fig. 4.7.4.

2. Clipping Procedure

- The algorithm quickly detects two trivial cases.
- If both endpoints of a line lie inside the window, the entire line is visible. It is trivially accepted and needs no clipping.

TechKnowledge Publications

- On the other hand, if both endpoints of a line lie completely on one side of the window, the line lies completely outside the window. It is completely rejected.
- After calculating outcodes of both endpoints of the line, logical OR-AND conditions are evaluated to check trivial acceptance, rejection or partial visibility. We will discuss three possible cases.

Case I

- The line is completely inside clipping window if logical OR operation of outcodes yields to 0000. For example, line AB in Fig. 4.7.5 is trivially accepted.
- In short, if outcodes of both endpoints are 0000, the line is fully visible.

Case II

- If logical OR of the outcodes is not 0000, there are two possibilities, the line may be partially visible or it may be completely outside.
- The line is completely outside the clipping window if logical AND operation of outcomes does not yield to 0000. Above both tests are performed in the same order, i.e. Case II is applicable only if Case I is not true.

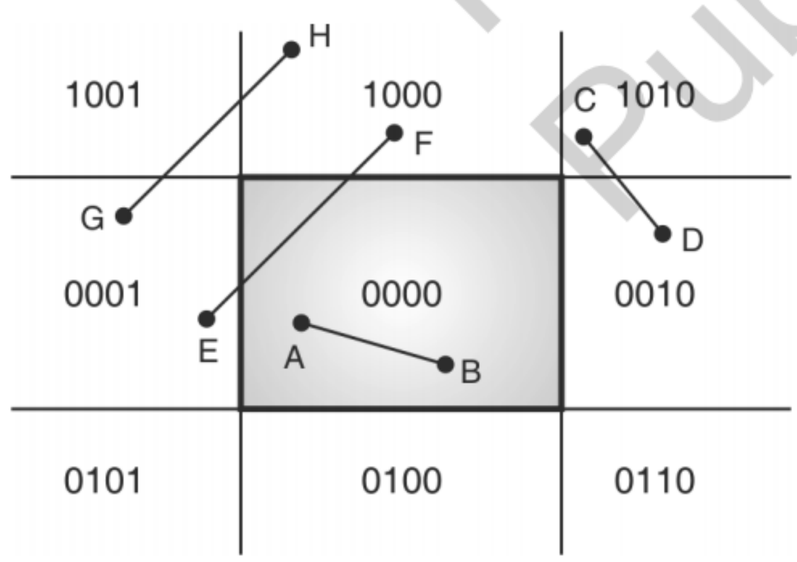


Fig. 4.7.5 : Different clipping cases

- For example, one endpoint of the line CD in Fig. 4.7.5 has outcode 1010 while the other endpoint has a code 0010. Logical OR is 1010, so it is not fully visible. The logical AND would be 0010 which indicates the line segment lies completely outside of the window.

Case III

- Outcodes of endpoints of the line, EF are 1000 and 0001. Logical OR of endpoints is 1001, so the line is not fully visible. The logical AND would be 0000, and the line could not be trivially rejected.
- In this case, we need to process line further. If logical AND operation of both endpoints is 0000, the line may or may not pass from clipping window. For example, like line EF, logical AND of outcodes of endpoints of line GH is also 0000, but that line is completely outside.
- If the line is not trivially accepted or rejected, then compare endpoints with window boundary to determine how much line segment can be discarded. The intersection point of line with clipping region edge can be computed as follow.

  Consider the line with endpoints(x_1, y_1) and (x_2, y_2). The slope of a line is given as,

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

- Using explicit representation of the line, $y = mx + c$, we can compute Y-intercept of the line as, $c = y - mx$, where (x, y) is any point on the line. Both endpoints of the line are known, so we can put anyone in this equation to compute c, let's put (x_1, y_1), so $c = y_1 - mx_1$.
- The Y-coordinate of an intersection with a vertical window boundary can be calculated as $y = mx + c$, where x is either X_{min} or X_{max}, depends on for which vertical line we are calculating Y.

$$Y = m \cdot X_{min} + c$$

(for intersection with left edge)

$$\text{Or} \quad Y = m \cdot X_{max} + c$$

(for intersection with right edge)

Similarly, X-coordinate of an intersection with a horizontal window boundary is computed as,

$$X = (y - c) / m$$

TechKnowledge Publications

Where y is either Y_{min} or Y_{max}, depends on for which horizontal line we are calculating X.

$$X = \frac{(Y_{min} - c)}{m} \quad \text{(for intersection with bottom edge)}$$

$$\text{Or} \quad X = \frac{(Y_{max} - c)}{m} \quad \text{(for intersection with top edge)}$$

This process is repeated until we find the entire line segment within the window.

Algorithm

Algorithm for Cohen-Sutherland line clipping approach is depicted here.

Step 1 : Read (X_{min}, Y_{min}) and (X_{max}, Y_{max}) – Lower-left and top-right corner of clipping window

Step 2 : Read A(x_1, y_1) and B(x_2, y_2) end points of line

Step 3 : Compute outcode of A and B

if $y_1 > Y_{max}$ **then** Outcode_A(1) = 1 **else** 0

if $y_1 < Y_{min}$ **then** Outcode_A(2) = 1 **else** 0

if $x_1 > X_{max}$ **then** Outcode_A(3) = 1 **else** 0

if $x_1 < X_{min}$ **then** Outcode_A(4) = 1 **else** 0

if $y_2 > Y_{max}$ **then** Outcode_B(1) = 1 **else** 0

if $y_2 < Y_{min}$ **then** Outcode_B(2) = 1 **else** 0

if $x_2 > X_{max}$ **then** Outcode_B(3) = 1 **else** 0

if $x_2 < X_{min}$ **then** Outcode_B(4) = 1 **else** 0

Step 4 : **if** Outcode_A **OR** Outcode_B = = 0000 **then**

Display entire line and goto step 5

else if Outcode_A **AND** Outcode_B ≠ 0000 **then**

Reject the entire line

else

Compute the intersection point with window boundaries

Repeat Step 4.

Step 5 : Stop

Advantages

- It is easy to understand.
- Simple to implement.
- Best suitable for the lines fully inside or outside.
- It can easily be extended for 3D line clipping.

Limitations

- Repeated clipping is expensive.
- Only applicable to rectangular clipping window. It cannot handle any other shape.
- It can be improved using more regions (e.g. Nichol Lee Nichol approach).

Tech Knowledge Publications

Examples

Ex. 4.7.1 : Use Cohen-Sutherland algorithm to clip two lines P_1 (40,15) – P_2 (75, 45) and P_3 (70, 20) – P_4 (100,10) against a window A(50,10), B (80,10), C (80,40) and D (50,40).

Soln. :

The spatial position of lines and rectangle is shown in Fig. P. 4.7.1.

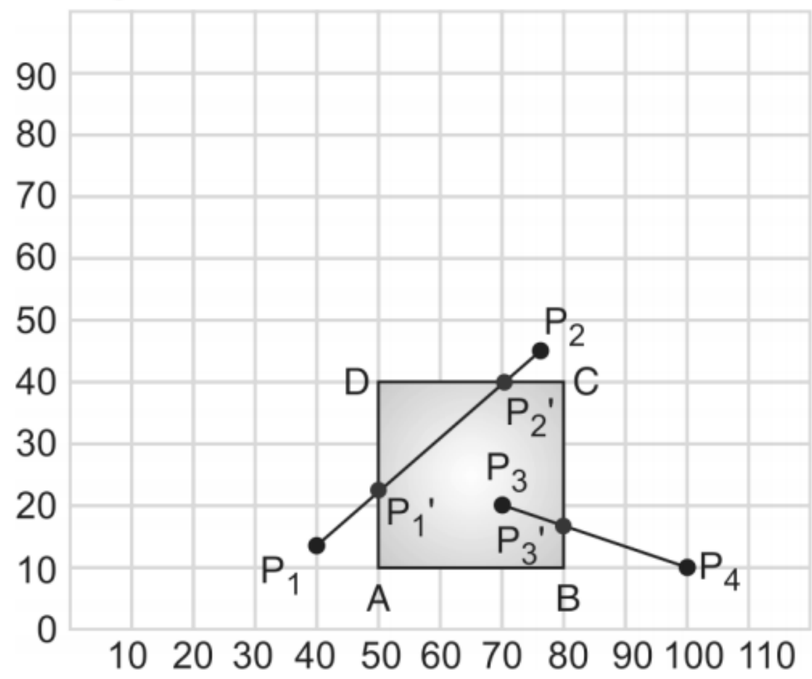


Fig. P. 4.7.1

Line P_1P_2

Given that, end points of line are P_1(40, 15) and P_2(75, 45)

$$\text{Outcode } (P_1) = 0001 = A_1$$

$$\text{Outcode } (P_2) = 1000 = B_1$$

$$A_1 \text{ OR } B_1 = 0001 \quad \text{OR} \quad 1000 = 1001$$

Logical OR of outcodes of both endpoints is not 0000, so the line is not completely visible.

$$A_1 \text{ AND } B_1 = 0001 \text{ AND } 1000 = 0000$$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint A is 0001, that means line intersects left edge while moving from P_1 to P_2. Let us call the intersection of point P_1'. P_1' is on line $x = x_{min} = 50$. So its x-coordinate is 50, we have to compute only y-coordinate of P_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{45 - 15}{75 - 40} = \frac{30}{35} = 0.857$$

$$\text{From, } y = mx + c, \; c = y - mx$$

By putting one of the points of line AB in this equation, we can find intercept of line with Y-axis. Let's put P_1 (40, 15) in above equation,

$$\therefore \quad c = 15 - (0.857) \times (40)$$

$$= 15 - 34.28 = -19.28$$

Put, $x = x_{min} = 50$ in explicit line equation to compute corresponding Y-coordinate,

$$y = mx + c = 0.857 \times (50) - 19.28$$

$$= 42.85 - 19.28 = 23.57$$

$$\text{Thus,} \quad P_1' = (50, 23.57)$$

Outcode of endpoint P_2 is 1000, that means line intersects top edge. Let us call the intersection of point P_2'. P_2' is on line y = 40. So its y-coordinate is 40, we have to compute only x-coordinate of B′.

$$\text{From,} \quad y = mx + c,$$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x-coordinate of P_2'.

$$\therefore \quad x = \frac{40 + 19.28}{0.857} = \frac{59.28}{0.857} = 69.17$$

$$\text{Thus,} \quad P_2' = (69.17, 40)$$

So, line segment with endpoints P_1'(50, 23.57) and P_2' (69.17, 40) is the visible segment.

Line P_3P_4

Given that, endpoints of the line are P_3(70, 20) and P_4(100, 10)

$$\text{Outcode } (P_3) = 0000 = C_1$$

$$\text{Outcode } (P_4) = 0010 = D_1$$

$$C_1 \text{ OR } D_1 = 0000 \quad \text{OR} \quad 0010 = 0010$$

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

$$C_1 \text{ AND } D_1 = 0000 \text{ AND } 0010 = 0000$$

Technknowledge Publication

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

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

$$= \frac{10 - 20}{100 - 70} = \frac{-10}{30} = -0.33$$

From, $y = mx + c, \; c = y - mx$

By putting one of the points of line P_3P_4 in this equation, we can find intercept of line with Y-axis. Let's put $P_3(70, 20)$ in above equation,

$$\therefore \quad c = 20 - (-0.33) \times (70)$$

$$= 20 + 23.1 = 43.1$$

Outcode of endpoint P_4 is 0010, that means line intersects right edge. Let us call the intersection of point P_3'.

From, $y = mx + c,$

By putting the value of x of a right edge in this equation, we can find the intersection y-coordinate of P_3'.

$$\therefore \quad y = (-0.33 \times 80) + 43.1$$

$$= -26.4 + 43.1 = 16.7$$

So, line segment with endpoints $P_3(70, 20)$ and $P_3'(80, 16.7)$ is the visible segment.

Original line Segments	Visible line segments
$P_1(40, 15)$ and $P_2(75, 45)$	$P_1'(50, 23.57)$ and $P_2'(69.17, 40)$
$P_3(70, 20)$ and $P_4(100, 10)$	$P_3'(70, 20)$ and $P_3'(80, 16.7)$

Ex. 4.7.2 : Let R be the rectangular window whose lower left-hand corner at L(–3, 1) and the upper right corner is at R(2, 6). Find the region codes for the endpoints and use the Cohen-Sutherland algorithm to clip the line segments. Coordinates for line segments are,

For line AB, A(– 4, 2) and B(– 1, 7)

For line CD, C(– 1, 5) and D(3, 8)

For line EF, E(– 2, 3) and F(1, 2)

Soln. :

The spatial position of lines and rectangle is shown in Fig. P. 4.7.2.

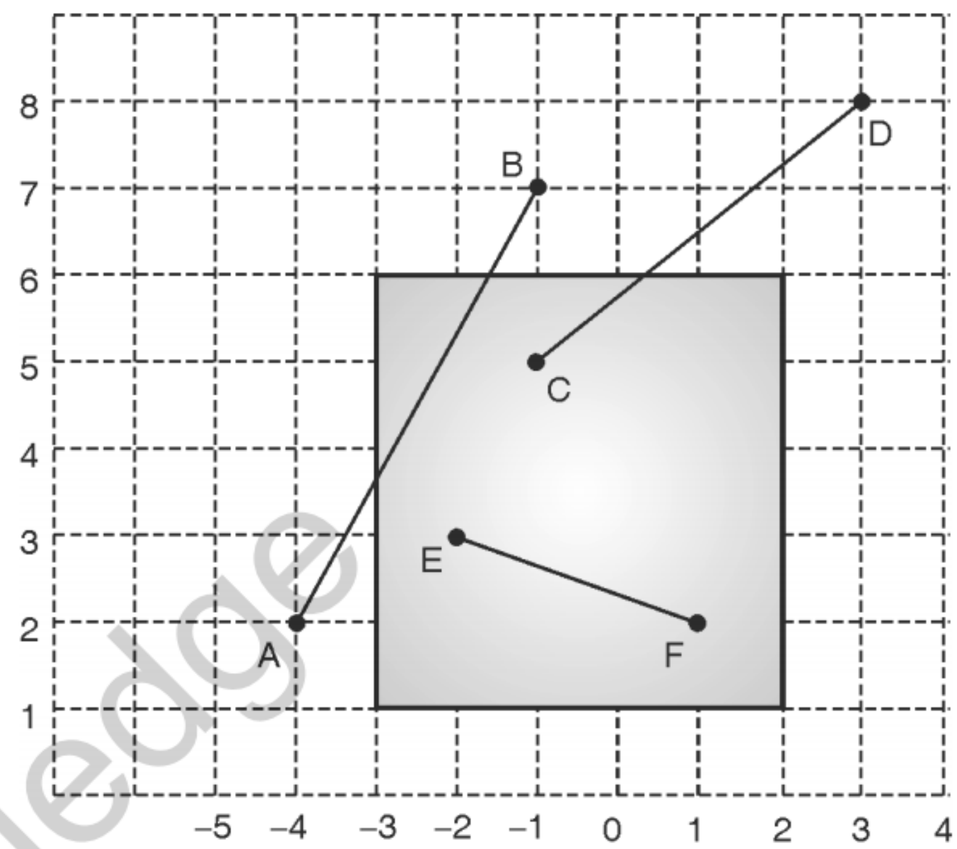


Fig. P. 4.7.2

Line AB

Given that, endpoints of the line are A (– 4, 2) and B (– 1, 7)

$$\text{Outcode (A)} = 0001 = A_1$$

$$\text{Outcode (B)} = 1000 = B_1$$

$$A_1 \text{ OR } B_1 = 0001 \quad \text{OR} \quad 1000 = 1001$$

Logical OR of outcodes of both endpoints is not 0000, so the line is not completely visible.

$$A_1 \text{ AND } B_1 = 0001 \text{ AND } 1000 = 0000$$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint A is 0001, that means line intersects left edge while moving from A to B. Let us call the intersection of point A with the left edge is A′. A′ is on line x = – 3. So its x coordinate is – 3, we have to compute only y-coordinate of A′.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7 - 2}{-1 + 4} = \frac{5}{3} = 1.67$$

From, $y = mx + c, \; c = y - mx$

TechKnowledge Publications

By putting one of the points of line AB in this equation, we can find intercept of line with y axis. Let's put A (– 4, 2) in above equation,

$$\therefore \quad c = 2 - (1.67) \times (-4) = 2 + 6.68 = 8.68$$

$$\text{From ,} \quad y = mx + c = 1.67 \times (-3) + 8.68$$

$$= -5.01 + 8.68 = 3.67$$

Outcode of endpoint B is 1000, that means line intersects top edge while moving from A to B. Let us call the intersection of point B with the top edge is B′. B′ is on line y = 6. So its y coordinate is 6, we have to compute only x-coordinate of B′.

$$\text{From,} \quad y = mx + c,$$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x-coordinate of B′.

$$\therefore \quad x = \frac{6 - 8.68}{1.67} = \frac{-2.68}{1.67} = -1.60$$

Outcode (A′) OR outcode (B′) = 0000 OR 0000 = 0000

So line segment with endpoints A′(– 3, 3.67) and B′ (– 1.6, 6) is the visible segment.

Line CD

Given that, end points of line are C (–1, 5) and D (3, 8)

$$\text{Outcode (C)} = 0000 = C_1$$

$$\text{Outcode (D)} = 1010 = D_1$$

$$C_1 \text{ OR } D_1 = 0000 \quad \text{OR} \quad 1010 = 1010$$

Logical OR of outcodes of both end point is not 0000, so line is not completely visible.

$$C_1 \text{ AND } D_1 = 0000 \text{ AND } 1010 = 0000$$

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint C is 0000, that means endpoint C is inside the clipping region.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{8 - 5}{3 + 1} = \frac{3}{4} = 0.75$$

$$\text{From,} \quad y = mx + c, \quad c = y - mx$$

By putting one of the points of line CD in this equation, we can find intercept of line with the y-axis. Let's put C (– 1, 5) in the above equation,

$$\therefore \quad c = 5 - (0.75) \times (-1) = 5 + 0.75 = 5.75$$

Outcode of endpoint D is 1010, that means line intersects top and right edges while moving from A to B. Let us call the intersection of point D with the top edge is D′. D′ is on line y = 6. So its y-coordinate is 6, we have to compute only x-coordinate of D′.

$$\text{From,} \quad y = mx + c,$$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x-coordinate of D′.

$$\therefore \quad x = \frac{6 - 5.75}{0.75} = \frac{0.25}{0.75} = 0.33$$

Outcode (C) OR outcode (D′) = 0000 OR 0000 = 0000

So line segment with endpoints C(– 1, 5) and D′(0.33, 6) is the visible segment.

Line EF

Given that, endpoints of the line are E (– 2, 3) and F (1, 2)

$$\text{Outcode (E)} = 0000 = E_1$$

$$\text{Outcode (F)} = 0000 = F_1$$

$$E_1 \text{ OR } F_1 = 0000 \quad \text{OR} \quad 0000 = 0000$$

Logical OR of outcodes of both endpoints is 0000, so the line is completely visible. It does not require any clipping

Techknowledge Publication

Original line Segments	Visible line segments
A(– 4, 2) and B(– 1, 7)	A′(– 3, 3.67) and B′ (– 1.6, 6)
C(– 1, 5) and D(3, 8)	C(– 1, 5) and D′ (0.33, 6)
E(– 2, 3) and F(1, 2)	E(– 2, 3) and F(1, 2)

Ex. 4.7.3 : Let ABCD be the rectangular window with A(20, 20), B(90, 20), C(90, 70) and D(20, 70). Find region codes for endpoints and use the Cohen-Sutherland algorithm to clip the lines :

(i) $P_1 P_2$ with $P_1(10, 30)$, $P_2(80, 90)$

(ii) $Q_1 Q_2$ with $Q_1(10, 10)$, $Q_2(70, 60)$

Soln. : The spatial position of lines and rectangle is shown in Fig. P. 4.7.3.

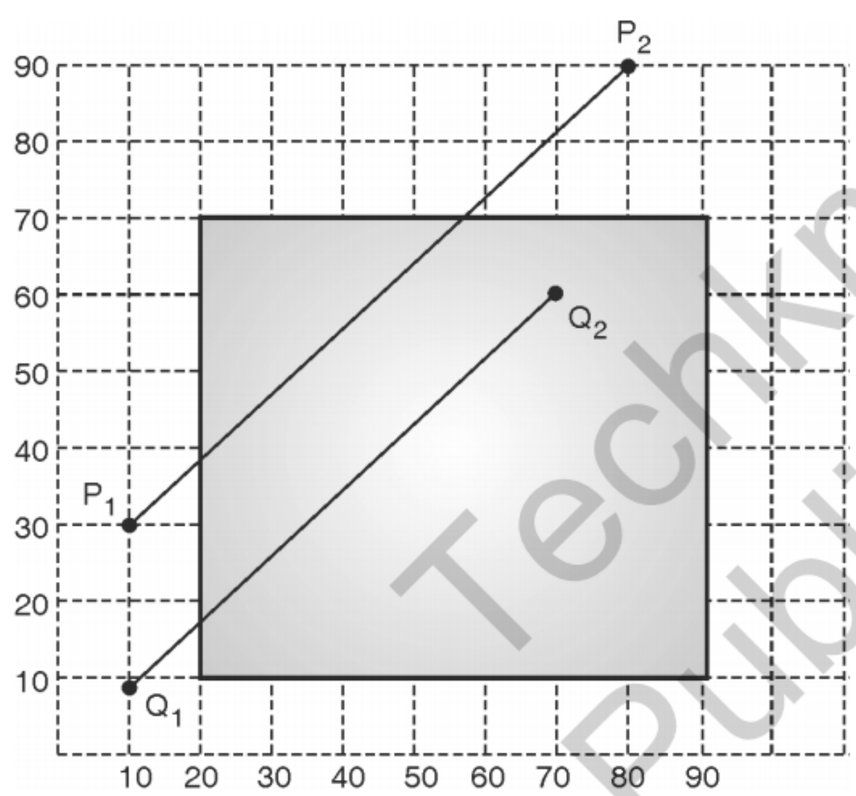


Fig. P. 4.7.3

Line P_1P_2

Given that, end points of line are P_1 (10, 30) and P_2 (80, 90)

Outcode (P_1) = 0001 = X

Outcode (P_2) = 1000 = Y

X OR Y = 0001 OR 1000 = 1001

Logical OR of outcodes of both endpoints is not 0000, so the line is not completely visible.

X AND Y = 0001 AND 1000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint, P_1 is 0001, that means line intersects left edge while moving from P_1 to P_2. Let us call the intersection of point P_1 with the left edge is P_1'. P_1' is on line x = 20. So its x-coordinate is 20, we have to compute the only y-coordinate of P_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{90 - 30}{80 - 10} = \frac{60}{70} = 0.86$$

From, $y = mx + c, \quad c = y - mx$

By putting one of the points of line P_1P_2 in this equation, we can find intercept of line with y-axis. Let's put P_1 (10, 30) in above equation,

$$\therefore \quad c = 30 - (0.86) \times (10) = 30 - 8.6 = 21.4$$

From, $y = mx + c = 0.86 \times (20) + 21.4$

$= 17.2 + 21.4 = 38.6$

Outcode of end point P_2 is 1000, that means line intersects top edge while moving from P_1 to P_2. Let us call the intersection of point P_2 with top edge is P_2'. P_2' is on line y = 70. So its y-coordinate is 70, we have to compute only x-coordinate of P_1'.

From, $y = mx + c,$

$$x = \frac{y - c}{m}$$

By putting the value of y of top edge in this equation, we can find the intersection x-coordinate of P_2'.

$$\therefore \quad x = \frac{70 - 21.4}{0.86} = \frac{48.6}{0.86} = 56.51$$

Outcode (A′) OR outcode (B′) = 0000 OR 0000 = 0000

So line segment with endpoints P_1' (20, 38.6) and P_2' (56.51, 70) is the visible segment.

Line Q_1Q_2

Given that, endpoints of the line are Q_1 (10, 10) and Q_2 (70, 60)

Outcode (Q_1) = 0101 = X

TechKnowledge Publications

Outcode (Q_2) = 0000 = Y

X OR Y = 0101 OR 0000 = 0101

Logical OR of outcodes of both endpoint is not 0000, so the line is not completely visible.

X AND Y = 0101 AND 0000 = 0000

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint Q_1 is 0101, that means line intersects bottom and left edges while moving from Q_1 to Q_2. Let us call the intersection of point Q_1 with the bottom edge is Q_1'. Q_1' is on line y = 20. So its y-coordinate is 20, we have to compute only x-coordinate of Q_1'.

$$\text{Slope of line} = m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{60 - 10}{70 - 10}$$

$$= \frac{50}{60} = 0.83$$

From, $y = mx + c, \quad c = y - mx$

By putting one of the points of line P_1P_2 in this equation, we can find intercept of line with y-axis. Let's put Q_1 (10, 10) in above equation,

$$\therefore \quad c = 10 - (0.83) \times (10) = 10 - 8.3 = 1.7$$

From, $y = mx + c,$

$$x = \frac{y - c}{m}$$

By putting value of y of top edge in this equation, we can find the intersection x-coordinate of Q_1'.

$$\therefore \quad x = \frac{20 - 1.7}{0.83} = \frac{18.3}{0.83} = 22.04$$

Outcode (Q_1') OR outcode (Q_2) = 0000 OR 0000 = 0000

So line segment with endpoints Q_1' (22.04, 20) and P_2' (70, 60) is the visible segment.

Original line Segments	Visible line segments
$P_1(10, 30)$ and $P_2(80, 90)$	P_1' (20, 38.6) and P_2' (56.51, 70)
$Q_1(10, 10)$ and $Q_2(70, 60)$	Q_1' (22.04, 20) and $Q_2(70, 60)$

Ex. 4.7.4 : Clip the line PQ having coordinates P(4,1) and Q(6,4) against the clip window having vertices A(3,2), B(7,2), C(7,6) and D(3,6) using Cohen-Sutherland line clipping algorithm.

Soln. :

Given that, endpoints of the line are P (4, 1) and Q (6, 4)

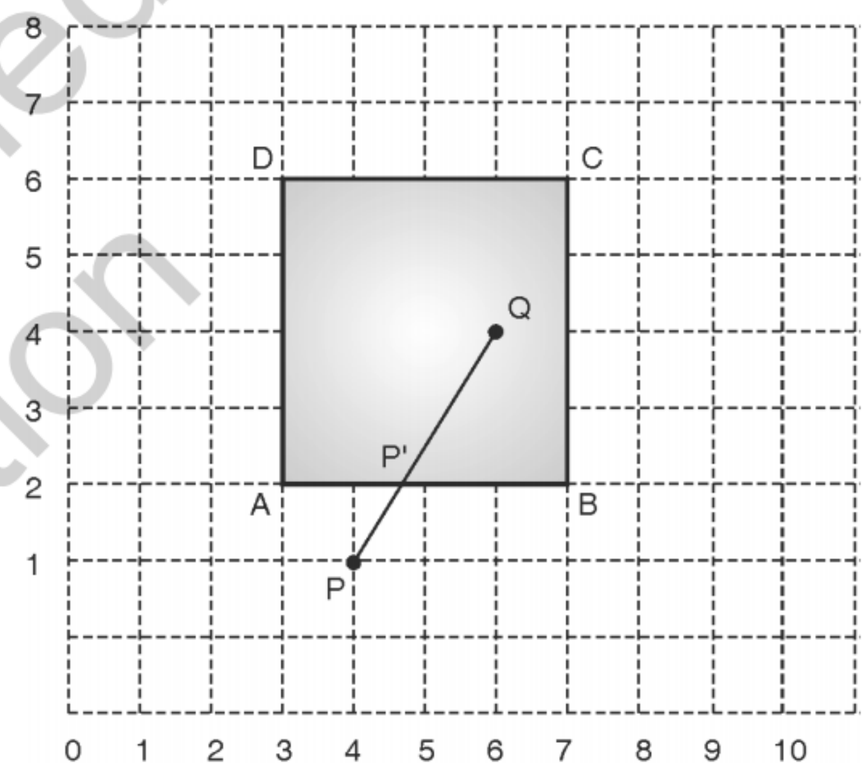


Fig. P. 4.7.4

The line is to be clipped against rectangle A (3, 2), B (7, 2), C (7, 6) and D (3, 6).

The spatial position of the line and rectangle is shown in Fig. P. 4.7.4.

Outcode (P) = 0100 = P_1

Outcode (Q) = 0000 = Q_1

P_1 OR Q_1 = 0100 OR 0000 = 0100

Logical OR of outcodes of both endpoints is not 0000, so the line is not completely visible.

P_1 AND Q_1 = 0100 AND 0000 = 0000

Technknowledge Publications

Logical AND of outcodes of both endpoint is 0000. So the line is not completely outside. Let us decide the visible portion of the line.

Outcode of endpoint P is 0100, that means line intersects bottom edge while moving from P to Q. Let us call the intersection point of P with the bottom edge is P′. P′ is on line y = 7. So its y-coordinate is 7, we have to compute only x-coordinate of P′.

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

$$= \frac{4-1}{6-4} = \frac{3}{2} = 1.5$$

From, $y = mx + c, \quad c = y - mx$

By putting one of the points of line PQ in this equation, we can find intercept of line with y axis. Let's put P (4, 1) in above equation,

$$\therefore \quad c = 1 - (1.5) \times (4)$$

$$= 1 - 6 = -5$$

From, $y = mx + c$

$$x = \frac{y - c}{m}$$

By putting the value of y of a bottom edge in this equation, we can find the intersection x-coordinate of P′.

$$\therefore x = \frac{2-(-5)}{1.5} = \frac{7}{1.5} = 4.67$$

∴ Coordinate of P′ would be (4.67, 2).

outcode (P′) OR outcode (Q) = 0000 OR 0000 = 0000

So line segment P′Q is the visible segment.

Original line Segments	Visible line segments
P(4, 1) and Q(6, 4)	P′(4.67, 2) and Q(6, 4)

4.8 Polygon Clipping

- Clipping of polygon requires more attention.
- The output of line clipping is either point or line.
- The output of polygon clipping may be point, line or polygon.
- Following issues arise in polygon clipping. All in all, polygon clipping is more complex compared to line clipping.
- In polygon clipping, each edge needs to be tested against each edge of the clipping region. This process may add new edges, may delete existing edges, may retain or divide the edges.

Issues in Polygon Clipping

1. Clipping a line segment yields at the most one-line segment.
2. Clipping of convex polygon yields at most one polygon (Fig. 4.8.1(a)).
3. Clipping of concave polygon can yield multiple polygons (Fig. 4.8.1(b)).
4. For same polygon, different number of edges may be generated depending on its orientation and spatial location (Fig. 4.8.1(c)).
5. May add new edges.
6. Edge may be discarded or divided.

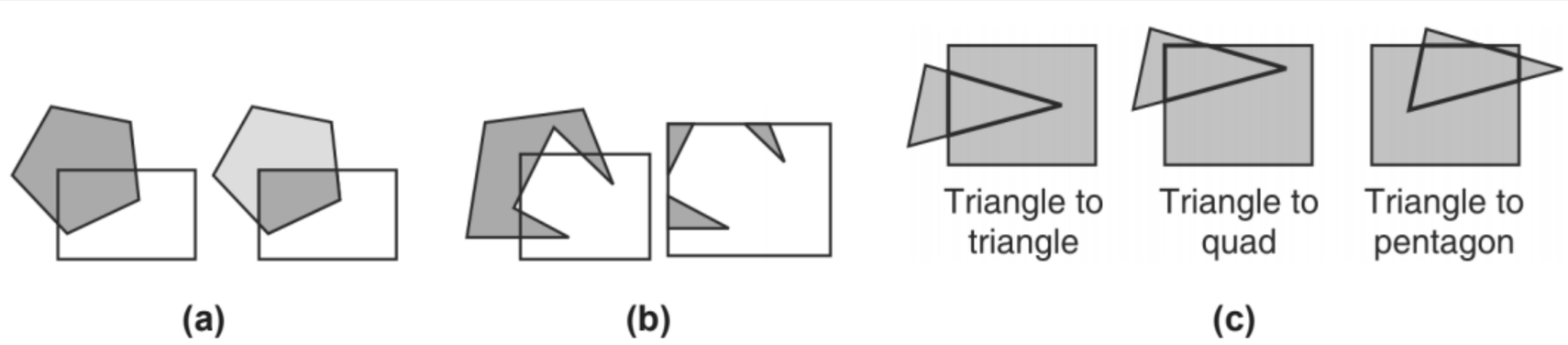


Fig. 4.8.1 : Cases of polygon clipping

Techknowledge Publication

4.8.1 Sutherland Hodgeman Polygon Clipping Algorithm

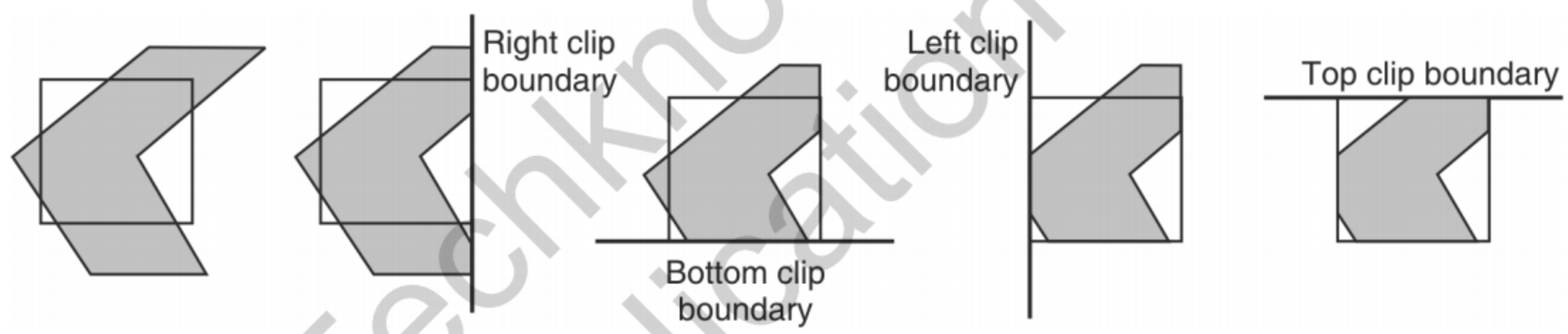
Q. Explain Sutherland-Hodgeman algorithm for polygon clipping. Compare it with Cohen-Sutherland Clipping. **SPPU - Dec. 17, 6 Marks**

Q. Write and explain with example Sutherland-Hodgeman polygon clipping algorithm.

SPPU - Dec. 18, 6 Marks

Working Mechanism

- Sutherland-Hodgman polygon clipping algorithm uses a divide-and-conquer strategy. Divide and conquer method divides the large problem into small subproblems, solve them recursively and combine the results to build the solution of parent problem.
- Sutherland-Hodgeman algorithm works in four stages. In each stage, each edge of subject polygon is tested against the edge of the clipping rectangle.
- To accomplish this task, algorithm process all vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we could first clip the polygon against any of the edge (i.e left, right, top or bottom) to produce a new sequence of vertices.
- This new set of vertices then successively passed to remaining boundary clippers. At each step, a new sequence of output vertices is generated and passed to the next clipper. Fig. 4.8.2 shows the simulation of the algorithm.



(a) Original polygon **(b) Output of right clipper** **(c) Output of bottom clipper** **(d) Output of left clipper** **(e) Output of right clipper**

Fig. 4.8.2 : Stages of polygon clipping

When we traverse along the polygon boundary, one of the following four cases occurs :

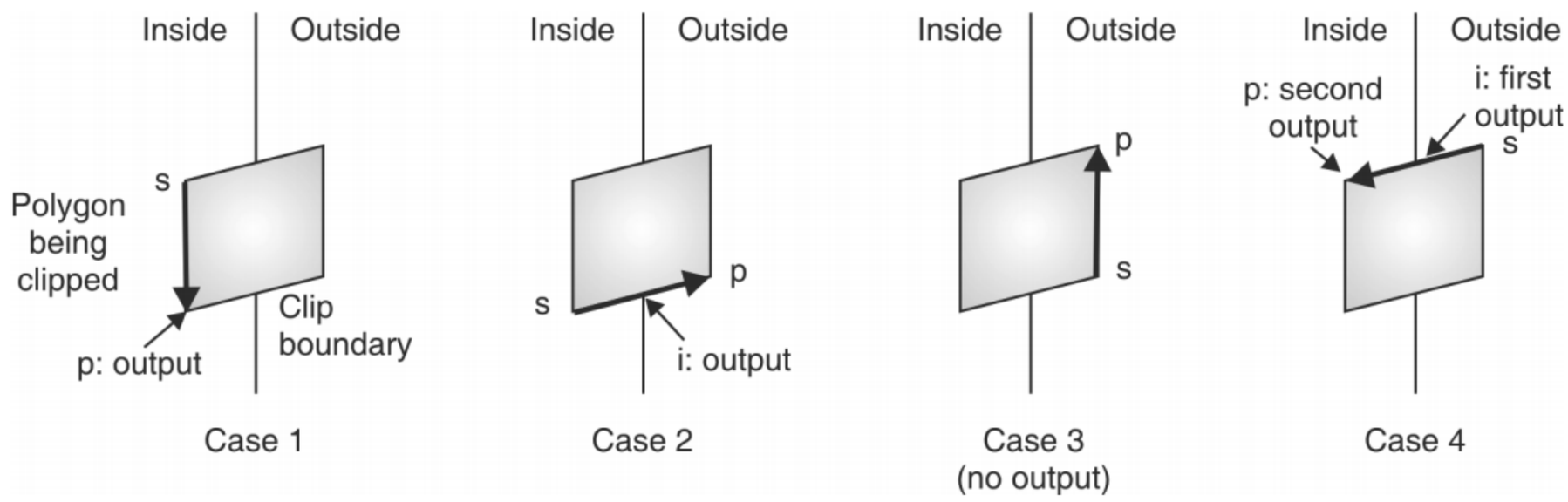


Fig. 4.8.3 : Output for various clippers

Case 1 :

While traversing along the boundary of the subject polygon, if the movement is from inside to inside i.e. when the source(s) and destination(p) vertices of the edge are inside the clipping region, we output destination vertex (p) only.

Case 2 :

When the movement is from inside to outside i.e. when source vertex is inside the clipping region and destination vertex is outside the clipping region, the polygon edge intersects the clipping boundary. In that case, we output the intersection point ' i ' only.

Case 3 :

When the movement is from outside to outside i.e when source vertex is outside the clipping region and destination vertex is also outside the clipping region, nothing is selected for output.

Case 4 :

When the movement is from outside to inside i.e. when source vertex is outside the clipping region and destination vertex is inside the clipping region, the polygon edge intersects the clipping boundary. In that case, we output both, the intersection point and the destination vertex.

Output list of the first clipper is passed to next clipper and processed in the same way.

Convex polygons are correctly clipped by this algorithm, but concave polygons sometimes may generate extraneous lines as shown in Fig. 4.8.4. Edge ab was not part of original polygon, but it has been now part of output polygon.

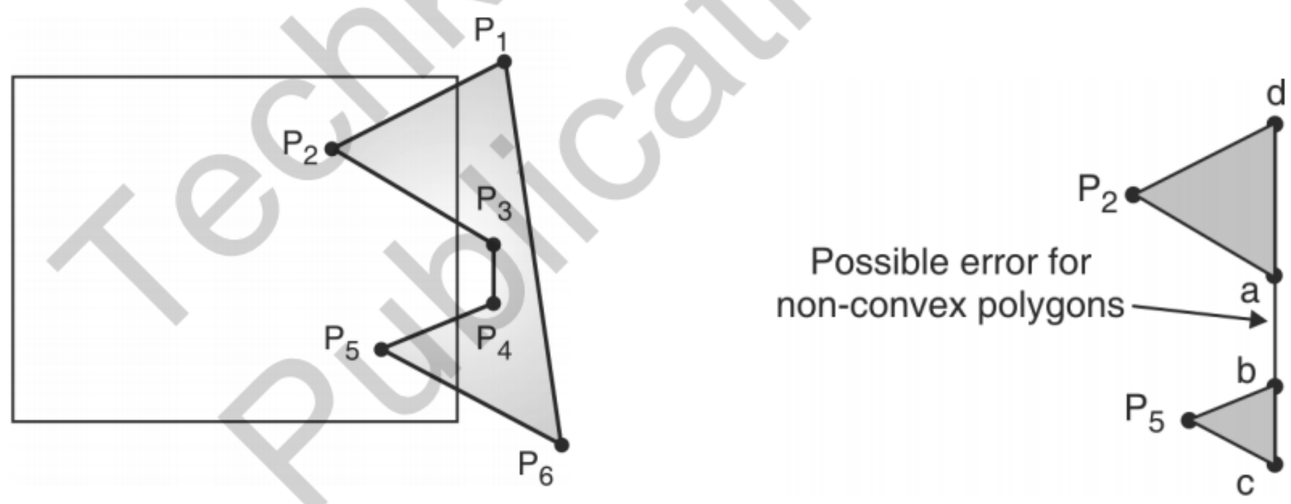


Fig. 4.8.4 : Issue with the Sutherland-Hodgeman algorithm

- This problem can be handled in many ways. One way is to convert the concave polygon to multiple convex polygons as shown in Fig. 4.8.5. The process of converting a polygon to a set of triangles is known as **tessellation**. Then process each convex polygon individually.

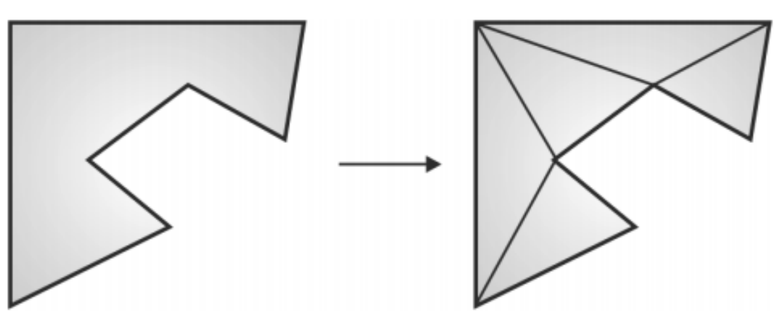


Fig. 4.8.5 : Tessellation process

- Another way of solving this problem is to tag the different new vertices as follow.
- Call α the new vertex generated on the transition from IN to OUT. Call β the new vertex generated on the transition from OUT to IN. Starting from OUT, connect α and β as soon α as is generated. This process successfully eliminates the extraneous edge.
- This algorithm is well suited for hardware implementation and can easily be parallelized on multiple processors, each processor works as a clipper. However, this method works only for convex clipping region.

Techknowledge Publications

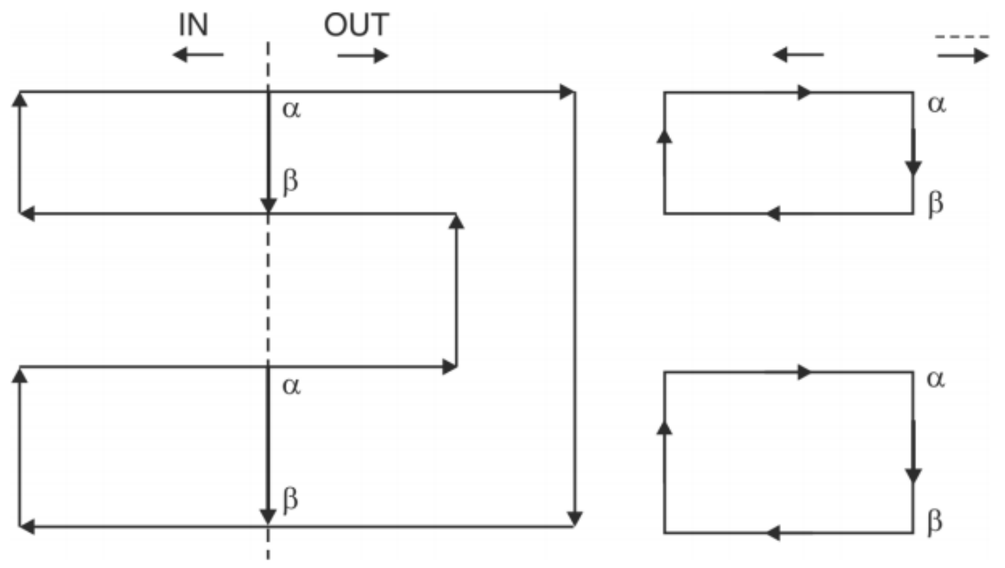


Fig. 4.8.6 : Solving the issue of Sutherland-Hodgeman algorithm

Algorithm

Step 1 : Read coordinates of vertices of the Polygon.

Step 2 : Read coordinates of the clipping window.

Step 3 : Consider the left edge of the window.

Step 4 : Compare the vertices of each edge of the polygon, individually with the clipping plane.

Step 5 : Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.

Step 6 : Repeat the steps 4 and 5 for remaining edges or the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.

Step 7 : Stop.

Advantages

- It is easy to implement.
- It clips polygon against all edges of the clipping polygon.
- Well suited for hardware implementation.

Disadvantages

- Only works for convex clipping polyon.
- It clips to each window boundary one at a time.
- It has redundant edge-line cross computation.
- Requires considerable amount of memory to store all polygons in original form.
- Sometime produces single polygon instead of multiple polygons for concave polygon.

Examples

Ex. 4.8.1 : Clip the subject polygon (V_1 - V_2 - V_3 - V_4 - V_5 - V_1) against the given clipping polygon.

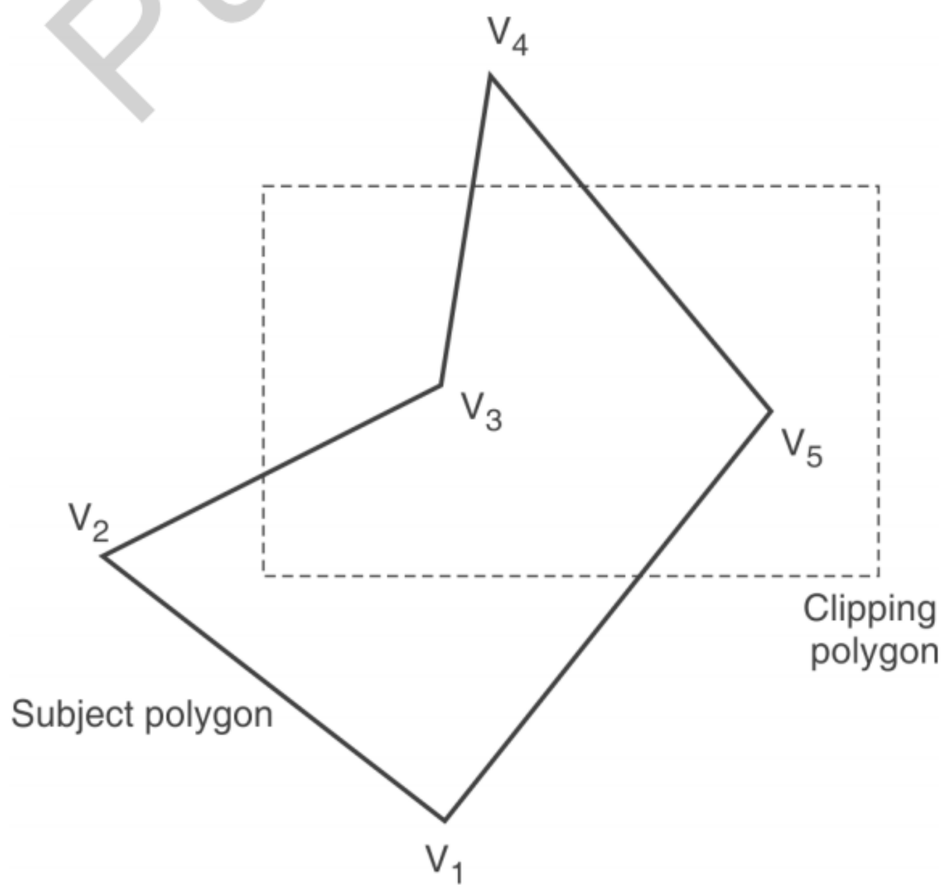


Fig. P .4.8.1

Technknowledge Publication

Soln. :

Intersection of the different clippers with subject polygon are shown in Fig. P. 4.8.1(a) :

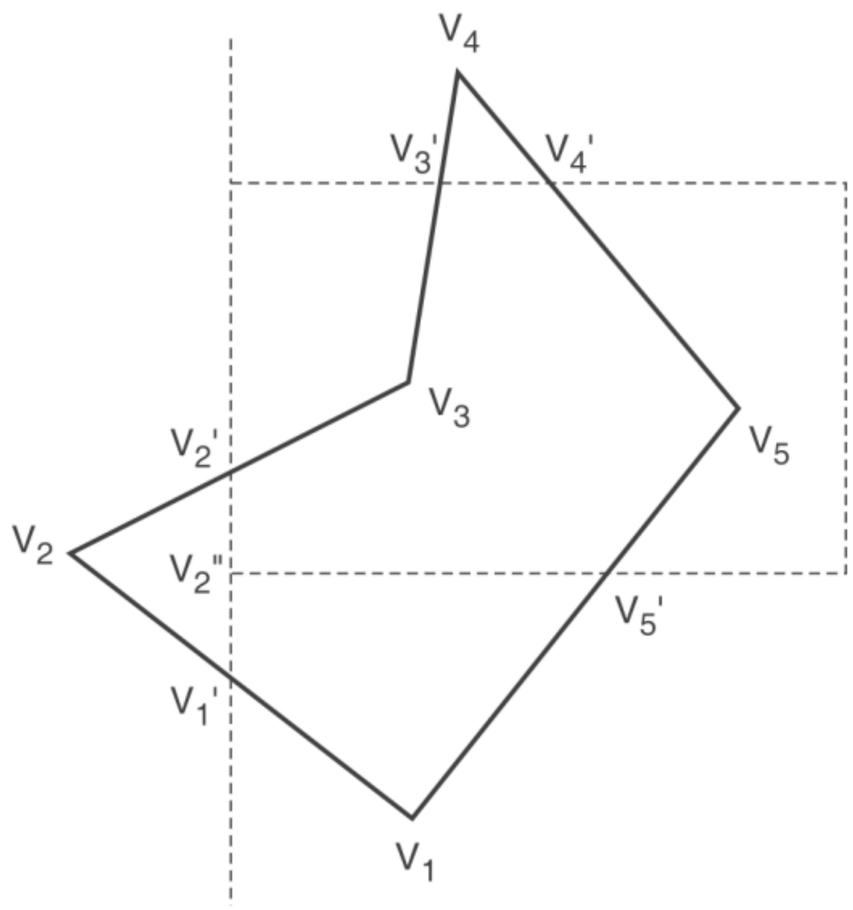


Fig. P .4.8.1(a)

Input to the algorithm : $\{V_1, V_2, V_3, V_4, V_5\}$

Output of left clipper : $\{V_1, V_1', V_2', V_3, V_4, V_5\}$

Output of right clipper : $\{V_1, V_1', V_2', V_3, V_4, V_5\}$

Output of top clipper : $\{V_1, V_1', V_2', V_3, V_3', V_4', V_5\}$

Output of bottom clipper : $\{V_2'', V_2', V_3, V_3', V_4', V_5, V_5'\}$

The final clipped polygon would be $\{V_2'', V_2', V_3, V_3', V_4', V_5, V_5'\}$.

4.8.2 Weiler Atherton Polygon Clipping Algorithm

Q. Explain the term with example - Generalized Clipping. **SPPU - Dec. 17, 2 Marks**

Working Mechanism

- The Weiler-Atherton clipping algorithm is a general-purpose 2D clipping algorithm. It is capable of clipping a concave polygon with interior holes to the boundaries of another concave polygon, also with interior holes. The polygon to be clipped is called the Subject Polygon (SP) and the clipping region is called the Clip Polygon (CP).
- SP and the CP are defined by a circular list of vertices. Exterior boundaries are described in clockwise, and the interior boundaries or holes are described in anti-clockwise. The boundaries of the SP and the CP may or may not intersect.
- If they intersect, the intersections occur in pairs. One of the intersections occurs when the SP edge enters inside of the CP and one when it leaves.
- Proceeding in clockwise around the original polygon vertex list, perform the following steps :
  - If a line segment enters the clip region, add the intersection to the output list and follow the polygon boundary.
  - If a linesegment leaves the clip region, add the intersection to the output list, follow the boundary of the clip region, clockwise.
- This could be achieved more fundamentally in the following way :

1. Creating intersection vertex lists

- Entering list contains intersections for SP edge entering inside of the CP.
- Leaving list contains intersections for the SP edge leaving the inside of the CP.

2. Perform actual clipping

- From entering a list, remove an intersection vertex and check if the list is empty. If it is so, the clipping process is done.

Technknowledge Publications

- Traverse the SP vertex list until next intersection point is found. Store the SP list upto this point to the inside holding list.
- Jump to the CP vertex list at the same intersection point location.
- Traverse the CP vertex list until the next intersection point is found. Store the CP vertex list up to this point to the inside holding list.
- Jump back to SP vertex list at the same intersection point location.
- Repeat the process until the loop is formed.
- Polygons outside the CP are found using the same procedure, except that the initial intersection vertex is obtained from the leaving list and the CP vertex list is followed in the reverse direction. The polygon lists are copied to the outside holding list.

Advantages

- This algorithm can also handle concave polygons easily.
- It does not require splitting the concave polygon to convert it into a convex polygon.
- Unlike the Sutherland-Hodgeman algorithm, it does not produce spurious edge, so no post-processing is required.

Disadvantages

Not easy to parallelize.

Examples

Ex. 4.8.2 : Consider the following pair of clipping and subject polygon. Find the visible region of the subject polygon.

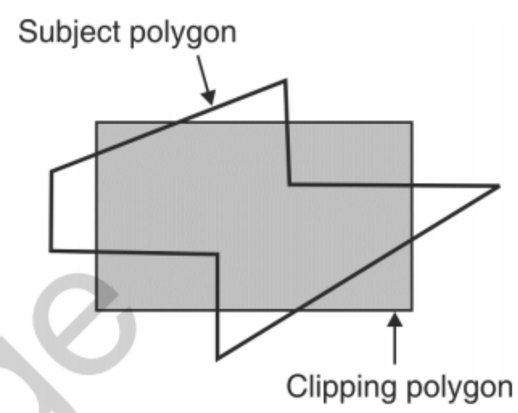


Fig. P. 4.8.2

Soln. :

Let us first label the vertices of the subject polygon as s_i and clipping polygon as c_i. Intersection points of both polygons are labelled as i_i.

As per algorithm, start visiting the polygon from any of the entering intersection points. List of entering intersection points : { i_1, i_3, i_5, i_7 }.

Let us start from i_1 and keep traversing along window boundary as mentioned in the above procedure.

Fig. P. 4.8.2(b) shows the procedure.

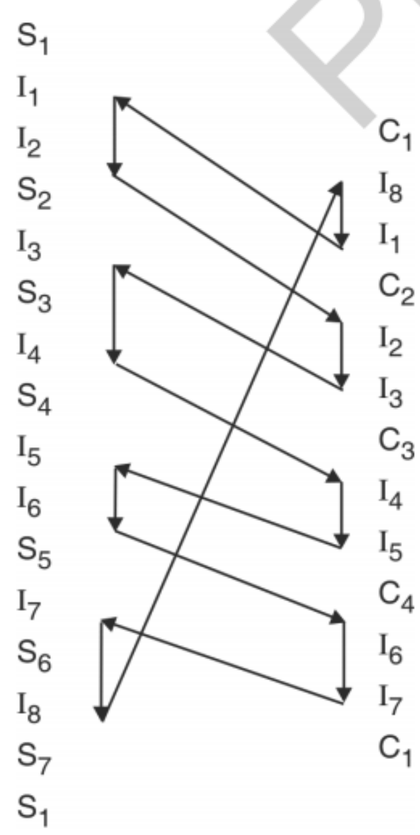


Fig. P. 4.8.2 (a)

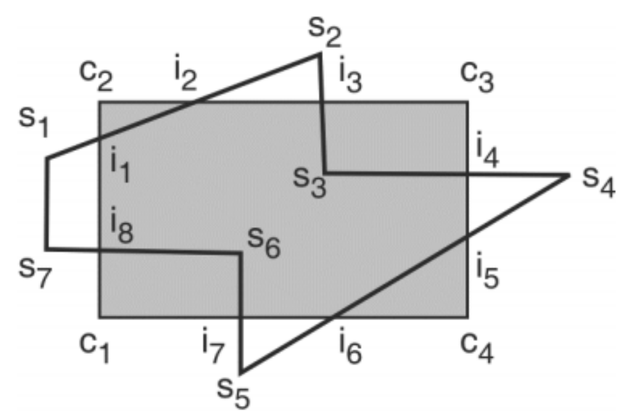


Fig. P. 4.8.2 (b)

Collect all the visited vertices in the same order. That would be the visible section of the subject polygon. So visible region of S is : { i_1, i_2, i_3, s_3, i_4, i_5, i_6, i_7, s_6, i_8, i_1 }

Ex. 4.8.3 : Consider the following pair of clipping and subject polygon. Find the visible region of the subject polygon.

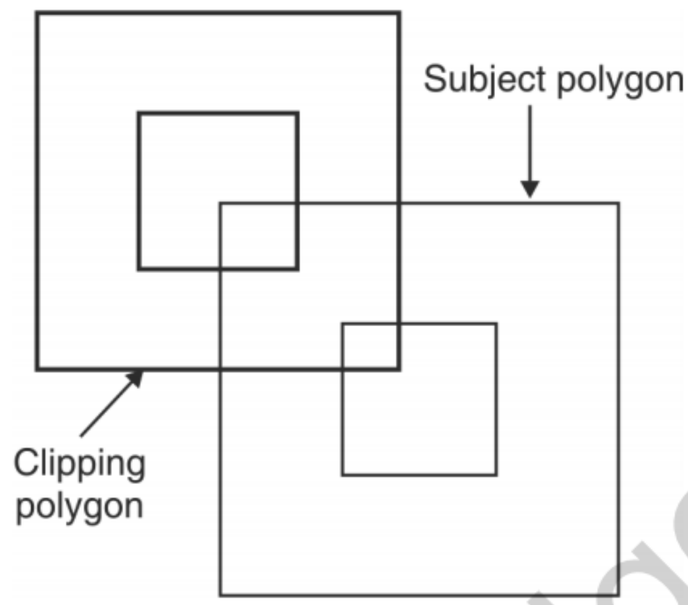


Fig. P. 4.8.3

Soln. :

Let us first label the vertices of the subject polygon as s_i and clipping polygon as c_i . Intersection points of both polygons are labelled as i_i .

- As per algorithm, start visiting the polygon from any of the entering intersection points. List of entering intersection points : { i_1, i_3, i_5 }.
- Let us start from i_1 and keep traversing along the window boundary as mentioned in the above procedure. Fig. P. 4.8.3(a) shows the procedure.
- Collect all the visited vertices in the same order. That would be the visible section of the subject polygon. So visible region of S is : { i_1, i_2, c_6, i_3, i_4, i_5, s_8, i_6, i_1 }

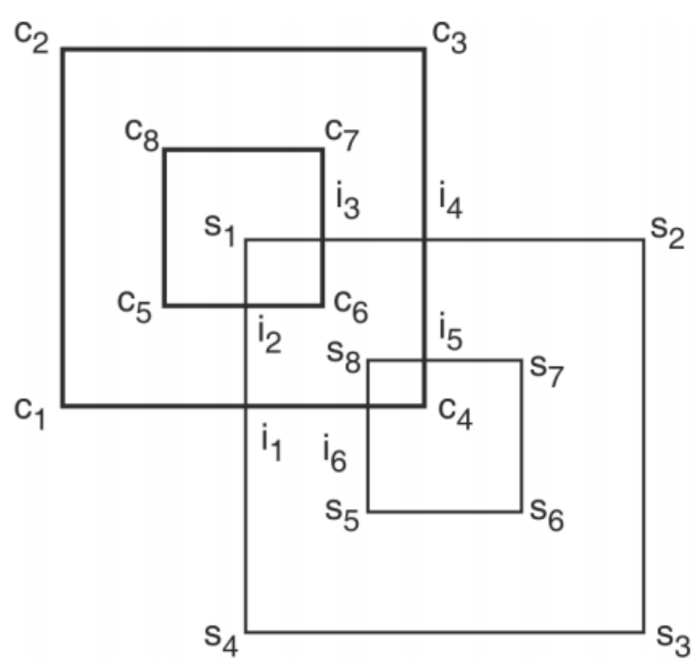


Fig. P. 4.8.3 (a)

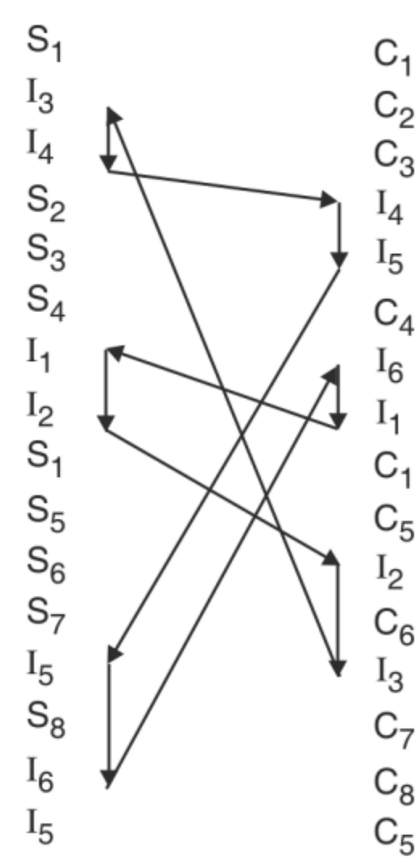


Fig. P. 4.8.3 (b)

Review Questions

Q. 1 List types of Polygon.

Q. 2 What is a polygon? What are the different types of polygons ?

Q. 3 Explain two methods for testing whether the point is inside the polygon or not.

Q. 4 Explain the different methods for testing a pixel inside a polygon.

Q. 5 Describe the Even-Odd test method.

Q. 6 Explain the winding number method.

Q. 7 List various polygon filling algorithms.

Q. 8 Explain the boundary fill algorithm with pseudo-code. Also mention its limitations, if any.

Q. 9 Write an algorithm for boundary fill.

Q. 10 State merits and demerits of boundary fill algorithm.

Q. 11 Write algorithm to fill the polygon area using flood fill method.

Q. 12 Write a short note on polygon filling with patterns.

Q. 13 Explain the concept of windowing and clipping.

Q. 14 Explain various approaches for clipping the primitives.

Q. 15 Write a short note on viewing pipeline.

Q. 16 Define the terms - Window, Viewport, Window-to-viewport transformation.

Q. 17 Discuss the viewing pipeline in computer graphics with a neat diagram.

Q. 18 Discuss the effect of different size and position of window and viewport in viewing.

Q. 19 Derive transformation matrix for window-to-viewport transformation.

Q. 20 What is the point clipping? How to achieve it?

Q. 21 What is line clipping? Discuss various cases of clipping with suitable diagram.

Q. 22 Write and explain with an example Cohen-Sutherland line clipping algorithm.

Q. 23 How region codes are computed in the Cohen-Sutherland Algorithm?

Q. 24 State the advantages and disadvantages of the Cohen-Sutherland Algorithm.

Q. 25 State the issues in polygon clipping.

Q. 26 Explain Sutherland-Hodgeman polygon clipping algorithm.

Q. 27 What are the issues involved in Sutherland-Hodgeman polygon clipping algorithm? How to handle them.

Technknowledge Publication

❑❑❑

CHAPTER 5

UNIT III

2D Transformations

Syllabus

2-D transformations : Introduction, homogeneous coordinates, 2-D transformations - Translation, scaling, rotation and shear, rotation about an arbitrary point.

5.1 Introduction to 2D Transformation

- In the real world, the user frequently rearranges objects. Picture or scene in computer is also made up of objects. A graphics system should allow users to change the way by which the object appears.
- The process of changing the size, shape or position of the object is called **transformation**.

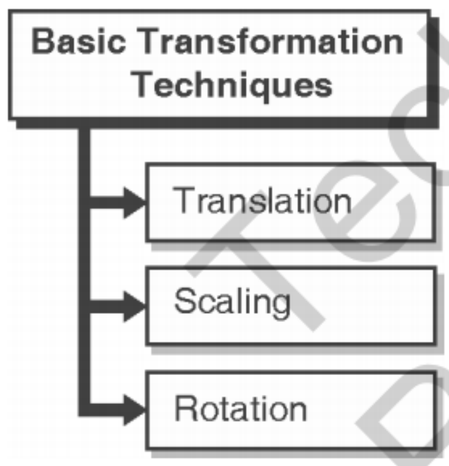


Fig. 5.1.1 : Basic transformation techniques

- Transformation is a very common operation used in graphics applications. Translation, rotation and scaling are the basic transformation operations. Reflection and shearing are other common transformation operations.
- Translation, rotation and reflection are also known as **rigid body transformations** because they do not alter the shape of the object. They just reposition the object from one location to another location.
- Scaling and shearing deforms the shape of the object under consideration, so they are not rigid-body transformation.
- The main objective of two-dimensional transformations is to simulate movement and manipulate 2D objects in the XY plane.

Terminology

The following terminology will be used for the rest of the discussion.

P = Set of points of the object which has to be transformed.

P′ = Set of points of the objectwhich has been transformed.

M = Transformation matrix.

Transformation is a function of object coordinates with various operations as another argument.

$$\therefore \quad P' = f(P, M)$$

Where, M is a transformation matrix to be applied on a set of points P. More common representation of transformation operation is $P' = M \cdot P$.

Translation, rotation and reflection are also known as ***rigid body transformations*** because they do not alter the shape of the object. They just reposition the object from one location to another location. Scaling and shearing deform the shape of the object under consideration, so they are not rigid-body transformation.

Assumption : Matrices are represented in column major form. For row major representation, $P' = P \cdot M$

Technknowledge Publication

5.2 Revising Matrices

Transformation and matrix operations go hand in hand. Let us first discuss various representation, properties and operations on matrix.

5.2.1 Matrix Representation

The matrix can be represented in two ways.

Column Measure Representation

In this representation, point coordinates are represented in the column.

For example,

$$A = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Let $M = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$ be some transformation matrix.

The coordinates of transformed coordinates are given by,

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & c \\ b & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\therefore \; x' = ax + cy$$

$$y' = bx + dy$$

In column major representation, M is *pre-multiplied* to original points of an object.

Row Measure Representation

– In this representation, point coordinates are represented in row. For example,

$$B = A^T = [x \;\; y] = [2 \;\; 3]$$

$$P'^T = P^T \cdot M^T$$

$$[x' \;\; y'] = [x \;\; y] \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$\therefore \; x' = ax + cy$$

$$y' = bx + dy$$

– In row-major representation, M is *post-multiplied* to original points of an object.

– We can multiply two matrices A and B only if dimensions of A are $p \times q$ and dimensions of B are $q \times r$. For matrix multiplication, it is necessary to have a number of columns in the first matrix equivalent to the number of rows in the second matrix.

$$A_{p \times q} \times B_{q \times r} = C_{p \times r}$$

– If number of columns in A and number of rows in B are not same, then matrix dimensions are not compatible and multiplication is not possible.

5.2.2 Matrix Properties

1. **Associative :** Matrix multiplication is associative. i.e.,

$$A \cdot B \cdot C = A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

2. **Commutative :** Matrix multiplication is not commutative. i.e.,

$$A \cdot B \neq B \cdot A$$

5.2.3 Determinant of Matrix

Determinant of 2×2 matrix is calculated as below :

$$\text{Let } A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$\text{Determinant } \Delta(A) = |A| = a_{11}\, a_{22} - a_{12}\, a_{21}$$

Determinant of 3×3 matrix is calculated as below :

$$\text{Let } B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$\text{Determinant } \Delta(B) = |B| = b_{11}\,(b_{22}b_{33} - b_{23}b_{32}) - b_{12}(b_{21}b_{33} - b_{23}b_{31}) + b_{13}\,(b_{21}b_{32} - b_{22}b_{31})$$

Techknowledge Publication

5.2.4 Multiplying Two Matrices

Let A and B are 2 × 2 square matrices to be multiplied and C is the resultant matrix.

$$\text{If, } A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$\text{And } B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = A \times B$$

$$\therefore \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\text{Where, } C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

For some random dimensional compatible, rectangular matrices A and B,

$$C = A \times B$$

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

Where,

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} + A_{13} \cdot B_{31}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} + A_{13} \cdot B_{32}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} + A_{23} \cdot B_{31}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} + A_{23} \cdot B_{32}$$

5.3 Translation

Q. Write 2D transformation matrices of translation.

SPPU - May 15, 2 Marks

- The **translation** is defined as a shifting of an object along a straight path.
- Translation does not alter the shape or size of the object. It just moves the entire object from one location to another location along a straight path.
- This is achieved by first shifting object in the X-direction and then in the Y-direction by amount t_x and t_y, respectively.
- The vector $T = [t_x \quad t_y]$, specifying the amount of shift in both the directions is called **shift-vector** or **translation vector**.
- Translation operation is formulated as,

$$x' = x + t_x$$

$$y' = y + t_y$$

Where (x, y) is the original point, $[t_x, t_y]$ is shift vector and (x', y') is translated point.

Equivalent matrix representation of this operation is,

$$P' = T + P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix}$$

- If an object has more than one vertex,we can apply a transformation to individual vertex and then join them all to reconstruct the object.
- Rigid body transformation moves object without deformation. Amount of shift is identical for all the points.
- In Fig. 5.3.1, object O is translated by shift-vector [6, 5]. O′ is the translated object.

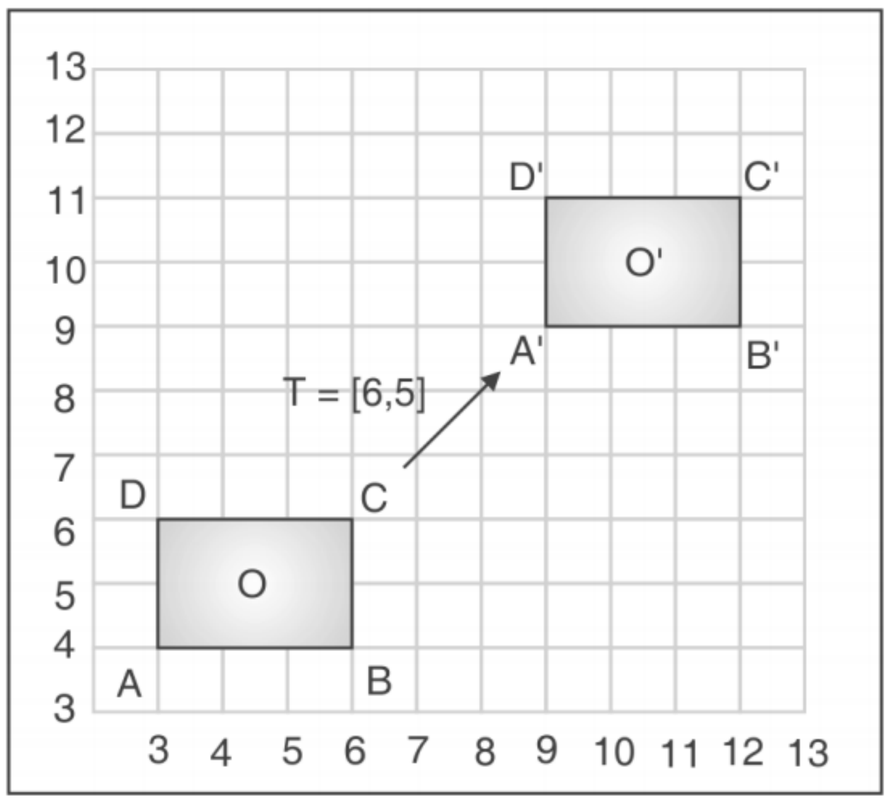


Fig. 5.3.1 : Translation operation

Ex. 5.3.1 : Translate a triangle with coordinates A(2, 2), B(6, 2) and C(4, 4) with translation vector $[4\ \ 7]^T$.

Soln. :

Here translation vector,

$$T = \begin{bmatrix} 4 \\ 7 \end{bmatrix}$$

Translation operation is defined as,

$$P' = T + P$$

$$\therefore A' = T + A = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 9 \end{bmatrix}$$

$$B' = T + B = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 9 \end{bmatrix}$$

$$C' = T + C = \begin{bmatrix} 4 \\ 7 \end{bmatrix} + \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 8 \\ 11 \end{bmatrix}$$

After translation, triangle A(2, 2), B(6, 2), C(4, 4) will move to A′ (6, 9), B′(10, 9), C′(8, 11).

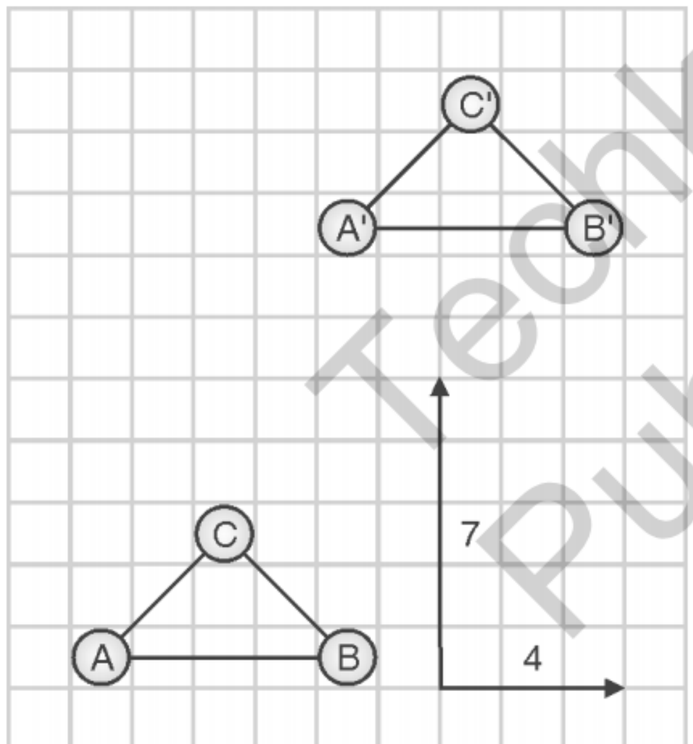


Fig. P. 5.3.1 : Translation of triangle ABC by shift-vector $[4\ \ 7]^T$

Original Coordinates	Transformed Coordinates
A(2, 2)	A'(6, 9)
B(6, 2)	B'(10, 9)
C(4, 4)	C'(8, 11)

Ex. 5.3.2 : Explain the translation of a line in detail.

Soln. :

- The translation is a rigid-body transformation.
- After translation operation, the shape of the object does not change. That gives the intuition to translate only endpoints of line and again joining them.
- We do not need to translate individual pixel on line. Let us consider the line with endpoints A (x_1, y_1) and B (x_2, y_2). Let's assume the translation vector is T $[t_x\ t_y]^T$.
- Translation of line requires translation of only and endpoints of the line. Translated endpoints are further passed to line generation algorithm to generate remaining pixels on the straight path joining them.

If (x_1', y_1') and (x_2', y_2') are the translated coordinates, then

$$x_1' = t_x + x_1$$

$$y_1' = t_y + y_1$$

$$x_2' = t_x + x_2$$

$$y_2' = t_y + y_2$$

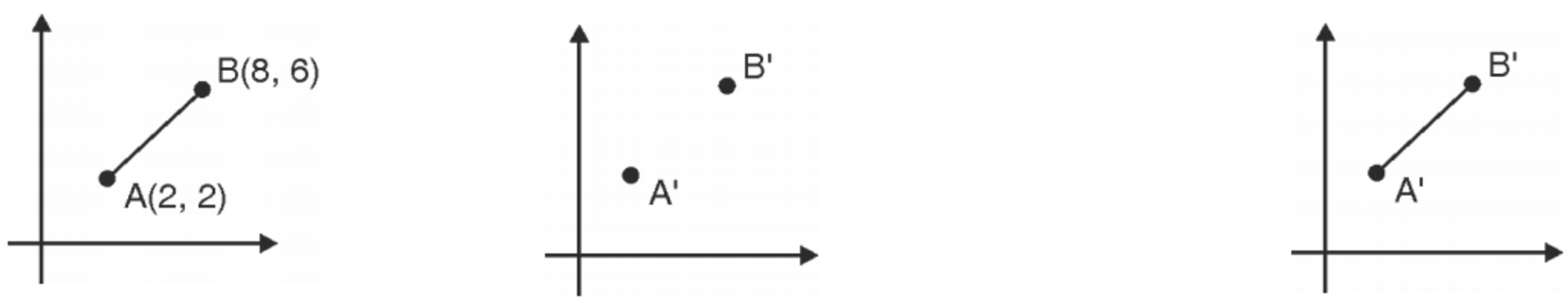
In matrix form,

$$P' = T \cdot P$$

Where, P′ and P are set of translated and original points respectively. Homogeneous representation (Discussed in section 5.6) of transformation sequence would be,

$$\therefore P' = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \\ y_1 & y_2 \\ 1 & 1 \end{bmatrix}$$

Technknowledge Publications



(a) Original line **(b) Translated endpoints** **(c) Points on the line after joining endpoint A' B'**

Fig. P. 5.3.2 : Translation of line

5.4 Scaling

Q. Write 2D transformation matrices of scaling.

SPPU - May 15, 2 Marks

- Translation and rotation change the position of an object; they do not deform the shape. Scaling may alter the shape of an object.
- Scaling can be performed with respect to origin or some reference point.

5.4.1 Scaling About Origin

Scaling is performed by multiplying the vertex coordinates by scaling parameters S_x and S_y respectively. So,

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

We can represent the scaling operation in matrix form as,

$$P' = S \cdot P$$

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

S_x, S_y are any positive numeric values.

Note : For 2 x 2 matrix representation, translation is an additive operation, whereas scaling and rotation are multiplicative.

When object is scaled up with respect to origin, it goes away from the origin. Converse is true when object is scaled down. Scaling may be in one direction or in both directions, or even it can be uniform ($S_x = S_y$) or non-uniform ($S_x \neq S_y$)

Various Cases of Scaling

$(S_x = S_y) = 1$	$\Rightarrow$	No scaling
$(S_x = S_y) \neq 1$	$\Rightarrow$	Uniform scaling
$S_x \neq S_y$	$\Rightarrow$	Non-uniform scaling
$(S_x = S_y) < 1$	$\Rightarrow$	Uniform compression
$(S_x = S_y) > 1$	$\Rightarrow$	Uniform expansion
$S_x > 1$	$\Rightarrow$	X-coordinates of the object will move away from the origin
$S_x < 1$	$\Rightarrow$	X-coordinates of the object will move near to the origin
$S_y > 1$	$\Rightarrow$	Y-coordinates of the object will move away from the origin
$S_y < 1$	$\Rightarrow$	Y-coordinates of the object will move near to the origin

As shown in Fig. 5.4.1, line AB has coordinates A(1, 1) and B(1, 4). With $S_x = 3$, $S_y = 2$, the line goes away from the origin and we get A'(3, 2) and B'(3, 8).

$$A' = S \cdot A = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$B' = S \cdot B = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Techknowledge Publications

Similarly, if we select scaling factor less than one, the object comes closer to the origin. With $S_x = S_y = 0.5$, line CD with endpoints C(10, 4) and D(10, 8) comes at C'(5, 2) and D'(5, 4).

$$C' = S \cdot C = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 10 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$$

$$D' = S \cdot D = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

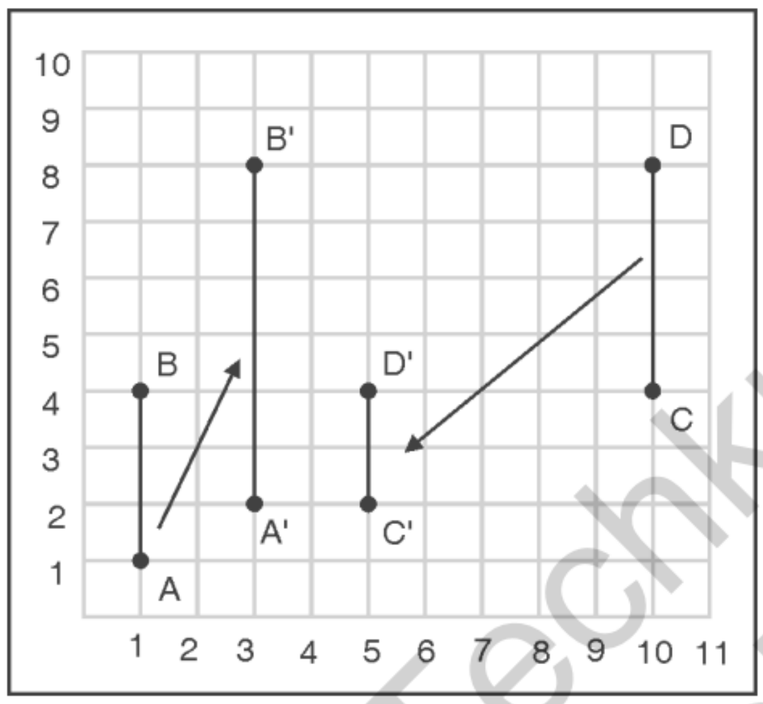


Fig. 5.4.1 : Demonstration of scaling

Note : These scaling formulas are only applicable if scaling is with respect to the origin.

5.4.2 Scaling with Respect to Reference Point

- In this section, we will derive the transformation matrix to scale the object with respect to some reference point.
- Let (x_r, y_r) be the reference point, with respect to which object is to be scaled.

Following steps should be performed to carry out the desired operation.

1. Translate the reference point to the origin.

$$T = \begin{bmatrix} -x_r \\ -y_r \end{bmatrix}$$

2. Apply scaling on the translated object.

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

3. Translate reference point back to its actual location.

$$T^{-1} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

The sequence of operations is shown in Fig. 5.4.2.

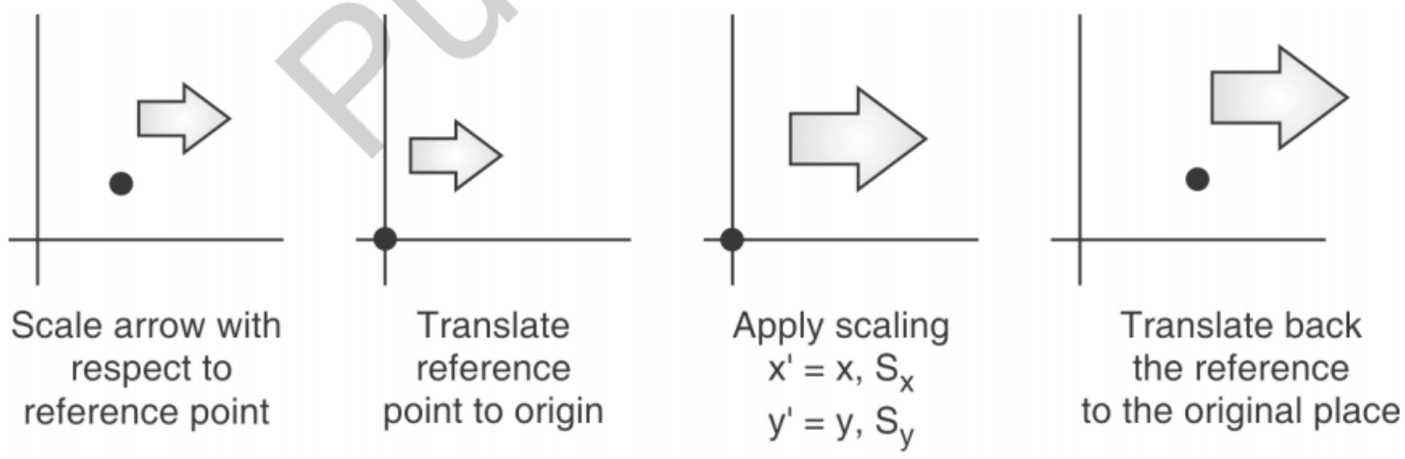


Fig. 5.4.2 : Sequence of transformation operations for scaling with respect to a reference point

Recall that the transformation sequence is written from right to left for column major representation. So, the transformed coordinates of an object are given by,

$$P' = [T^{-1} + [S[T + P]]]$$

$$P' = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \right\} \cdot \left\{ \begin{bmatrix} -x_r \\ -y_r \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \right\} \right\} = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix} \right\} \right\}$$

TechKnowledge Publications

$$= \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \begin{bmatrix} (x - x_r)\, S_x \\ (y - y_r)\, S_y \end{bmatrix}$$

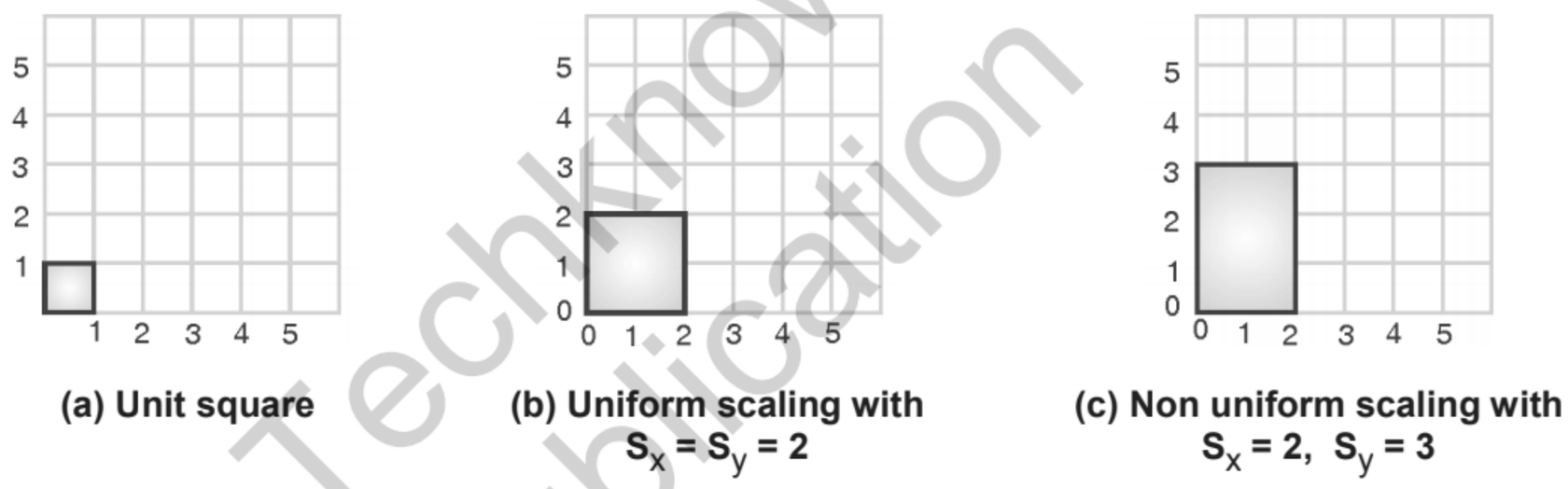
$$x' = x_r + (x - x_r)\, S_x = S_x \cdot x + x_r \cdot (1 - S_x)$$

$$y' = y_r + (y - y_r)\, S_y = S_y \cdot y + y_r \cdot (1 - S_y)$$

As scaling parameters and reference points are fixed, the terms $(x_r \cdot (1 - S_x))$ and $(y_r \cdot (1 - S_y))$ are constant for all the vertices of the object.

5.4.3 Uniform Vs. Non Uniform Scaling

- If scaling parameters are same for both the directions, it is called **uniform scaling**, otherwise, it is called **non-uniform scaling**.
- Uniform scaling only scale up or scale down the object size; it does not change the shape of the object. In the case of uniform scaling, the square remains square and circle remains circle after scaling operation, only the size of an object alters.
- Non-uniform scaling may alter the shape of the object. In the case on non-uniform scaling, the circle may turn out to be ellipse, or square may turn out to be rectangle.
- Consider the unit square with vertices A(0, 0), B(1, 0), C(1, 1) and D(0, 1). If we apply uniform scaling on this unit square with $s_x = s_y = 2$, we get A′(0, 0), B′(2, 0), C′(2, 2) and D′(0, 2). And if we apply non uniform scaling with $S_x = 2$ and $S_y = 3$, we get A″(0, 0), B″(2, 0), C″(2, 3) and D″(0, 3). Fig. 5.4.3 illustrates the concept of uniform and non-uniform scaling.
- In case of uniform scaling, the square remains square, whereas in case of non-uniform scaling square turns into a rectangle.



(a) Unit square **(b) Uniform scaling with $S_x = S_y = 2$** **(c) Non uniform scaling with $S_x = 2$, $S_y = 3$**

Fig. 5.4.3 : Uniform and non-uniform scaling

Scaling is called differential scaling if $S_x \neq S_y$. Differential scaling is used to construct complex shapes with the help of basic translation and rotation transformation. Differential scaling is also known as non-uniform scaling.

Ex. 5.4.1 : Scale a square with coordinates A(2, 2), B(4, 2), C(4, 4), D(2, 4), Where; $S_x = 3$ and $S_y = 2$.

Soln. :

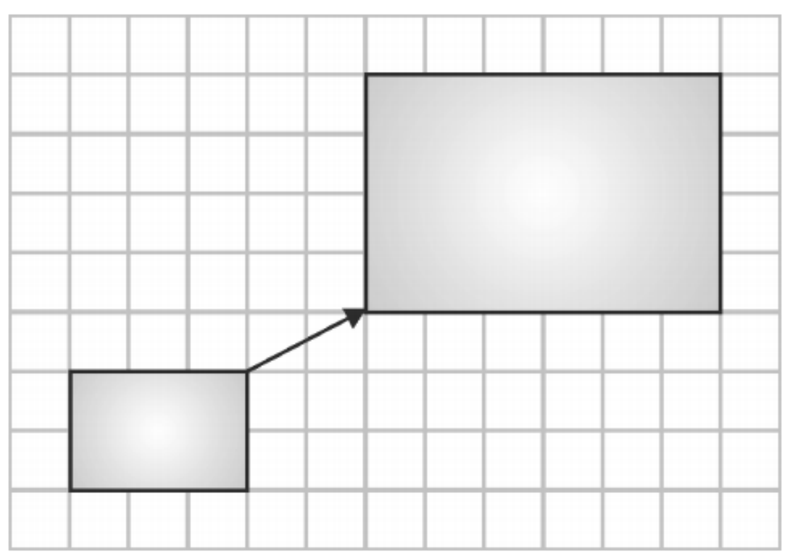


Fig. P. 5.4.1 : Scaling of object by scaling parameters, $S = [3 \quad 2]^T$

Transformation sequence for scaling is defined by,

$$P' = S \cdot P = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\therefore A' = S \cdot A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$B' = S \cdot B = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 4 \end{bmatrix}$$

$$C' = S \cdot C = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix}$$

$$D' = S \cdot D = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(2, 2)	A′(6, 4)
B(4, 2)	B′(12, 4)
C(4, 4)	C′(12, 8)
D(2, 4)	D′(6, 8)

Ex. 5.4.2 : Scale a triangle with vertices A(2, 2), B(6, 2) and C(4, 4) with respect to a reference point (3, 4) with $S_x = 2$ and $S_y = 3$.

Soln. :

Here, Scaling is with respect to a reference point. We have to first translate reference point to origin then apply scaling and then translate the reference point back to the actual location. Transformed coordinates of this sequence of operation are given by,

$$x' = S_x \cdot x + x_r(1 - S_x)$$

$$y' = S_y \cdot y + y_r(1 - S_y)$$

$x_r(1 - S_x)$ and $y_r(1 - S_y)$ are constant for all points.

$$X_r(1 - S_x) = 3(1 - 2) = -3$$

$$y_r(1 - S_y) = 4(1 - 3) = -8$$

$$A' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix}$$

$$= \begin{bmatrix} 4 \\ 6 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

$$B' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix}$$

$$= \begin{bmatrix} 12 \\ 6 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 9 \\ -2 \end{bmatrix}$$

$$C' = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix}$$

$$= \begin{bmatrix} 8 \\ 12 \end{bmatrix} + \begin{bmatrix} -3 \\ -8 \end{bmatrix} = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(2, 2)	A′(1, – 2)
B(6, 2)	B′(9, – 2)
C(4, 4)	C′(5, 4)

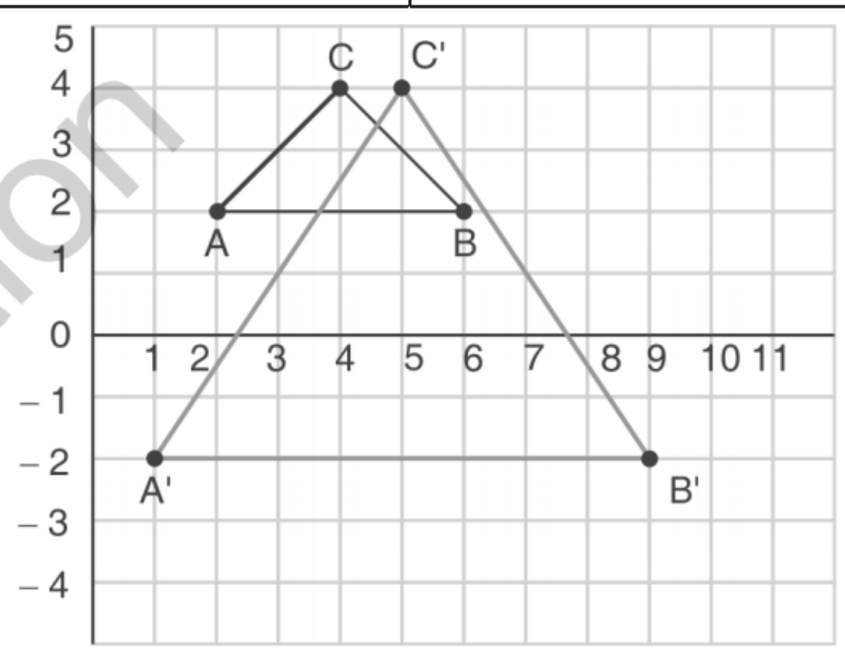


Fig. P. 5.4.2 : Output of scaling operation

Ex. 5.4.3 : Apply scaling to unit square with a bottom left corner at(3, 4) with $S_x = 2$ and $S_y = 0.5$.

Soln. :

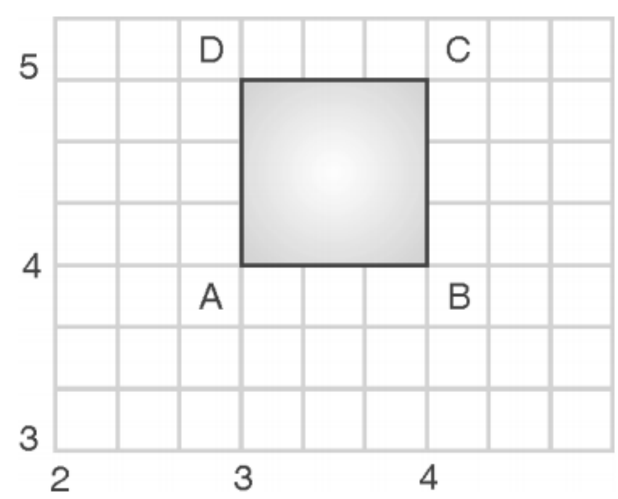


Fig. P. 5.4.3

TechKnowledge Publication

Assume that A is the corner with coordinates (3, 4). So remaining vertices of the unit square will be B(4,4), C(4,5) and D(3,5).

$$\text{Scaling matrix, } S = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

Let us assume P and P′ are original and transformed coordinates of an object respectively.

$$\therefore \quad P' = S \cdot P$$

$$= \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 3 & 4 & 4 & 3 \\ 4 & 4 & 5 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 6 & 8 & 8 & 6 \\ 2 & 2 & 2.5 & 2.5 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(3, 4)	A′(6, 2)
B(4, 4)	B′(8, 2)
C(4, 5)	C′(8, 2.5)
D(3, 5)	D′(6, 2.5)

Ex. 5.4.4 : Prove that the scaling operation is commutative.

Soln. :

- Assume that transformation matrix for scaling S_1 by parameters (s_{x1}, s_{y1}) followed by scaling S_2 with parameters (s_{x2}, s_{y2}) is $M_1 = S_1S_2$.

$$\text{L.H.S} = M_1 = S_1S_2 = \begin{bmatrix} S_{x2} & 0 \\ 0 & S_{y2} \end{bmatrix}$$

$$= \begin{bmatrix} S_{x1}S_{x2} & 0 \\ 0 & S_{y1}S_{y2} \end{bmatrix}$$

- And the transformation matrix for scaling S_2 by parameters (s_{x2}, s_{y2}) followed by scaling S_1 with parameters (s_{x1}, s_{y1}) is $M_2 = S_2S_1$.

$$\text{R.H.S} = M_2 = S_2S_1 = \begin{bmatrix} S_{x2} & 0 \\ 0 & S_{y2} \end{bmatrix} \begin{bmatrix} S_{x1} & 0 \\ 0 & S_{y1} \end{bmatrix}$$

$$= \begin{bmatrix} S_{x1}S_{x2} & 0 \\ 0 & S_{y1}S_{y2} \end{bmatrix} = \text{L.H.S}$$

- Hence the given statement is proved that scaling operation is commutative.

Ex. 5.4.5 : Perform scaling on a triangle (1, 1), (8, 1) and (1, 9) with a scaling factor of 2 in both x and y-directions. Find the final coordinates of the triangle.

SPPU - May 17, 2 Marks

Soln. :

Given the triangle A(1, 1), B(8, 1) and C(1, 9) with parameters $S_x = S_y = 2$. In homogenous coordinate representation, the transformed coordinates are given as,

$$P' = S \cdot P$$

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 8 & 1 \\ 1 & 1 & 9 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 16 & 2 \\ 2 & 2 & 18 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(1, 1)	A′(2, 2)
B(8, 1)	B′(16, 2)
C(1, 9)	C′(2, 18)

5.5 Rotation

- 2D rotation is described by repositioning all the points of an object *along a circular path* in the 2D plane.
- To perform rotation, we need a rotation angle θ and the reference point (x_r, y_r) with respect to which object is to be rotated. The reference point is also called a ***pivot point.***
- If rotation is performed in an anticlockwise direction, the value of the angle is considered positive, otherwise, it is negative. Rotation directions are shown in Fig. 5.5.1.

Techknowledge Publication

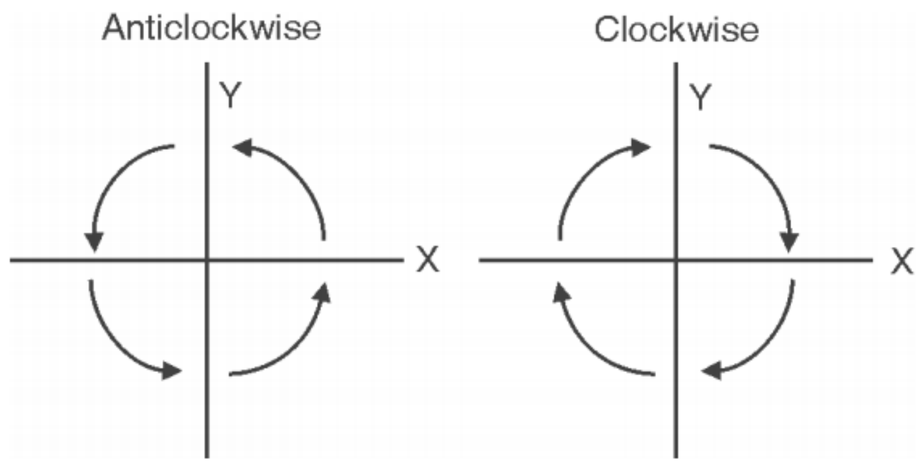


Fig. 5.5.1 : Anticlockwise and clockwise rotation direction

– Rotation can be performed with respect to origin or with respect to some reference point. We will discuss both the cases here.

5.5.1 Rotation with Respect to Origin

– Let's derive the transformation matrix for *rotation about the origin*. As shown in Fig. 5.5.2, P(x, y) is the original point, which is to be rotated in XY plane by the angle θ in an anticlockwise direction. P′(x′, y′) is the rotated point.

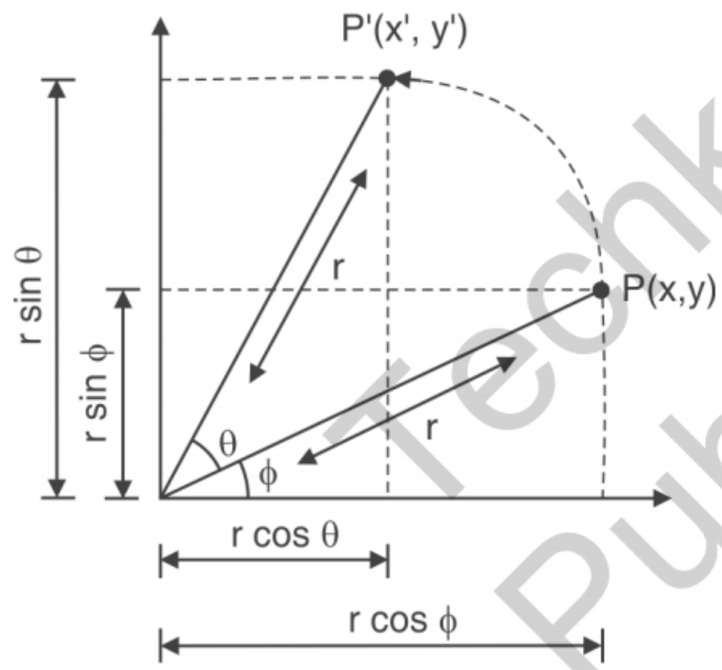


Fig. 5.5.2 : Rotational geometry of point P

– Distance of P from the origin is let's say *r*. Consider the angle of P with X axis is ϕ and is rotated by angle θ in an anticlockwise direction, so point P' makes anangle (ϕ + θ) with X-axis.

– From Fig. 5.5.2, Using trigonometric rule, for Point P,

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

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

$$\therefore\ x = r\cos\phi \qquad \ldots(5.5.1)$$

$$y = r\sin\phi \qquad \ldots(5.5.2)$$

For point P′,

$$\cos(\phi+\theta) = \frac{x'}{r}$$

$$\sin(\phi+\theta) = \frac{y'}{r}$$

$$\text{So, } x' = r\cos(\phi+\theta)$$

$$= r\cos\phi\cdot\cos\theta - r\sin\phi\cdot\sin\theta \qquad \ldots(5.5.3)$$

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

$$= r\cos\phi\cdot\sin\theta + r\sin\phi\cdot\sin\theta \qquad \ldots(5.5.4)$$

By putting value of Equations (5.5.1) and (5.5.2) in Equation (5.5.3) and Equation (5.5.4), we get

$$x' = x\cos\theta - y\sin\theta \qquad \ldots(5.5.5)$$

$$y' = x\sin\theta + y\cos\theta \qquad \ldots(5.5.6)$$

Matrix representation of above equation is written as ;

$$P' = R\cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \qquad \ldots(5.5.7)$$

Where, $$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

represents rotation matrix for rotation about origin

Inverse rotation is defined by considering negative value of θ

$$R^{-1} = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

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

$$= R^T$$

The inverse rotation matrix is equal to the transpose of the original matrix. Such a matrix is called the **orthogonal matrix**.

Properties of the Pure Rotation Matrix

– The determinant is always 1.

– The matrix should be orthogonal.

Technknowledge Publication

Note : 1. If the reference point is not specified, rotation is by default about the origin.

2. If rotation direction is not specified, by default it is in anticlockwise.

5.5.2 Rotation with Respect to Reference Point

Matrix sequence of Equation (5.5.7) is only valid for rotation about the origin. If we want to rotate any point or object with respect to some reference point (x_r, y_r), then we should perform the following steps.

1. Translate the reference point (x_r, y_r) to origin: Translation vector is given as,

$$T = \begin{bmatrix} -x_r \\ -y_r \end{bmatrix}$$

2. Apply rotation by given angle θ. The rotation matrix is given as,

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

3. Translate reference point back to its actual location.

$$T^{-1} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

The sequence of operations is shown in Fig. 5.5.3.

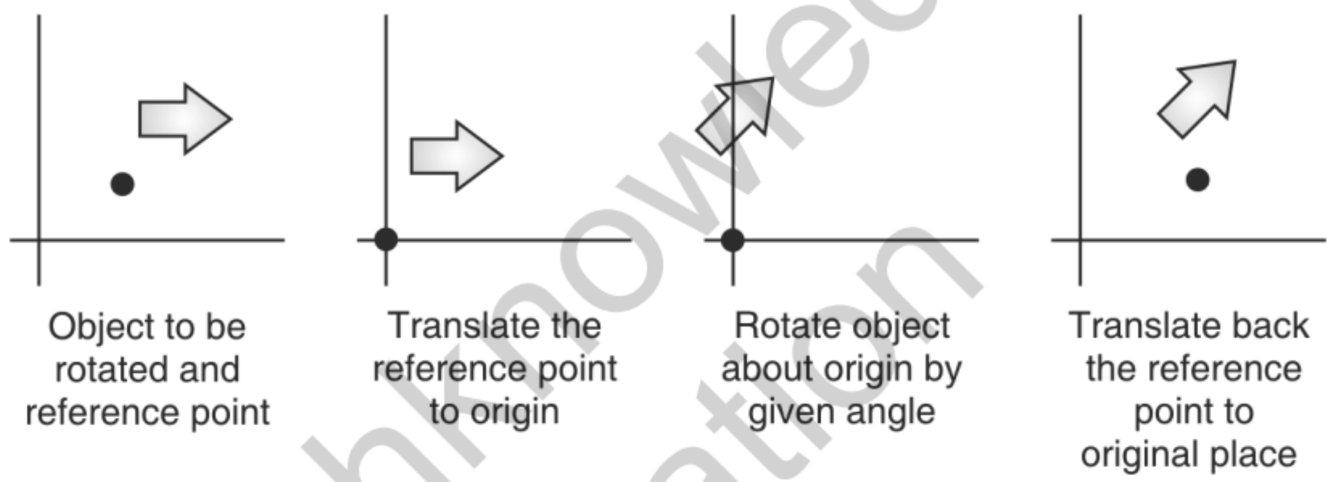


Fig. 5.5.3 : Transformation sequence for rotation with respect to the reference point

For column major representation, the sequence of operations is defined as :

$$P' = [T^{-1} + [R[T + P]]]$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \left\{ \begin{bmatrix} -x_r \\ -y_r \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \right\} \right\} \right\}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left\{ \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \left\{ \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x - x_r \\ y - y_r \end{bmatrix} \right\} \right\}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \end{bmatrix} + \begin{bmatrix} (x - x_r)\cos\theta - (y - y_r)\sin\theta \\ (x - x_r)\sin\theta + (y - y_r)\cos\theta \end{bmatrix}$$

$$\text{So, } x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

Note : For row major representation, the sequence of operations would be : $P' = [\,[[P + T]R] + T^{-1}]$

Ex. 5.5.1 : Rotate a point P(3, 2) around the origin in a counter clockwise direction by 90°.

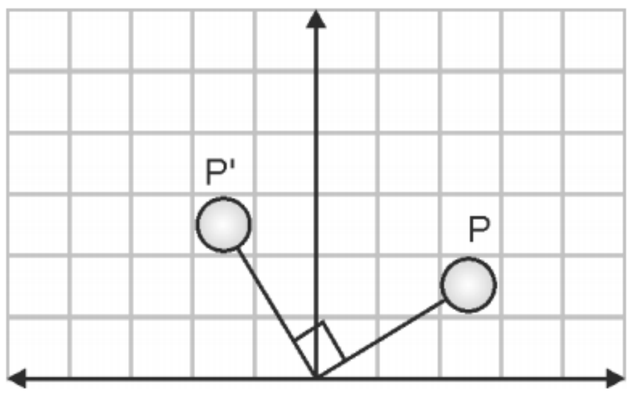


Fig. P. 5.5.1 : Rotation of point P(3, 2) by 90° in the counter-clockwise direction

Soln. :

Here, the direction of rotation is anticlockwise, So, $\theta = +90^\circ$

$$\therefore P' = R \cdot P$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \end{bmatrix}$$

Thus, point P(3, 2) becomes P′(– 2, 3) after rotation by 90°in the anti-clockwise direction.

Original Coordinates	Transformed Coordinates
A(3, 2)	A′ (– 2, 3)

Ex. 5.5.2 : Rotate a point A(3, 2) by 90° in anticlockwise direction with respect to reference point B(1, 2).

Soln. :

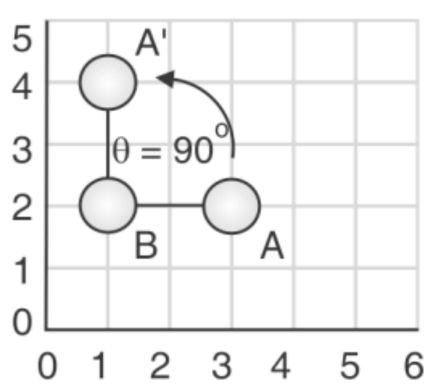


Fig. P. 5.5.2 : Output of said rotation operation

Here, Rotation direction is anticlockwise, so $\theta = +90^\circ$

We have to rotate a point around the reference point

$$\begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Rotation about reference point (x_r, y_r) is given by,

$$\therefore x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$

$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$

$$x' = 1 + (3 - 1)\cos 90^\circ - (2 - 2)\sin 90^\circ$$

$$= 1 + (2)\,0 - (0)1$$

$$\therefore x' = 1$$

$$y' = 2 + (3 - 1)\sin 90^\circ + (2 - 2)\cos 90^\circ$$

$$= 2 + (2)\,1 + (0)\,0$$

$$\therefore y' = 4$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

Thus, point A(3,2) becomes A′(1, 3) after rotation by 90°in anticlockwise direction with respect to the reference point B(1, 2).

Original Coordinates	Transformed Coordinates
A(3, 2)	A′(1, 4)

Ex. 5.5.3 : Consider a square P(0, 0), Q(0, 10), R(10, 10), S(10, 0). Rotate the square anticlockwise about fixed point R(10, 10) by an angle 45 degree. **SPPU - May 19, 4 Marks**

Soln. :

Here, given the square P(0, 0), Q(0, 10), R(10, 10), S(10, 0).

Square is to be rotated about a reference point R(10, 10) by 45°. The desired operation is achieved by performing the following steps :

1. Translate reference point R to the origin.
2. Perform rotation by 45° in an anticlockwise direction.
3. Inverse translation of point R.

The transformation matrix for this sequence is given by $M = T^{-1} \cdot R_{(\theta = 45^\circ)} \cdot T$

Technical Knowledge Publications

$$P' = M \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 10 & 10 \\ 0 & 10 & 10 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.709 & -0.709 & 0 \\ 0.709 & 0.709 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 10 & 10 \\ 0 & 10 & 10 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Multiply first two and last two matrices,

$$= \begin{bmatrix} 0.709 & -0.709 & 10 \\ 0.709 & 0.709 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -10 & -10 & 0 & 0 \\ -10 & 0 & 0 & -10 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 10 & 2.91 & 10 & 17.09 \\ -4.18 & 2.91 & 10 & 2.91 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
P (0, 0)	P′(10, – 4.18)
Q (0, 10)	Q′(2.91, 2.91)
R (10, 10)	R′(10, 10)
S (10, 0)	S′(17.09, 2.91)

Ex. 5.5.4 : Find a transformation of a triangle A(1, 0) B(0, 1) C(1, 1) by translating one unit in x and y directions and then rotating 45° about the origin. **SPPU - Dec. 19, 6 Marks**

Soln. :

Desired transformation is achieved by following sequence of operations :

$$P' = R_{(\theta = 45)} . T . P$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 45° & -\sin 45° & 0 \\ \sin 45° & \cos 45° & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.71 & -0.71 & 0 \\ 2.13 & 2.13 & 2.84 \\ 1 & 1 & 1 \end{bmatrix}$$

Technknowledge Publications

Original Coordinates	Transformed Coordinates
A(1, 0)	A′ (0.71, 2.13)
B(0, 1)	B′(–0.71, 2.13)
C(1, 1)	C′(0, 2.84)

Program 5.5.1 : Write C++/Java program to draw 2-D object and perform following basic transformations,

a) Scaling

b) Translation

c) Rotation

Use operator overloading.

```
#include<iostream>
#include<graphics.h>
#include<stdlib.h>
#include<math.h>
using namespace std;

int pi[10];
double b[3][3]={ 1,0,0,
                 0,1,0,
                 0,0,1,};
int c[1][1];
float a[1][1];
void matmul(float[ ]);

int main()
{
    int op,i,x,y,tx,ty,sx,sy,angle=10,xmax,ymax,xmid,ymid;
    int gd,gm=VGAMAX;
    gd=DETECT;
    float p1[10]={50,50,
                  100,50,
                  100,100,
                  50,100,
                  50,50,};
    cout<<"\n select transformation:=";
    cout<<"\n 1.translation:=";
    cout<<"\n 2.rotation:=";
    cout<<"\n 3.scaling:=";
    cout<<"\n 4.rotation about arbitrary point:=";
    cout<<"\n 5.enter the option:=";
    cin>>op;
```

Techknowledge Publications

```
switch(op)
{
  case 1:
        cout<<"\n enter the x translation:=";
        cin>>tx;
        cout<<"\n enter the y translation:=";
        cin>>ty;
        b[0][0]=1;
        b[0][1]=0;
        b[0][2]=0;
        b[1][0]=0;
        b[1][1]=1;
        b[1][2]=0;
        b[2][0]=tx;
        b[2][1]=ty;
        b[2][2]=1;
        break;

  case 2:
        cout<<"\n enter the rotation angle :";
        cin>>angle;
        b[0][0]=cos(angle*3.142/180);
        b[0][1]=sin(angle*3.142/180);
        b[0][2]=0;
        b[1][0]=-sin(angle*3.142/180);
        b[1][1]=cos(angle*3.142/180);
        b[1][2]=0;
        b[2][0]=0;
        b[2][1]=0;
        b[2][2]=1;
        break;

  case 3:
        cout<<"\n enter the x scaling:";
        cin>>sx;
        cout<<"\n enter y scaling:";
        cin>>sy;
        b[0][0]=sx;
        b[0][1]=0;
        b[0][2]=0;
        b[1][0]=0;
        b[1][1]=sy;
```

Technknowledge Publication

```
        b[1][2]=0;
        b[2][0]=0;
        b[2][1]=0;
        b[2][2]=1;
        break;

    case 4:
        cout<<"\n enter x coordinate of arbitatary point :"; cin>>x;
        cout<<"\n enter x coordinate of arbitatary point:"; cin>>y;
        cout<<"\n enter the rotation angle:=";
        cin>>angle;
        tx=x;
        ty=y;
        b[0][0]=cos(angle*3.142/180);
        b[0][1]=sin(angle*3.142/180);
        b[0][2]=0;
        b[1][0]= –sin(angle*3.142/180);
        b[1][1]=cos(angle*3.142/180);
        b[1][2]=0;
        b[2][0]= –tx*cos(angle*3.142/180)+ty*sin(angle*3.142/180)+tx;
        b[2][1]= –tx*sin(angle*3.142/180)–ty*cos(angle*3.142/180)+ty; b[2][2]=1;
    }

    initgraph(&gd,&gm,NULL);        //initialize graphics
    xmax=getmaxx();                 //get the maximum y coordinate
    ymax=getmaxy();                 //get the maximum x co-ordinate
    xmid=xmax/2;                    //get the center x coordinate
    ymid=ymax/2;                    //get the center y coordinate
    setcolor(1);
    line(xmid,0,xmid,ymax);
    line(0,ymid,xmax,ymid);
      //draw y coordinate
      //draw x coordinate
    setcolor(4);
    for(i=0;i<8;i=i+2)
    {
      line(p1[i]+xmid,ymid-p1[i+1],xmid+p1[i+2],ymid-p1[i+3]);
    }
    matmul(p1);
    setcolor(15);
```

Techknowledge Publications

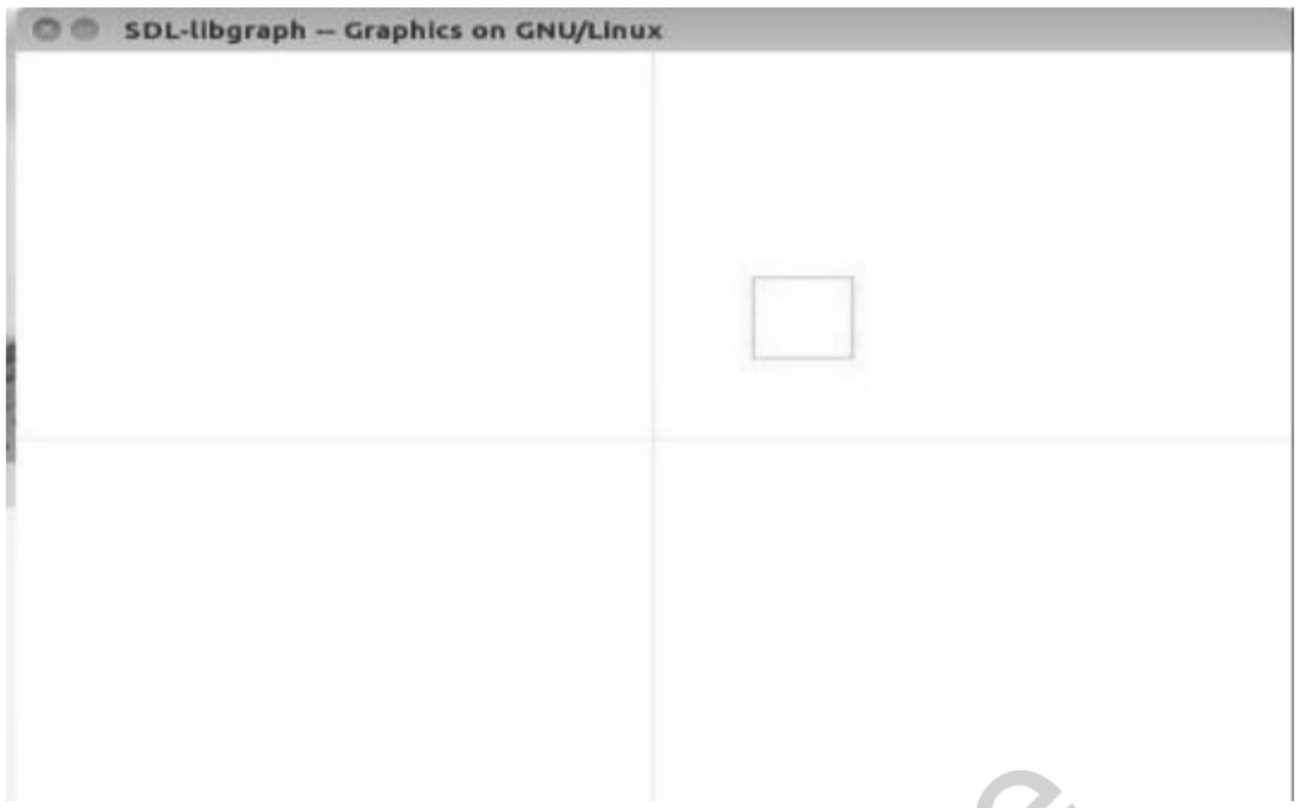
```
    for(i=0;i<8;i=i+2)
    {
      line(xmid+pi[i],ymid-pi[i+1],xmid+pi[i+2],ymid-pi[i+3]);
    }
    getch();
    closegraph();
    return 0;
}

void matmul(float p[10])
{
    int i;
    for(i=0;i<9;i=i+2)
    {
      a[0][0]=p[i];
      a[0][1]=p[i+1];
      c[0][0]=a[0][0]*b[0][0]+a[0][1]*b[1][0]+b[2][0];
      c[0][1]=a[0][0]*b[0][1]+a[0][1]*b[1][1]+b[2][1];
      pi[i]=c[0][0];
      pi[i+1]=c[0][1];
    }
}
```

Output :

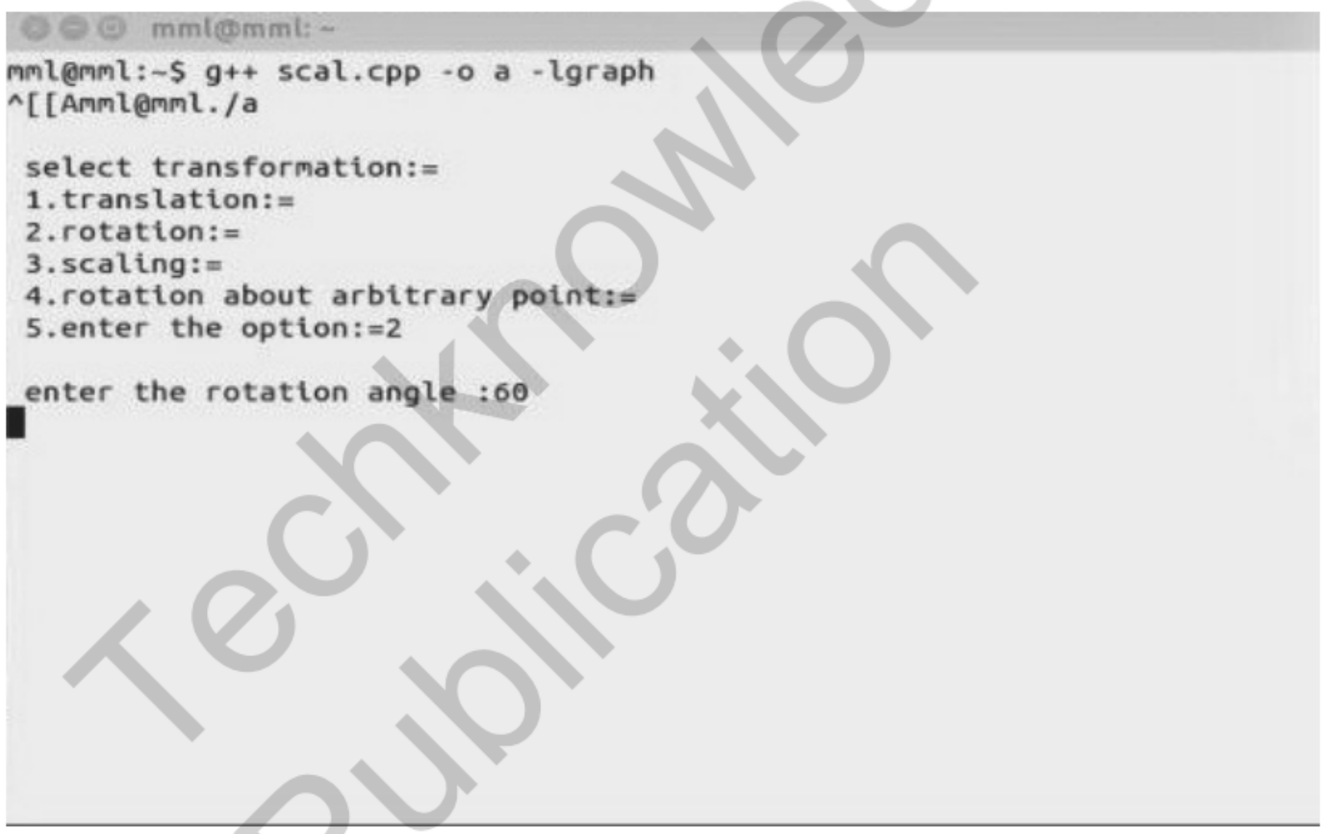
1) Scaling

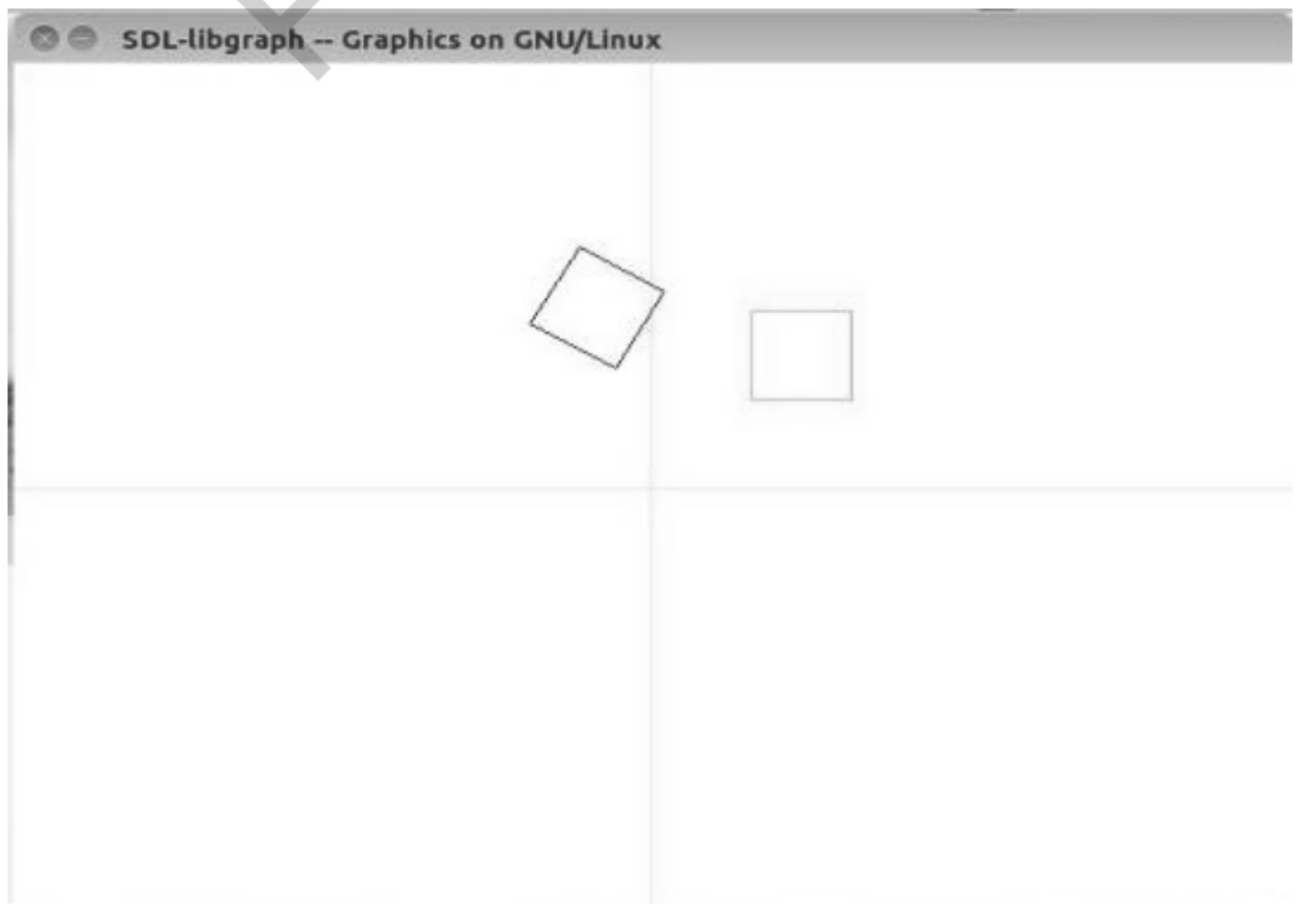
```
mml@mml:~$ g++ scal.cpp -o a -lgraph
mml@mml:~$ ./a

 select transformation:=
 1.translation:=
 2.rotation:=
 3.scaling:=
 4.rotation about arbitrary point:=
 5.enter the option:=3

 enter the x scaling:30

 enter y scaling:40
Pixel request out of range!!    (1819,-1761)
Pixel request out of range!!    (3319,-3761)
Pixel request out of range!!    (1819,-3761)
Pixel request out of range!!    (1819,-3761)
```

Techknowledge Publication



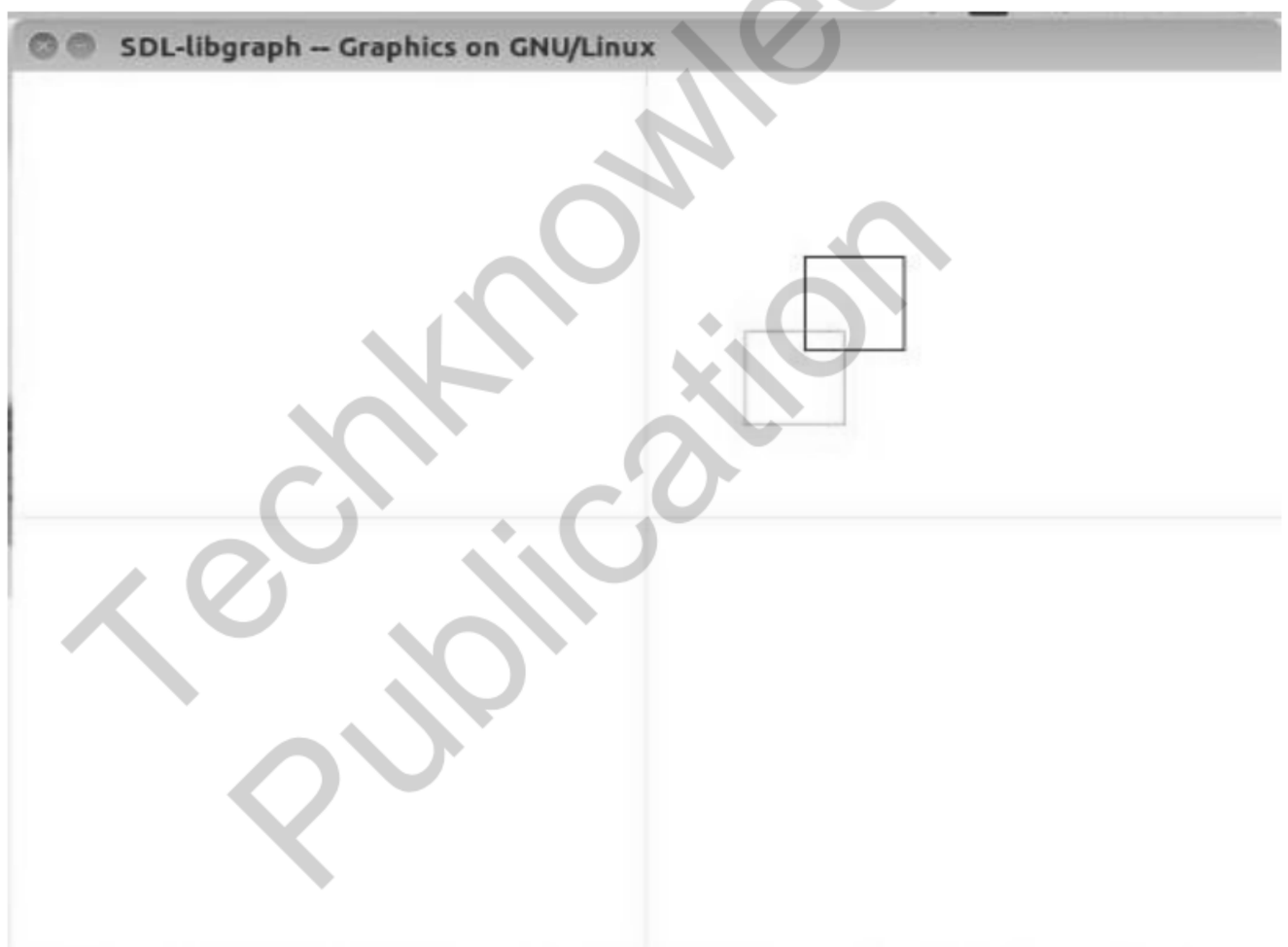
2) Rotation





TechKnowledge Publications

3) Translation



Techknowledge Publications

5.6 Homogeneous Coordinates

Q. What is a homogenous coordinate system ?

SPPU - May 16, Dec. 18, 2 Marks

- In real-world applications, it is very common that the object goes through sequences of different geometric transformations.
- The object may require translation, rotation, scaling, shearing or reflection to be applied any number of times in any order.
- In previous sections, we saw that translation operation is additive while scaling and rotation are multiplicative. So all three basic transformations can be represented in generalized form as,

$$P' = M_1 + M_2P \qquad \ldots(5.6.1)$$

- For translation, M_2 is the identity matrix (multiplicative identity).

  So, $P' = M_1 + P$ represents the translation matrix

  And for rotation and scaling, M_1 is zero matrix (additive identity),

So, $P' = M_2P$, Where M_2 represents rotation or scaling matrix

- Point (x, y) in the Cartesian coordinate has infinite representation in homogeneous coordinates.
- For example, (x, y) = (x, y, 1) = (2x, 2y, 2) = ... = (nx, ny, n), all correspond to the same representation of (x, y).

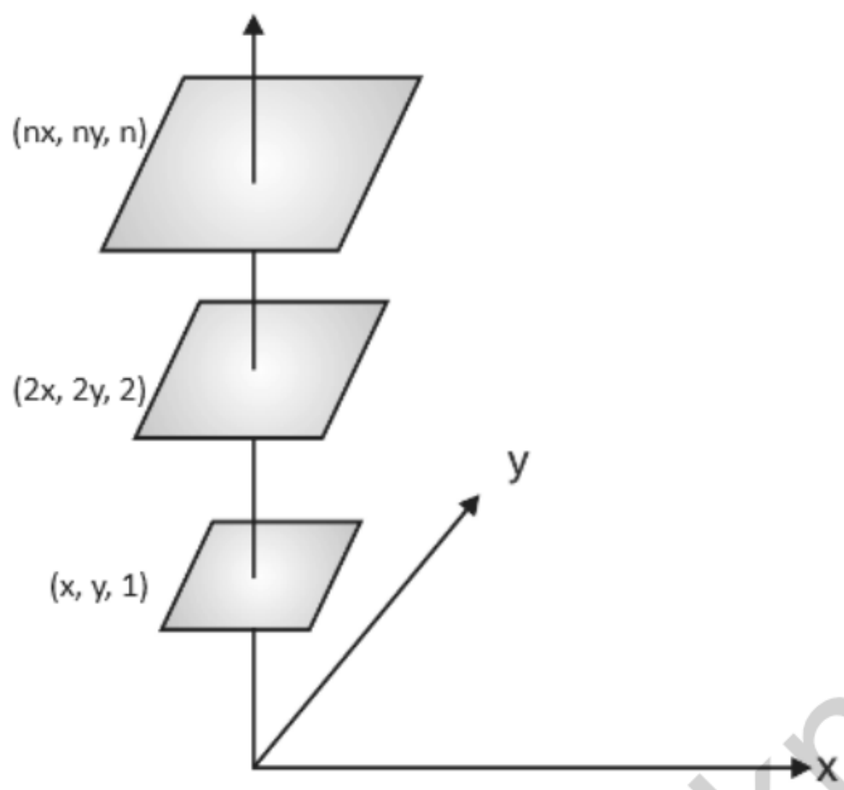


Fig. 5.6.1 : Representation of point in the homogeneous coordinate system

- We will reformulate this matrix representation of Equation (5.6.1) to combine additive and multiplicative matrices in a single matrix by extending 2 × 2 representation to 3 × 3 using homogeneous representation.
- In the homogeneous coordinate system, if we multiply the point by some constant, it represents the same point.
- Any Cartesian coordinate (x, y) can be represented as (x_h, y_h, h) in a homogeneous system, such that

$$x = \frac{x_h}{h} \text{ and } y = \frac{y_h}{h}$$

- Homogeneous coordinate introduces the third dimension but it doesn't alter the value of (x, y).
- Fig. 5.6.1 explains how we can represent the same points at different levels.

Properties of Homogenous Coordinate Representation

1. Any 2D point in the homogeneous coordinate system is represented by a triplet (x, y, h), where x, y and h are not all zero. (0, 0, 0) does not represent any point. Origin is represented as (0, 0, 1).
2. In homogeneous coordinate systems, two points are identical, if one point is derived by multiplying some constant to the second point.
3. If h is not zero, then point (x_h, y_h, h) in a homogenous coordinate system is represented as $(x_h/h, y_h/h)$ in the Euclidean/Cartesian coordinate system.
4. If h is 0, point represented is at infinity.
   - Homogenous representation allows us to write all geometric transformation equations in matrix multiplication form.
   - It brings uniformity in operation. Implementation of transformation operation in programming language becomes easier with this representation as all the operations are performed using matrix multiplication operations only.
   - Basic transformation operations matrix using homogeneous coordinate are shown below.

5.6.1 Translation

Translation operation in the homogenous system can be represented as follow :

$$P' = T \cdot P$$

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

$$x' = x + t_x$$

$$y' = y + t_y$$

$$1 = 1$$

With homogeneous representation, translation operation is converted to multiplicative operation.

Technical Knowledge Publication

5.6.2 Scaling

Scaling operation in a homogenous system can be represented as follow :

$$P' = S \cdot P$$

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

$$x' = x \cdot S_x$$

$$y' = y \cdot S_y$$

$$1 = 1$$

5.6.3 Rotation

Q. Write transformation matrices for 2-D rotation clockwise direction. **SPPU - Dec. 18, 3 Marks**

Rotation operation in homogenous system can be represented as follow :

$$P' = R \cdot P$$

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

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

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

$$1 = 1$$

5.7 Composite Transformation

- Composite transformation is any sequence of basic transformation operations.
- In practice, pure operations (translation, rotation or scaling) are rare. In interactive graphics packages, designer often place the object on screen and adjusts its size, location and orientation incrementally. Originally placed object goes through sequence of various basic transformation operations.
- Combination of basic transformation operations is called composite transformation.
- We can generate a composite transformation matrix by multiplying matrices of basic transformation operations like translation, scaling, rotation, reflection, shearing etc.
- For column measure representation matrices are placed from right to left.
- Let's say $M_1, M_2, \ldots, M_n$ are the transformation matrices of some transformation operation in the same order, so for column measure representation of transformed coordinates is given as,

$$P' = M_n \ldots M_2 \cdot M_1 \cdot P$$

- Where M_i indicates i^{th} operation in sequence.
- For row measure representation, transformed coordinates are computed as follow :

$$P'^T = P^T \cdot M_1^T \cdot M_2^T \ldots\ldots M_n^T$$

5.7.1 Successive Transformations

1. Translation

Let us translate the point (x, y) by $[t_{x1} \quad t_{y1}]^T$ followed by $[t_{x2} \quad t_{y2}]^T$. Here translation by $M_1 = [t_{x1} \quad t_{y1}]^T$ is the first operation, which is followed by translation vector $M_2 = [t_{x2}\ t_{y2}]^T$.

So transformed coordinates are given as,

$$P' = M_2 \cdot M_1 \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x2} \\ 0 & 1 & t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x1} + t_{x2} \\ 0 & 1 & t_{y1} + t_{y2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Thus, two successive transformations are additive.

2. Rotation

Let us rotate the point (x, y) by the angle θ_1 followed by angle θ_2. Here rotation by angle θ_1 is the first operation, let's call the transformation matrix M_1, which is followed by rotation with θ_2 angle, call the transformation matrix for this rotation M_2. So transformed coordinates P' are given as,

Techknowledge Publication

$$P' = M_2 \cdot M_1 \cdot P = R(\theta_2) \cdot R(\theta_1) \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_2) & -\sin(\theta_2) & 0 \\ \sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) & -\sin(\theta_1)\cos(\theta_2) - \sin(\theta_2)\cos(\theta_1) & 0 \\ \sin(\theta_1)\cos(\theta_2) + \sin(\theta_2)\cos(\theta_1) & \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = R(\theta_2 + \theta_1) \cdot P$$

$\therefore$ Successive rotation is additive.

3. Scaling

By applying two successive scaling with scaling parameters $S_1 = [S_{x1}\ S_{y1}]^T$ and $S_2 = [S_{x2}\ \ S_{y2}]^T$on point $[x\ \ y]^T$, we get

$$P' = S_2 \cdot S_1 \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x2}S_{x1} & 0 & 0 \\ 0 & S_{y2}S_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\therefore\ x' = x \cdot (S_{x2}\, S_{x1})$$

$$y' = y \cdot (S_{y2}\, S_{y1})$$

Thus, two successive scaling is multiplicative.

5.8 Rotation About an Arbitrary Point

Q.	Explain the concept of 2D rotation about an arbitrary point with matrix representation.	**SPPU - May 15, 6 Marks**
Q.	Explain rotation about an arbitrary point. Generate a transformation matrix for the same.	**SPPU - Dec. 15, 6 Marks**
Q.	Derive transformation matrix for rotation about an arbitrary point.	**SPPU - May 16, 8 Marks**
Q.	Explain rotation about an arbitrary point in 2D.	**SPPU - Dec. 17, 4 Marks**
Q.	Write transformation matrices for 2D rotation about an arbitrary point.	**SPPU - Dec. 18, 3 Marks**

We have already discussed the rotation of the object with respect to some pivot point (x_r, y_r). In this section, we will derive the generalized 3×3 transformation matrix for rotation with respect to a reference (pivot) point.

Techknowledge Publications

General pivot point rotation can be performed by a sequence of the following steps :

1. Translate the pivot point to the origin.
2. Rotate the object.
3. Translate back the pivot point to its original place.

The sequence of operations is shown in Fig. 5.8.1.

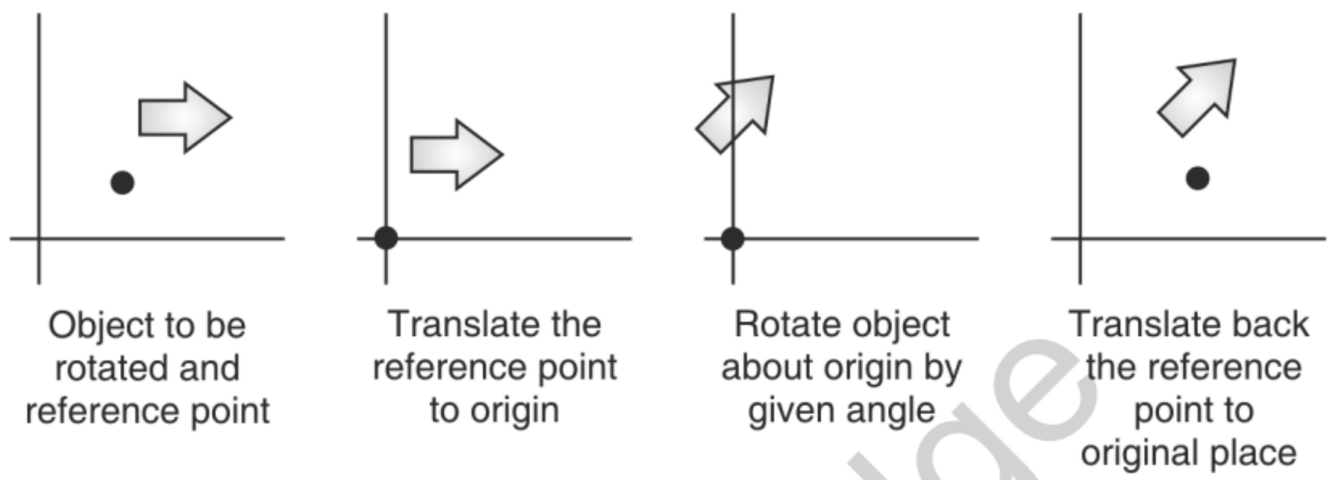


Fig. 5.8.1 : Transformation sequence for rotation with respect to the reference point

If M is the final transformation matrix, then

$$M = T^{-1} \cdot R \cdot T = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & -x_r\cos\theta + y_r\sin\theta \\ \sin\theta & \cos\theta & -x_r\sin\theta - y_r\cos\theta \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1-\cos\theta) + y_r\sin\theta \\ \sin\theta & \cos\theta & y_r(1-\cos\theta) - x_r\sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Above matrix represents the transformation matrix for rotation about a pivot point in homogenous coordinate.

Ex. 5.8.1 : Write 3×3 transformation matrix for each of the following rotation about the origin

(a) Counter clockwise rotation by 180°, (b) Clockwise rotation by 90°.

Soln. :

(a) 3×3 transformation matrix for counter-clockwise rotation by 180° about the origin.

$$M_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(180°) & -\sin(180°) & 0 \\ \sin(180°) & \cos(180°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) 2×2 transformation matrix for clockwise rotation by 90°, So $\theta = -90°$

$$M_2 = \begin{bmatrix} \cos(-90°) & -\sin(-90°) & 0 \\ \sin(-90°) & \cos(-90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ex. 5.8.2 : Rotate a triangle defined by A(0, 0), B(6, 0) and C(3, 3) by 90° about the origin in the anticlockwise direction.

Soln. :

Here, rotation is to be performed with respect to origin by 90° in anticlockwise rotation. So $\theta = +90°$.

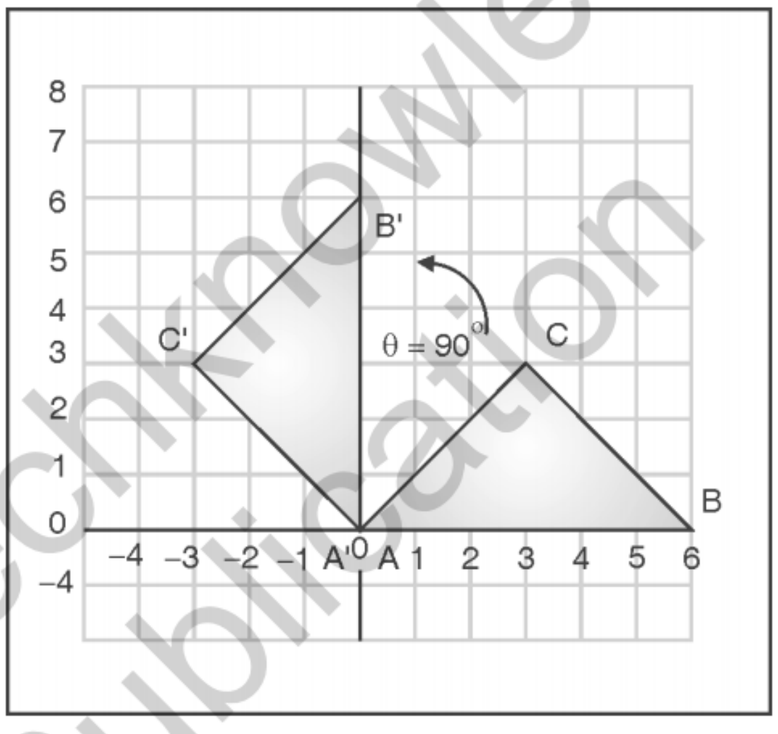


Fig. P. 5.8.2 : Output of said rotation operation

The transformed coordinates of the triangle are computed as,

$$P' = R \cdot P = \begin{bmatrix} \cos(90°) & -\sin(90°) & 0 \\ \sin(90°) & \cos(90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 6 & 3 \\ 0 & 0 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -3 \\ 0 & 6 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

Original coordinates	Transformed coordinates
A (0, 0)	A′ (0, 0)
B (6, 1)	B′ (0, 6)
C (3, 3)	C′ (– 3, 3)

Technknowledge Publication

Ex. 5.8.3 : Consider the square A(1, 0), B(0, 0), C(0, 1), D(1, 1). Rotate the square ABCD by 45° anticlockwise about point A(1, 0).

Soln. :

Square is to be rotated about a reference point A(1, 0) by 45°. The desired operation is achieved by performing the following steps :

1. Translate reference point A(1, 0) to the origin.
2. Perform rotation by 45° in an anticlockwise direction.
3. Inverse translation of point A.

The transformation matrix for this sequence is given by $M = T^{-1} . R_{(\theta = 45^\circ)} . T$

$$P' = M \cdot P$$

$$P' = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(45°) & -\sin(45°) & 0 \\ \sin(45°) & \cos(45°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.709 & -0.709 & 0 \\ 0.709 & 0.709 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Multiply first two and last two matrices,

$$= \begin{bmatrix} 0.709 & -0.709 & 1 \\ 0.709 & 0.709 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & -1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 1 & 0.291 & -0.418 & 0.291 \\ 0 & -0.709 & 0 & 0.709 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A (1, 0)	A′ (1, 0)
B (0, 0)	B′ (0.291, – 0.709)
C (0, 1)	C′ (– 0.418, 0)
D (1, 1)	D′ (0.291, 0.709)

Ex. 5.8.4 : Perform 45° rotation of a triangle A(0, 0), B(1, 1) and C(5, 2). Find transformed coordinates after rotation, (a) About origin, (b) About P(– 1, – 1).

Soln. :

Triangle A(0, 0), B(1, 1), C(5, 2) is to be rotated by 45°.

(a) Rotation with respect to the origin.

$$P' = M \cdot P = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{3}{\sqrt{2}} \\ 0 & \sqrt{2} & \frac{7}{\sqrt{2}} \\ 1 & 1 & 1 \end{bmatrix}$$

Original coordinates	Transformed coordinates
A (0, 0)	A′ (0, 0)
B (1, 1)	B′ (0, $\sqrt{2}$)
C (5, 2)	C′ ($\frac{3}{\sqrt{2}}, \frac{7}{\sqrt{2}}$)

(b) Rotation with respect to P (– 1, – 1)

Rotated coordinates with respect to some reference point (x_r, y_r) is given by,

$$x' = x_r + (x - x_r) \cdot \cos\theta - (y - y_r) \cdot \sin\theta$$

$$y' = y_r + (x - x_r) \cdot \sin\theta + (y - y_r) \cdot \cos\theta$$

Let's consider $A'(x_1', y_1')$, $B'(x_2', y_2')$ and $C'(x_3', y_3')$

$$x_1' = -1 + (0+1) \cdot \frac{1}{\sqrt{2}} - (0+1) \cdot \frac{1}{\sqrt{2}} = -1 + \frac{1}{\sqrt{2}} - \frac{1}{\sqrt{2}} = -1$$

$$y_1' = -1 + (0+1) \cdot \frac{1}{\sqrt{2}} + (0+1)\frac{1}{\sqrt{2}} = -1 + \frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}} = -1 + \sqrt{2}$$

$$x_2' = -1 + (1+1) \cdot \frac{1}{\sqrt{2}} - (1+1)\frac{1}{\sqrt{2}} = -1 + \frac{2}{\sqrt{2}} - \frac{2}{\sqrt{2}} = -1$$

$$y_2' = -1 + (1+1) \cdot \frac{1}{\sqrt{2}} + (1+1)\frac{1}{\sqrt{2}} = -1 + \frac{2}{\sqrt{2}} + \frac{2}{\sqrt{2}} = -1 + 2\sqrt{2}$$

$$x_3' = -1 + (5+1) \cdot \frac{1}{\sqrt{2}} - (2+1) \cdot \frac{1}{\sqrt{2}} = -1 + \frac{6}{\sqrt{2}} - \frac{3}{\sqrt{2}} = -1 + \frac{3}{\sqrt{2}}$$

$$y_3' = -1 + (5+1)\frac{1}{\sqrt{2}} + (2+1) \cdot \frac{1}{\sqrt{2}} = -1 + \frac{6}{\sqrt{2}} + \frac{3}{\sqrt{2}} = -1 + \frac{9}{\sqrt{2}}$$

$$\therefore \quad A' = (-1, -1 + \sqrt{2}),$$

$$B' = (-1, -1 + 2\sqrt{2}), \quad C' = \left(-1 + \frac{3}{\sqrt{2}}, -1 + \frac{9}{\sqrt{2}}\right)$$

Original Coordinates	Transformed Coordinates
A (0, 0)	A′ $(-1, -1 + \sqrt{2})$
B (1, 1)	B′ $(-1, -1 + 2\sqrt{2})$
C (5, 2)	C′ $\left(-1 + \frac{3}{\sqrt{2}}, -1 + \frac{9}{\sqrt{2}}\right)$

Tech Knowledge Publications

Ex. 5.8.5 : Consider a triangle with vertices A(1, 1), B(5, 2) and C(3, 4). Find out the transformation matrix which rotates given triangle about point C(3, 4) by an angle 300° clockwise. Also, find the coordinates of the rotated triangle.

Soln. :

The given triangle is A(1, 1), B(5, 2), (3, 4). This triangle is to be rotated about C(3, 4) by 30° in a clockwise direction,

So, $\theta = -30°$

Transformation sequence would be :

1. Translate C(3, 4) to the origin.
2. Rotate by 30° in a clockwise direction.
3. Inverse translation by shift vector C.

∴ Transformation matrix,

$$M = T^{-1} \cdot R_{(\theta = -30°)} \cdot T$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-30°) & -\sin(-30°) & 0 \\ \sin(-30°) & \cos(-30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -4 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{-4-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{3-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{2-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{11-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates,

$$P' = M \cdot P = \begin{bmatrix} \frac{\sqrt{3}}{2} & \frac{1}{2} & \frac{2-3\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} & \frac{11-4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 5 & 3 \\ 1 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = M \cdot P = \begin{bmatrix} \frac{3-2\sqrt{3}}{2} & \frac{2\sqrt{3}+4}{2} & 3 \\ \frac{10-3\sqrt{3}}{2} & \frac{6-2\sqrt{3}}{2} & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

Technknowledge Publications

Original Coordinates	Transformed Coordinates
A (1, 1)	$A'\left(\frac{3-2\sqrt{3}}{2}, \frac{10-3\sqrt{3}}{2}\right)$
B (5, 2)	$B'\left(\frac{2\sqrt{3}+4}{2}, \frac{6-2\sqrt{3}}{2}\right)$
C (3, 4)	C′ (3, 4)

5.9 Inverse Transformation

Q. What is inverse transformation? Explain with an example. **SPPU - Dec. 18, 4 Marks**

– Inverse translation and inverse rotation are achieved by flipping the sign of transformation parameters.

– The inverse transformation matrices for both the operations are shown below :

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation matrix Inverse translation matrix Rotation matrix Inverse rotation matrix

Fig. 5.9.1 : Transformation and inverse transformation matrices for translation and rotation operation

– Inverse scaling operation is achieved by replacing scaling parameters by their reciprocals.

$$\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling matrix Inverse scaling matrix

Fig. 5.9.2 : Transformation and inverse transformation matrices for scaling operation

– Inverse transformation of matrix M is represented by M^{-1}.

– M followed by M^{-1} produces the identity operation for the object.

$$M \cdot M^{-1} = I$$

Where I is the identity matrix.

5.10 Shear

– A transformation that slants the shape of an object is called the shear transformation. Shearing is typically applied to soft objects like rubber, sheet, cloth etc.

– When we apply a horizontal force to the book which is kept upright, it bends in the direction of the force. Pages of the books will slide over each other. Top pages will be displaced more than the pages at bottom.

– Shearing deforms the shape of an object and hence it is not rigid-body transformation.

– This deformation is proportional to shearing force and distance of the point of an object from the baseline.

Tech Knowledge Publications

5.10.1 X- Direction Shearing

- We can treat the object as a collection of layers. When we apply the horizontal force to the object, layers will slide. Neither the base layer alters nor the height of any internal layer changes.
- Effect of shearing in X direction is depicted in Fig. 5.10.1.

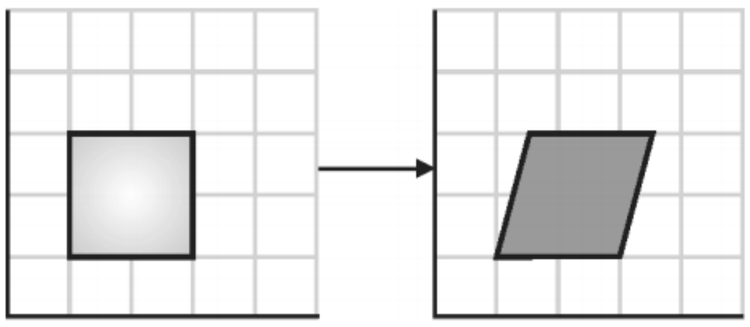


Fig. 5.10.1 : Shape deformation after shearing in the X direction

- It is intuitive that as the height of the internal layers in object increases from the base, its displacement would be more. So change in x-coordinate depends on two parameters :
  1. Height of internal layer from the base.
  2. The magnitude of the force.

  So, transformed coordinates are given by

$$x' = x + Sh_x \cdot y$$

$$y' = y$$

- Where Sh_x is a shear parameter (force) in X-direction and y is the height of the internal layer from the base.
- The transformation matrix for X-direction shearing is,

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

Transformation coordinates are computed as,

$$P' = SH_x \cdot P$$

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

Ex. 5.10.1 : Apply shearing on cube with vertices A(0, 0), B(2, 0), C(2, 2) and D(0, 2), where $Sh_x = 3$.

Soln. : For x-direction shearing with y = 0 as reference line

$$x' = x + Sh_x \cdot y$$

$$y' = y$$

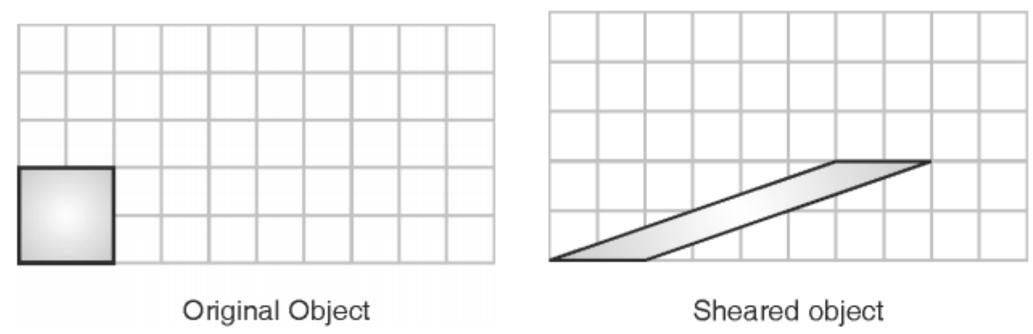


Fig. P. 5.10.1 : Shearing in the X direction

For Point A, $x' = 0 + 3(0) = 0$

For Point B, $x' = 2 + 3(0) = 2$

For Point C, $x' = 2 + 3(2) = 8$

For Point D, $x' = 0 + 3(2) = 6$

Original Co-ordinates	Transformed Co-ordinates
A (0, 0)	A′ (0, 0)
B (2, 0)	B′ (2, 0)
C (2, 2)	C′ (8, 2)
D(0, 2)	D′ (6, 2)

5.10.2 Shearing About a Line Parallel to X-axis

Like other operations, the previous transformation matrix is also position-dependent. The transformation matrix SH_x derived above is applicable only when the base of the object is on X-axis, means when the reference line is Y = 0.

We can derive generalized matrix for any reference line $Y = Y_{ref}$ parallel to X-axis using the following steps :

1. Translate line by $-Y_{ref}$ so the line gets aligned with X-axis.
2. Apply pure shear operation in the X-direction.
3. Translate line by $+Y_{ref}$ to bring it back to its original position.

Technknowledge Publications

$\therefore$ The composite transformation matrix would be,

$$M = T^{-1} \cdot SH_x \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & Sh_x & -y_{ref} \cdot Sh_x \\ 0 & 1 & -y_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & Sh_x & -y_{ref} \cdot Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ex. 5.10.2 : Apply shearing on a unit square whose base is on line $y_{ref} = -2$ with $Sh_x = 2$.

Soln. :

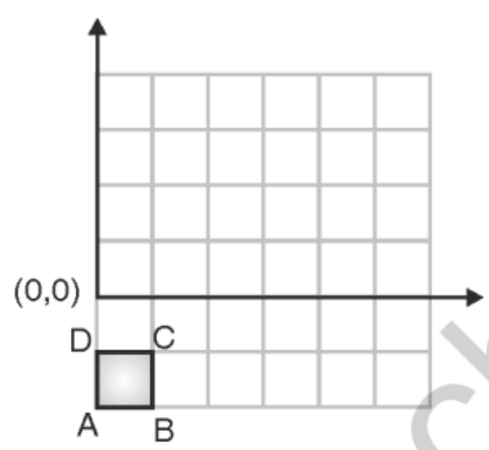


Fig. P. 5.10.2

$$P' = M \cdot P$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & Sh_x & -y_{ref} \cdot Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\therefore x' = x + y \cdot Sh_x - y_{ref} \cdot Sh_x$$

$$y' = y$$

Y-coordinate of all point will remain as it is, the only x will change.

For point A, $x' = 0 + 2(-2) - (-2)2 = 0$

For point B, $x' = 1 + 2(-2) - (-2)2 = 1$

For point C, $x' = 1 + (-1)2 - (-2)2 = 3$

For point D, $x' = 0 + (-1)2 - (-2)2 = 2$

Position of a sheared cube is shown in Fig. P. 5.10.2(a).

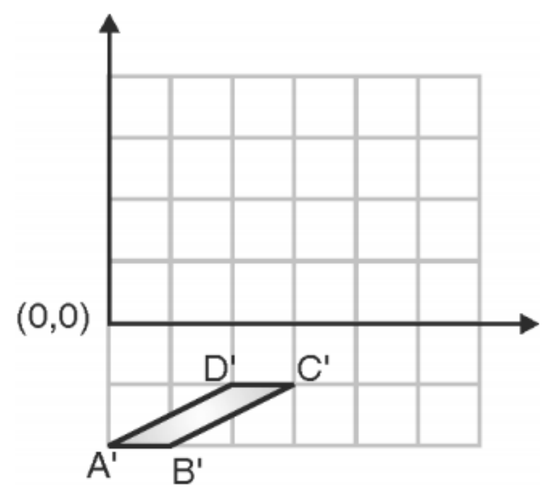


Fig. P. 5.10.2(a)

Original Coordinate	Transformed Coordinate
A(0, – 2)	A′ (0, – 2)
B(1, – 2)	B′ (1, – 2)
C(1, – 1)	C′ (3, – 1)
D(0, – 1)	D′ (2, – 1)

5.10.3 Y- Direction Shearing

- When shearing is applied in the y-direction, only y coordinates will change, x coordinates remain unchanged.
- The change in the y-direction is proportional to the magnitude of its x-coordinate and force applied in the y-direction.

$$\therefore x' = x$$

$$y' = y + x \cdot Sh_y$$

Techknowledge Publication

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

– Following Fig. 5.10.2 depicts the result of shearing operation in Y-direction.

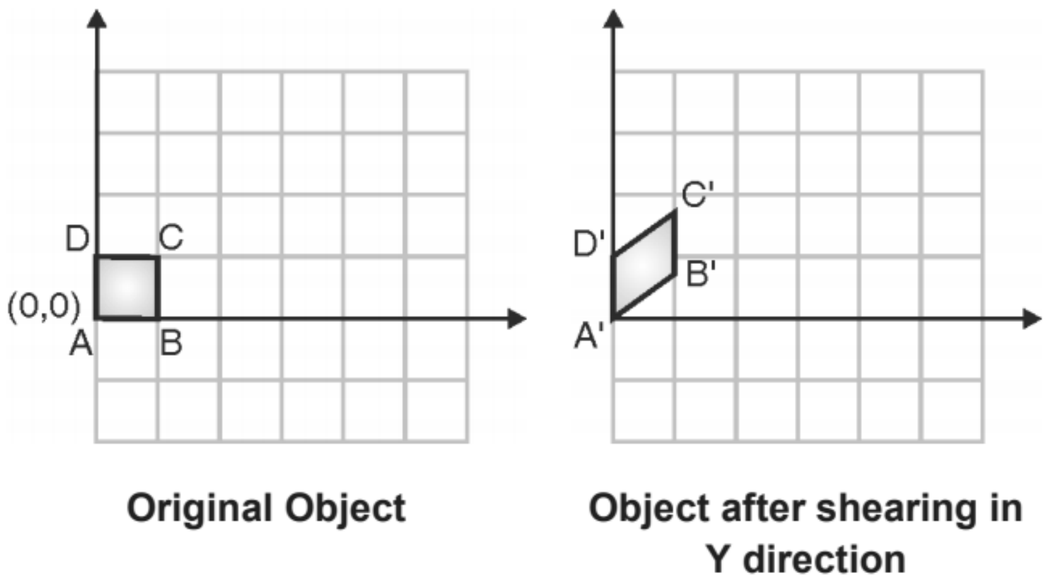


Fig. 5.10.2 : Shearing operation in Y- direction

Ex. 5.10.3 : Apply y-direction shearing on unit square with base on X = 0 line and one corner at the origin with $Sh_y = 1/2$.

Soln. :

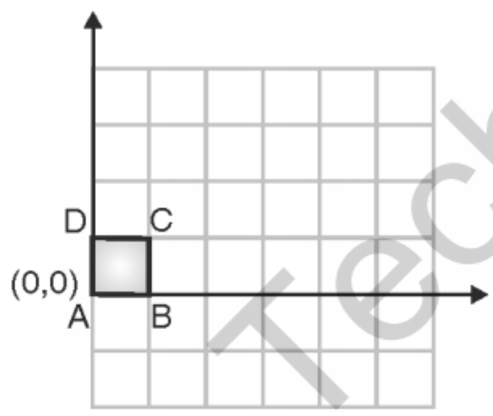


Fig. P. 5.10.3

Position of a unit cube is shown in the Fig. P. 5.10.3. Coordinates of vertices would be A(0, 0), B(1, 0), C(1, 1) and D(0, 1). For Y direction shear

$$x' = x$$
$$y' = y + x \cdot Sh_y$$

For point A, $y' = 0 + (0)1/2 = 0$

For point B, $y' = 0 + (1)1/2 = 1/2$

For point C, $y' = 1 + (1)1/2 = 3/2$

For point D, $y' = 1 + (0)1/2 = 1$

Position of sheared cube is shown in Fig. P. 5.10.3(a).

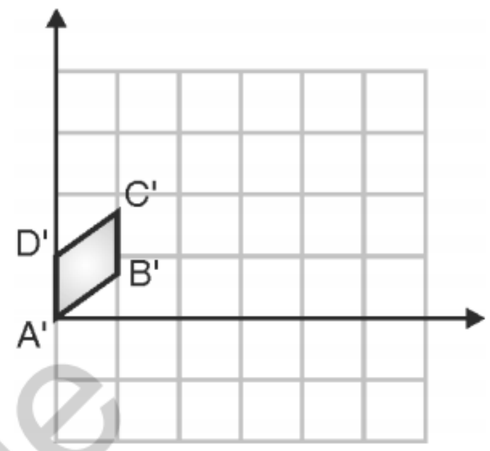


Fig. P. 5.10.3(a)

Original Coordinate	Transformed Coordinate
A(0, 0)	A′(0, 0)
B(0, 1)	B′(1, 1/2)
C(1, 1)	C′(1, 3/2)
D(1, 0)	D′(0, 1)

5.10.4 Shearing About a Line Parallel to Y- axis

We can derive generalized matrix for shearing about any reference line $X = X_{ref}$ parallel to Y-axis as follows :

1. Translate line by $-X_{ref}$ so the line will align with Y-axis.
2. Apply pure shear operation in the Y direction.
3. Translate line by $+X_{ref}$ to bring it back to the original place.

∴ The composite transformation matrix

$$M = T^{-1} \cdot Sh_y \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Technknowledge Publications

$$= \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{ref} \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

Y-direction shear relative to line $X = x_{ref}$ can be produced with transformation matrix :

$$M = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

Ex. 5.10.4 : Covert the unit square to shifted parallelogram using x-direction shear transformation operation where parameter sh_x = ½ and Y_{ref} = –1 and unit square dimensions are (0, 0), (1, 0), (0, 1) and (1, 1).

Soln. :

Let us consider the vertices of unit square as A(0, 0), B(1, 0), C(1, 1) and D(0, 1). Shearing parameters are $Sh_x = 1/2$ and $Y_{ref} = -1$.

Resultant co-ordinates are computed as,

$$P' = M \cdot P$$

$$M = \begin{bmatrix} 1 & Sh_x & -Y_{ref} \cdot Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\therefore \quad P' = \begin{bmatrix} 1 & \frac{1}{2} & \frac{-1}{2} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

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

Original Co-ordinates	Transformed Co-ordinates
A (0, 0)	A′ (– 1/2, 0)
B (1, 0)	B′ (1/2, 0)
C (1, 1)	C′ (1, 1)
D (0, 1)	D′ (0, 1)

Ex. 5.10.5 : Apply the shearing transformation to Square with A(0,0), B(1,0), C(1,1) and D(0,1) as given below

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

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

Soln. :

(a) Here, reference line is $Y_{ref} = 1$ and shear parameter is 0.5, so $Sh_x = 0.5$.

The transformation matrix for shearing with respect to reference line Y_{ref} is,

$$M_1 = \begin{bmatrix} 1 & Sh_x & -Y_{ref} \cdot Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates would be, $P' = M_1 \cdot P$

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

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

(b) Here, reference line is $X_{ref} = -1$, and shear parameter is 0.5, so $Sh_y = 0.5$

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -X_{ref} \cdot Sh_y \\ 0 & 0 & 1 \end{bmatrix}$$

Technknowledge Publications

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

$\therefore$ Transformed coordinates are,

$$P' = M_2 \cdot P$$

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

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

Original coordinates	After shearing in the X direction	After shearing in the Y direction
A (0, 0)	A′ (0.5, 0)	A′ (0, 0.5)
B (1, 0)	B′ (1.5, 0)	B′ (1, 1)
C (1, 1)	C′ (2, 1)	C′ (1, 2)
D (0, 1)	D′ (1, 1)	D′ (0, 1.5)

Ex. 5.10.6 : Write matrices in a homogenous coordinate system for the 2D Shear transformations.

SPPU - May 16, 2 Marks

Soln. :

Transformation matrix for shearing X direction,

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

Transformation matrix for shearing about line parallel to X-axis,

$$M = \begin{bmatrix} 1 & Sh_x & -y_{ref} \cdot Sh_x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix for shearing Y direction,

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

Transformation matrix for shearing about line parallel to Y-axis,

$$M = \begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

5.11 Reflection

- Almost all graphics packages provide functions for basic transformation operations like translation, scaling and rotation. Some packages also provide the facility of additional functions like reflection and shearing.
- **Reflection** is the transformation which produces a mirror image of an object about a given axis.
- Reflection is achieved by rotating an object by 180° around reference axis, perpendicular to the XY plane.
- Reflection does not alter the shape and size of the object, so this is a rigid body transformation.
- Let us discuss the various cases of reflection.

5.11.1 Reflection About X - Axis (Y = 0 Line)

- Reflection about X-axis is also known as a reflection about the Y = 0 line.
- Reflection about X-axis only flips y-coordinate, the x coordinate remains as it is. Point P(2, –2) becomes P′(2, 2) after reflection about X-axis.

Reflection about X-axis is depicted in Fig. 5.11.1.

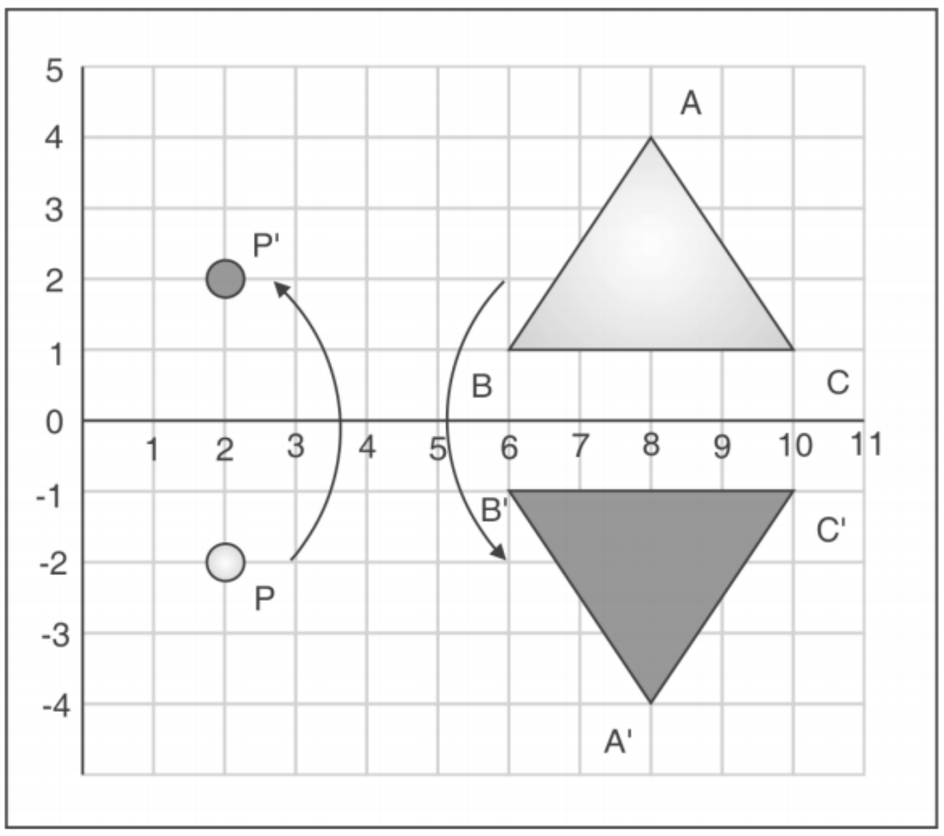


Fig. 5.11.1 : Reflection about the X-axis

– From Fig. 5.11.1, it is clear that,

$$\therefore x' = x$$
$$y' = -y$$

Transformation matrix for reflection about X axis is given as,

$$Ref_{(y=0)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.11.2 Reflection About Y - Axis (X = 0 Line)

– Reflection about Y axis is also known as reflection about X = 0 line. Point P(– 2, – 3) becomes P′(2, –3) after reflection about Y-axis.

– Reflection about Y-axis is shown in Fig. 5.11.2.

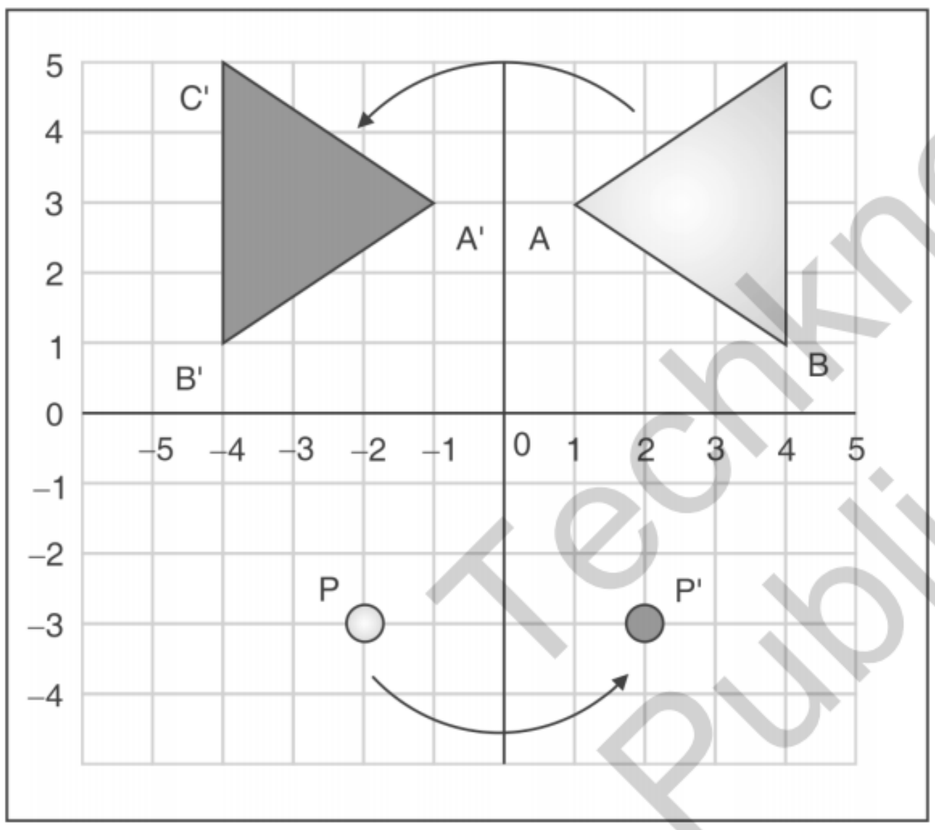


Fig. 5.11.2 : Reflection about the Y-axis

Reflection about Y-axis only flips x-coordinate, the y coordinate remains as it is.

$$\therefore x' = -x$$
$$y' = y$$

Transformation matrix for reflection about Y axis is given as,

$$Ref_{(x=0)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.11.3 Reflection About X = Y Axis

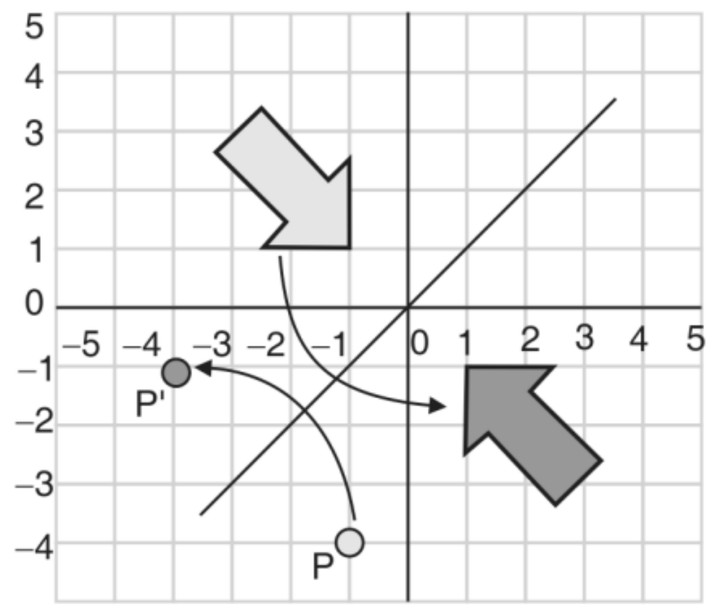


Fig. 5.11.3 : Reflection about X = Y axis

When a point (x, y) is reflected about a line X = Y, it becomes (y, x). As shown in Fig. 5.11.3, P(– 1, – 4) became P′ (– 4, – 1) after the reflection.

This operation swaps x and y coordinates.

$$\therefore x' = y$$
$$y' = x$$

Transformation matrix for reflection about Y = X line is given as,

$$Ref_{(x=y)} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.11.4 Reflection About X = – Y Axis

– When point (x, y) is reflected about a line X = – Y, it becomes (– y, – x). As shown in Fig. 5.11.4,

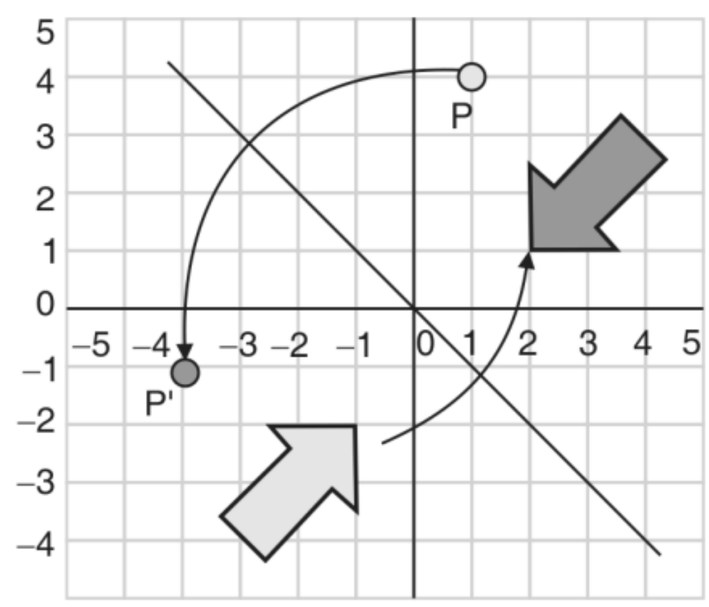


Fig. 5.11.4 : Reflection about X = – Y axis

– P(1, 4) became P′(– 4, – 1) after the reflection.

Tech Knowledge Publications

- This operation will swap x and y coordinates and also change their sign.

$$\therefore \; x' = -y$$
$$y' = -x$$

- Transformation matrix for reflection about Y = – X line is given as,

$$Ref_{(x=-y)} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Last both operations can be derived by aligning line Y = X or Y = – X with either of the principal axis and taking reflection about that particular axis followed by an inverse rotation.

5.11.5 Reflection About Origin

Rotation about the origin is carried out by flipping both, x and y coordinates.

This is 180° rotation about an axis that is perpendicular to the XY plane and that passes through the coordinate origin.

This operation flips both the coordinates.

$$\therefore \; x' = -x$$
$$y' = -y$$

The transformation matrix for reflection about the origin is given as,

$$Ref_{(origin)} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.11.6 Reflection About Any Line y = mx + c

- Reflection about any arbitrary line y = mx + c can be accomplished by the combination of translation, rotation and reflection.
- First, translate the line such that it passes through the origin.
- Then rotate the line so that it aligns with one of the principal axes.
- Perform reflection operation.
- Finally, restore the line to its actual position by performing inverse rotation and inverse translation.
- The composite transformation matrix for this sequence of operation is obtained by multiplying following matrices.

$$M = T^{-1} \cdot R^{-1} \cdot Ref \cdot R \cdot T$$

- We can implement reflections with respect to the coordinate axes or coordinate origin as scaling transformations with negative scaling factors.

5.11.7 Reflection About a Line Parallel to X-Axis

Reflection about a line parallel to the X axis is carried out by following steps

Step 1 : Translate the line such that it aligns with X-axis (Refer Fig. 5.11.5(b)).

Step 2 : Perform the reflection about X axis (Refer Fig. 5.11.5 (c)).

Step 3 : Translate back the line to its original position (Refer Fig. 5.11.5 (d)).

The sequence of operation is depicted in Fig. 5.11.5.

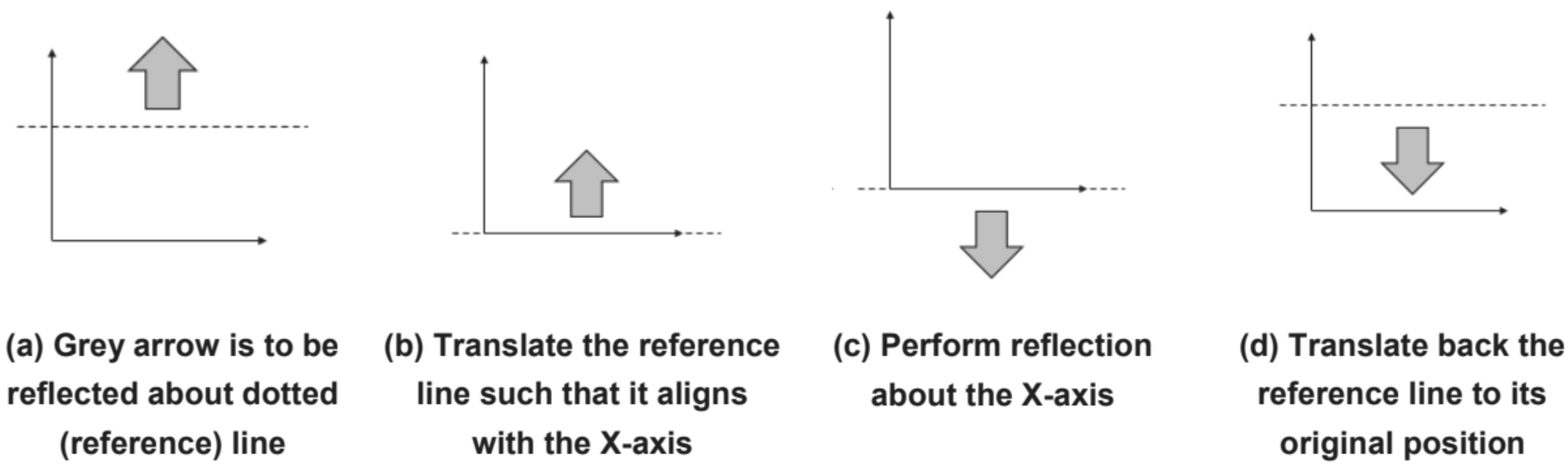


Fig. 5.11.5 : Sequence of steps to reflect object about a reference line parallel to X-axis

Techknowledge Publication

The transformation matrix for this operation is given by,

$$M = T^{-1} \cdot Ref_{(Y = 0)} \cdot T$$

5.11.8 Reflection About a Line Parallel to Y-Axis

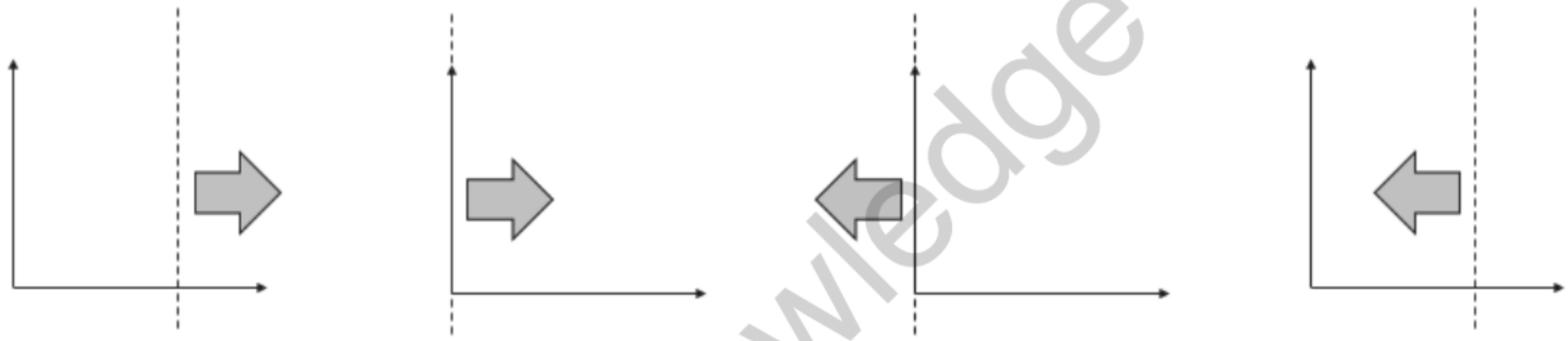
Reflection about a line parallel to the Y axis is carried out by the following steps :

Step 1 : Translate the line such that it aligns with Y-axis (Refer Fig. 5.11.6(b)).

Step 2 : Perform the reflection about the Y axis (Refer Fig. 5.11.6 (c)).

Step 3 : Translate back the line to its original position (Refer Fig. 5.11.6 (d)).

The sequence of operation is depicted in Fig. 5.11.6.



(a) Grey arrow is to be reflected about dotted (reference) line **(b) Translate the reference line such that it aligns with Y-axis** **(c) Perform reflection about the Y-axis** **(d) Translate back the reference line to its original position**

Fig. 5.11.6 : Sequence of steps to reflect object about a reference line parallel to the Y- axis

Transformation matrix for this operation is given by,

$$M = T^{-1} \cdot Ref_{(X = 0)} \cdot T$$

Ex. 5.11.1 : Show that transformation matrix for a reflection about line y = x is equivalent to reflection about X-axis followed by counter-clockwise rotation of 90°.

Soln. :

Let's say M_1 is the transformation matrix for a reflection about line y = x

$$\therefore M_1 = Ref_{(y = x)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

And M_2 is the transformation matrix for reflection about X-axis followed by counter-clockwise rotation by 90°.

$$\therefore M2 = R_{(\theta = 90°)} \cdot Ref_{(y = 0)}$$

$$= \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

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

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

Here, $M_1 = M_2$, hence given statement is proved.

Ex. 5.11.2 : Write 2 × 2 transformation matrix for each of the following rotation about the origin (a) Counter-clockwise rotation by 180^0 (b) Clockwise rotation by 90^0.

Soln. :

(a) 2 × 2 transformation matrix for counter-clockwise rotation by 180° about the origin.

Technical Knowledge Publications

$$M_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(180°) & -\sin(180°) & 0 \\ \sin(180°) & \cos(180°) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_1 = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) 2×2 transformation matrix for clockwise rotation by 90°,

So, $\theta = -90°$

$$M_2 = \begin{bmatrix} \cos(-90°) & -\sin(-90°) & 0 \\ \sin(-90°) & \cos(-90°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ex. 5.11.3 : Reflect a diamond shaped polygon whose vertices are A(– 1, 0), B(0, – 2), C(1, 0) and D(0, 2) about the line y = x + 2.

Soln. :

– Vertices of rectangle are A(– 1, 0), B(0, – 2), C(1, 0) and D(0, 2). Triangle is to be reflected about line y = x + 2.

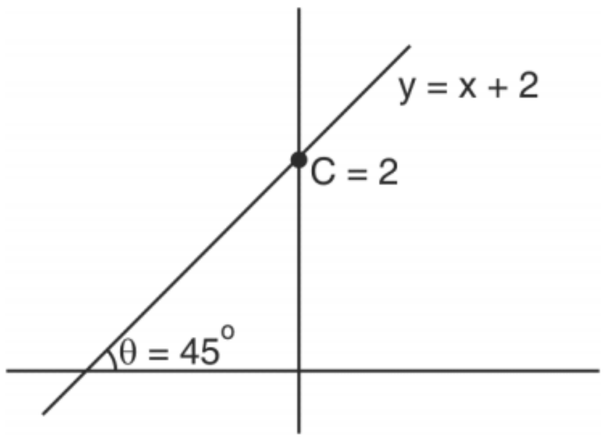


Fig. P. 5.11.3

Compare $y = x + 2$ with $y = mx + c$

$\therefore \tan\theta = m = 1$

$\theta = \tan^{-1}(1) = 45°$

and $c = 2$

– To perform said operation, we should carry out following steps.

1. Translate line by (0, – 2) so that it passes through the origin.
2. Rotate by 45° in a clockwise direction such that line gets align with the x axis.

   $\theta = -45°$
3. Reflection about the x-axis.
4. Inverse rotation.
5. Inverse translation.

$\therefore$ Transformation matrix would be,

$$M = T^{-1} \cdot R_{(\theta)}^{-1} \cdot Ref_{(y=0)} \cdot R_{(\theta)} T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

Put, $\theta = -45°$

$$\therefore M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

TechKnowledge Publications

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

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

Transformed coordinates are given by,

$$P' = M \cdot P$$

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

$$= \begin{bmatrix} -2 & -4 & -2 & 0 \\ 1 & 2 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original Co-ordinates	Transformed Co-ordinates
A (– 1, 0)	A′ (– 2, 1)
B (0, – 2)	B′ (– 4, 2)
C (1, 0)	C′ (– 2, 3)
D (0, 2)	D′ (0, 2)

5.12 Solved Examples

Ex. 5.12.1 : Translate a Square ABCD with the coordinates A(0, 0), B(5, 0), C(5, 5), D(0, 5) by 2 units in X-direction and 3 units in Y-direction.

Soln. :

Co-ordinates of square are A (0, 0), B (5, 0), C (5, 5) and D (0, 5). It is to be translated by 2 and 3 units in X and Y directions respectively.

So, $t_x = 2$ and $t_y = 3$

Transformed coordinates are computed as,

$$P' = M \cdot P$$

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

$$= \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 5 & 5 & 0 \\ 0 & 0 & 5 & 5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 2 & 7 & 7 & 2 \\ 3 & 3 & 8 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Original Co-ordinates	Transformed Co-ordinates
A (0, 0)	A′ (2, 3)
B (5, 0)	B′ (7, 3)
C (5, 5)	C′ (7, 8)
D (0, 5)	D′ (2, 8)

Ex. 5.12.2 : Find the sequence of transformation to scale the object with respect to point A in XY plane.

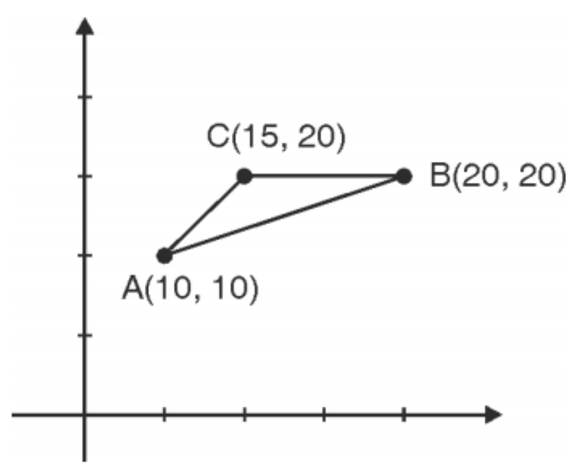


Fig. P. 5.12.2

TechKnowledge Publications

Soln. :

Here, an object is to be scaled with respect to point A. so reference point is A (10, 10). Scaling with respect to the reference point is achieved by the following steps.

1. Translate reference point to the origin.
2. Scale the object
3. Inverse translation of reference point

$\therefore$ Composite transformation $M = T^{-1} \cdot S \cdot T$

Consider S_x and S_y are the scaling parameters in X and Y direction respectively.

$$M = \begin{bmatrix} 1 & 0 & 10 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -10 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

This is the required sequence of transformations.

Ex. 5.12.3 : Consider the triangle with vertices (10, 10), (40, 10), (30, 30). Apply scaling transformation with scale factor 5 in X and Y direction. Draw the triangle before and after transformation.

Soln. :

Let's consider the triangle A (10, 10), B (40, 10), C (30, 30).

Scaling factors in X and Y direction is 5.

$\therefore \quad s_x = s_y = 5$

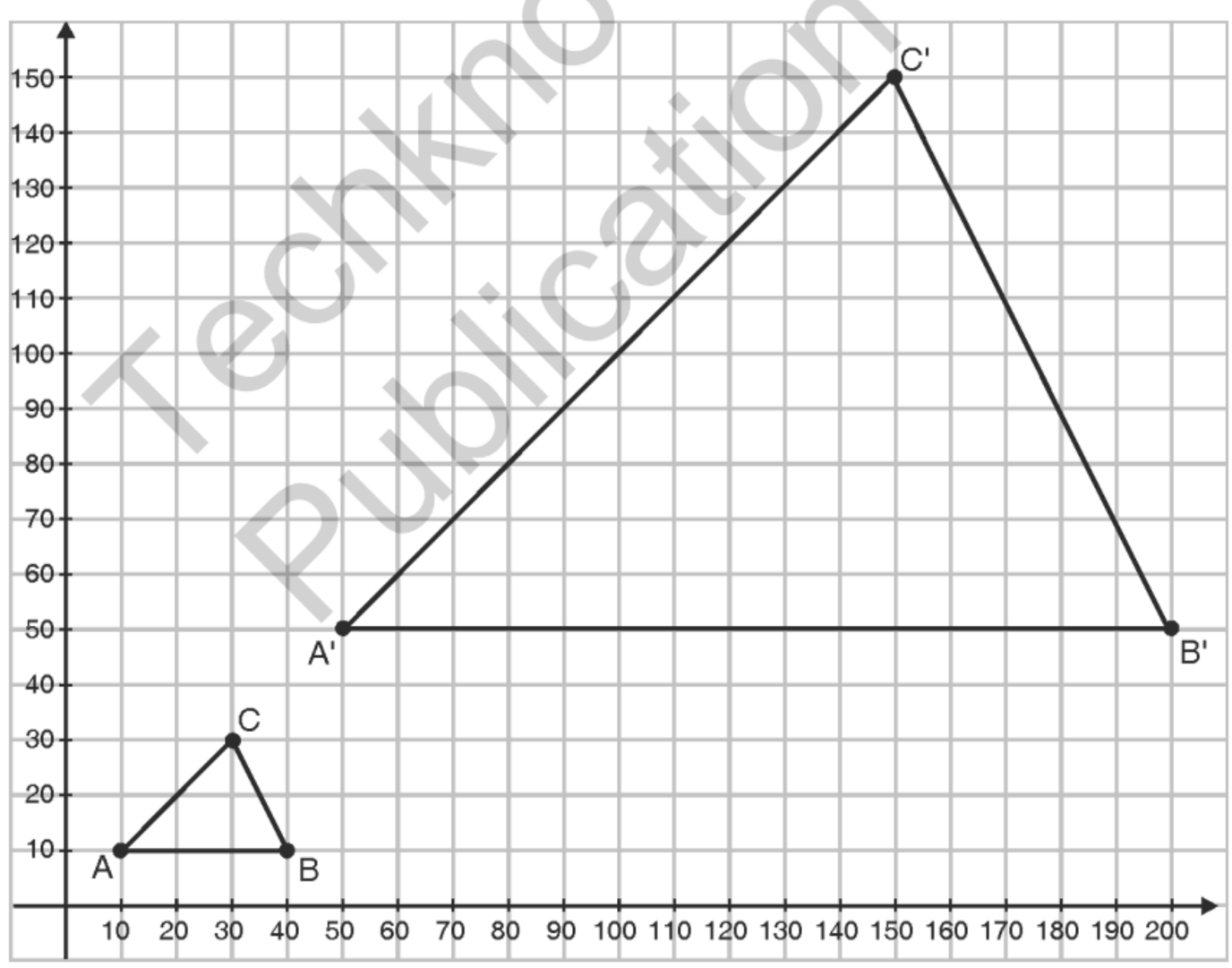


Fig. P. 5.12.3

Scaling operation is carried out as,

$$P' = S \cdot P$$

$$P' = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 40 & 30 \\ 10 & 10 & 30 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 50 & 200 & 150 \\ 50 & 50 & 150 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A (10, 10)	A′ (50, 50)
B (40, 10)	B′ (200, 50)
C (30, 30)	C′ (150, 150)

Ex. 5.12.4 : Rotate a triangle ABC by an angle 30° where the triangle has coordinates A(0, 0) B(10, 2) and C(7,4).

Soln. :

Triangle is to be rotated about the origin by the angle $\theta = 30°$. Let's assume P′ is the set of rotated coordinates.

$$\therefore \quad P' = M \cdot P$$

$$= R_{(\theta = 30)} \times P$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(30°) & -\sin(30°) & 0 \\ \sin(30°) & \cos(30°) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\sqrt{3}}{2} & -1/2 & 0 \\ 1/2 & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 10 & 7 \\ 0 & 2 & 4 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \frac{10\sqrt{3}-2}{2} & \frac{7\sqrt{3}-4}{2} \\ 0 & \frac{10+2\sqrt{3}}{2} & \frac{7+4\sqrt{3}}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

Ex. 5.12.5 : Derive 2 × 2 transformation matrix for following operations :

1. Counter-clockwise rotation by $\pi/2$.
2. Clockwise rotation by $-\pi$.
3. Reflection around X = 0 line followed by 180°counter-clockwise direction.
4. Uniform scaling with $S_x = 2$ followed by reflection about Y = – X line followed by 90°counter-clockwise direction.

Soln. :

1. Counter-clockwise rotation by $\pi/2$:

Here, rotation direction is counter-clockwise, so θ is positive.

$$\theta = \pi/2$$

Transformation matrix,

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos\pi/2 & -\sin\pi/2 \\ \sin\pi/2 & \cos\pi/2 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

2. Clockwise rotation by $-\pi$:

Here, rotation is in clockwise direction and rotation angle $\theta = -\pi$

Transformation matrix,

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

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

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

3. Reflection around X = 0 line followed by 180°counter-clockwise direction :

Reflection about X = 0 line flips only X coordinate, it does not alter Y coordinate. Hence, the transformation matrix for reflection around X = 0,

$$M_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Transformation matrix for rotation by 180° in counter-clockwise direction is

$$M_2 = \begin{bmatrix} \cos\pi & -\sin\pi \\ \sin\pi & \cos\pi \end{bmatrix}$$

Techknowledge Publication

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

$\therefore$ Resultant transformation matrix,

$$M = M_2 \cdot M_1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

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

4. Uniform scaling with $S_x = 2$ followed by reflection about Y = – X line followed by 90° counter-clockwise direction :

Composite transformation matrix for specified sequence of operation is derived as,

$$M = R_{(\theta = 90°)} \cdot Ref_{(Y = -X)} \cdot S$$

$$M = \begin{bmatrix} \cos(90°) & -\sin(90°) \\ \sin(90°) & \cos(90°) \end{bmatrix}$$

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

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

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

Ex. 5.12.6 : Derive transformation matrix :

1. To increase the area of square 4 times.
2. To increase the size by four times.

Soln. :

If we double the height and width of the square, its area becomes four times than original. So we need to perform scaling with $S_x = 2$ and $S_y = 2$.

$\therefore$ Resultant transformation matrix $M = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

To increase size four times, we have to multiply each side by 4.

$$\therefore \quad S_x = S_y = 4$$

$\therefore$ Resultant transformation matrix, $M = \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}$

Ex. 5.12.7 : 1. Find transformation matrix to rotate the object about the origin by 45° in the counter-clockwise direction.

2. Find new coordinates of the point (8,4) after rotation.

Soln. :

1. Here, rotation direction is counter-clockwise , so θ is positive.

$$\therefore \quad \theta = 45°$$

The transformation matrix is,

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} \cos(45°) & -\sin(45°) \\ \sin(45°) & \cos(45°) \end{bmatrix}$$

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

2. Find new coordinates of the point (8,4) after rotation.

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore \quad P' = M \cdot P = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 4 \end{bmatrix}$$

$$= \begin{bmatrix} 2\sqrt{2} \\ 6\sqrt{2} \end{bmatrix}$$

Ex. 5.12.8 : A triangle is defined by $\begin{bmatrix} 2 & 4 & 4 \\ 2 & 2 & 4 \end{bmatrix}$. Find transformed coordinates after the following transformation

1. 90° rotation about the origin.
2. Reflection about line X = Y.

Soln. :

1. 90° rotation about origin :

Here, $\theta = 90°$

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore P' = M \cdot P$$

$$= \begin{bmatrix} \cos 90° & -\sin 90° \\ \sin 90° & \cos 90° \end{bmatrix} \begin{bmatrix} 2 & 4 & 2 \\ 2 & 2 & 4 \end{bmatrix}$$

Technknowledge Publications

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

$$= \begin{bmatrix} -2 & -2 & -4 \\ 2 & 4 & 4 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(2, 2)	A′ (– 2, 2)
B(4, 2)	B′ (– 2, 4)
C(4, 4)	C′ (– 4, 4)

2. Reflection about line X = Y

Reflection about line X = Y

$$P'' = M \cdot P'$$

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

$$= \begin{bmatrix} 2 & 4 & 4 \\ -2 & -2 & -4 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(2, 2)	A′(2, – 2)
B(4, 2)	B′(4, – 2)
C(4, 4)	C′(4, – 4)

Ex. 5.12.9 : Perform 90° rotation of triangle with vertices A(2,2), B(4,2) and C(3,3)

1. About origin
2. About reference point (– 2,2)

Soln. :

1. Rotation about origin :

Here, rotation direction is counter-clockwise . So θ is positive.

$$\theta = 90^\circ$$

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore P' = M \cdot P$$

$$= \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ \\ \sin 90^\circ & \cos 90^\circ \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} -2 & -2 & -3 \\ 2 & 4 & 3 \end{bmatrix}$$

Original Coordinate	Transformed Coordinate
A(2, 2)	A′(– 2, 2)
B(4, 2)	B′(– 2, 4)
C(3, 3)	C′(– 3, 3)

2. About reference point (– 2, 2) :

The composite transformation matrix for rotation about some reference point is given as,

$$M = T^{-1} \cdot R \cdot T$$

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore P' = M \cdot P$$

$$= \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 6 & 5 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -2 & -3 \\ 6 & 8 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

Technknowledge Publication

Original Coordinates	Transformed Coordinates
A(2, 2)	A′ (– 2, 6)
B(4, 2)	B′ (– 2, 8)
C(3, 3)	C′ (– 3, 7)

Ex. 5.12.10 : Magnify a triangle with vertices A(0, 0), B(1, 1) and C(5, 2) to twice of its size keeping B fixed.

Soln. : This is fixed-point scaling with respect to reference point B(1, 1), so we need to perform following steps to carry out the desired operation.

1. Translate B to the origin.
2. Apply scaling with $S_x = S_y = 2$.
3. Inverse translation of B.

The composite transformation matrix for scaling with respect to some reference point is given as,

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

Let us assume P and P′ are original and transformed coordinates of the object respectively.

$$\therefore \; P' = M \cdot P$$

$$\therefore \; P' = T^{-1} \cdot S \cdot T \cdot P = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 5 \\ 0 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix}$$

Multiplying first two and last two matrices

$$= \begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 4 \\ -1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 9 \\ -1 & 1 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(0, 0)	A′(– 1, – 1)
B(1, 1)	B′(1, 1)
C(5, 2)	C′(9, 3)

Ex. 5.12.11 : Comment on statement: "Two dimensional rotation and scaling are cumulative, if $S_x = S_y$ or $\theta = n\pi$".

Soln. : Matrix for 2D rotation,

$$M_1 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

Matrix for 2D scaling,

$$M_2 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

We can say that operations are cumulative if $M_1M_2 = M_2M_1$.

(A) If $S_x = S_y$

Case 1 : Rotation followed by scaling

$$M_1M_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_x & 0 \\ 0 & 0 & 1 \end{bmatrix}(S_x = S_y) = \begin{bmatrix} S_x\cos\theta & -S_x\sin\theta & 0 \\ S_x\sin\theta & S_x\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Case 2 : Scaling followed by a rotation

$$M_2M_1 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} S_x\cos\theta & -S_x\sin\theta & 0 \\ S_x\sin\theta & S_x\cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here $M_1M_2 = M_2M_1$

So given statement is true for $S_x = S_y$.

(B) If $\theta = n\pi$

Case 1 : Scaling followed by rotation

$$M_1M_2 = \begin{bmatrix} \cos(n\pi) & -\sin(n\pi) & 0 \\ \sin(n\pi) & \cos(n\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$(\sin(n\pi) = 0, \cos(n\pi) = 1)$

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

Case 2 : Rotation followed by scaling

$$M_2M_1 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(n\pi) & -\sin(n\pi) & 0 \\ \sin(n\pi) & \cos(n\pi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

Here, $M_1M_2 = M_2M_1$, So given statement is true for $\theta = n\pi$

Techknowledge Publications

Ex. 5.12.12 : The reflection along line Y = X is equivalent to reflection along X-axis followed by counter-clockwise rotation by θ. Find the value of θ.

Soln. : Transformation matrix for reflection about Y = X line

$$M_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix for reflection about X-axis

$$M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transformation matrix for counter-clockwise rotation by angle θ

$$M_3 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

As per given sequence of operation

$$M_1 = M_3 M_2$$

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

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

$$\therefore \cos\theta = 0, \sin\theta = 1$$

$$\therefore \theta = 90^\circ$$

Ex. 5.12.13 : Show that $\begin{bmatrix} \frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2} \\ \frac{2t}{1+t^2} & \frac{1-t^2}{1+t^2} \end{bmatrix}$ represents pure rotation.

Soln. : For pure rotation, the determinant of the rotation matrix is one.

$$\text{Given matrix is } M = \begin{bmatrix} \frac{1-t^2}{1+t^2} & \frac{2t}{1+t^2} \\ \frac{2t}{1+t^2} & \frac{1-t^2}{1+t^2} \end{bmatrix}$$

The determinant of given matrix M is :

$$|M| = \frac{1-t^2}{1+t^2} \cdot \frac{1-t^2}{1+t^2} - \frac{2t}{1+t^2} \cdot \frac{-2t}{1+t^2}$$

$$= \frac{(1-t^2)^2 + 4t^2}{(1+t^2)^2} = \frac{(1+t^2)^2}{(1+t^2)^2} = 1$$

So the given matrix represents pure rotation.

Ex. 5.12.14 : Show that two-dimensional reflection through X-axis followed by two-dimensional reflection through line Y = – X is equivalent to a pure rotation about the origin by 270°.

Soln. :

Transformation matrix for two dimensional reflection through X-axis followed by two dimensional reflection through line Y = – X is,

$$M_1 = \text{Ref}(Y = -X) \cdot \text{Ref}(Y = 0)$$

$$M_1 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots(1)$$

Transformation matrix for rotation about origin is,

$$M_2 = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Here $\theta = 270^\circ$, so

$$M_2 = \begin{bmatrix} \cos(270^\circ) & -\sin(270^\circ) & 0 \\ \sin(270^\circ) & \cos(270^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \ldots(2)$$

Here $M_1 = M_2$, so statement is proved.

Note : In general, two pure reflections about a line passing through origin is equivalent to the purerotation.

Tech Knowledge Publications

Ex. 5.12.15 : A mirror is placed vertically such that passes through points (10,0) and (0,10). Find the mirrored coordinate of a triangle with vertices A (2,2) B(4,2) and C(3,3).

Soln. :

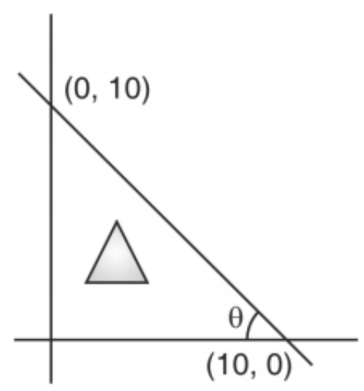


Fig. P. 5.12.15

Arrangement of object and line is shown in Fig. P. 5.12.15.

From Fig. P. 5.12.15,

$$\tan\theta = \frac{dy}{dx} = \frac{10}{10} = 1,$$

$$\text{so, } \theta = \tan^{-1}(1) = 45^\circ$$

To reflect the given triangle about given line, we have to align it with one of the four line, x = 0, y = 0, x = y or x = – y.

Let's align it with X-axis. To align it with X-axis we should perform the following steps :

1. Translate (0,10) to the origin.
2. Perform anticlockwise rotation by $\theta = 45^\circ$
3. Perform reflection about X-axis.
4. Inverse rotation.
5. Inverse translation.

So the sequence of a resultant transformation matrix is,

$$M = T^{-1} \cdot R^{-1} \cdot Ref_{(Y=0)} \cdot R \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 45^\circ & \sin 45^\circ & 0 \\ \sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(-45^\circ) & \sin(-45^\circ) & 0 \\ \sin(-45^\circ) & \cos(-45^\circ) & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -1/\sqrt{2} & 1/\sqrt{2} & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} & 0 \\ -1/\sqrt{2} & -1/\sqrt{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

$$= \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -10 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 10 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates of the triangle are,

$$P' = M \cdot P = \begin{bmatrix} 0 & -1 & 10 \\ -1 & 0 & 10 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 2 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 8 & 7 \\ 8 & 6 & 7 \\ 1 & 1 & 1 \end{bmatrix}$$

Reflected coordinates of the triangle are A′(8, 8), B′ (8, 6) and C′(7, 7)

Ex. 5.12.16 : Reflect triangle ABC with vertices A(2,2), B(4,2) and C(3,3) around line – 3x + 4y – 8 = 0.

Soln. :

From line equation – 3x + 4y – 8 = 0

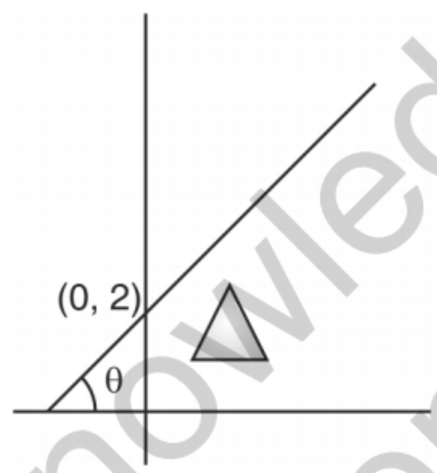


Fig. P. 5.12.16

$$4y = 3x + 8$$

$$\therefore \; y = \frac{3x}{4} + 2 = mx + c$$

$$\therefore \; m = \frac{3}{4}, \; c = 2$$

$$\tan\theta = \frac{3}{4} \; ; \quad \therefore \; \sin\theta = \frac{3}{5} \; ; \quad \cos\theta = \frac{4}{5}$$

Line makes a positive angle with X axis (because tan θ > 0) and it intersects Y axis at point (0, 2) (c = 2). Line orientation is shown in Fig. P. 5.12.16.

The transformation matrix for reflection is about the given line is achieved by the following steps:

1. Translate (0,2) to the origin.
2. Rotate line by $\theta = \tan^{-1}\left(\frac{3}{4}\right)$ angle in a clockwise direction.
3. Reflect triangle about X-axis.
4. Inverse rotation by θ.
5. Inverse translation.

$$\therefore \quad M = T^{-1} \cdot R^{-1} \cdot Ref_{(Y=0)} \cdot R \cdot T$$

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & -3/5 & 0 \\ 3/5 & 4/5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & 3/5 & 0 \\ -3/5 & 4/5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4/5 & -3/5 & 0 \\ 3/5 & 4/5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & 3/5 & -6/5 \\ -3/5 & 4/5 & -8/5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4/5 & 3/5 & 0 \\ 3/5 & -4/5 & 2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4/5 & 3/5 & -6/5 \\ -3/5 & 4/5 & -8/5 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 7/25 & 24/25 & -48/25 \\ 24/25 & -7/25 & 64/25 \\ 0 & 0 & 1 \end{bmatrix}$$

Coordinates of reflected triangle is given by,

$$P' = M \cdot P = \begin{bmatrix} 7/25 & 24/25 & -48/25 \\ 24/25 & -7/25 & 64/25 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 3 \\ 4 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \frac{62}{25} & \frac{28}{25} & \frac{45}{25} \\ \frac{84}{25} & \frac{146}{25} & \frac{115}{25} \\ 1 & 1 & 1 \end{bmatrix}$$

$$\therefore \quad A' = \left(\frac{62}{25}, \frac{84}{25}\right),$$

$$B' = \left(\frac{28}{25}, \frac{146}{25}\right),$$

$$C' = \left(\frac{45}{25}, \frac{115}{25}\right)$$

Original Coordinates	Transformed Coordinates
A(2, 4)	$A'\left(\frac{62}{25}, \frac{84}{25}\right)$
B(4, 2)	$B'\left(\frac{28}{25}, \frac{146}{25}\right)$
C(3, 3)	$C'\left(\frac{45}{25}, \frac{115}{25}\right)$

Tech Knowledge Publications

Review Questions

Q. 1 What is a transformation ?

Q. 2 List out basic transformations techniques?

Q. 3 Explain 2D Transformation. Derive transformation matrix for the same.

Q. 4 Derive the suitable transformation matrix for translation.

Q. 5 Explain scaling transformation with respect to 2D.

Q. 6 What is scaling ? Derive 2D transformation matrix for scaling with respect to origin and a reference point

Q. 7 Derive the transformation matrix for scaling with respect to origin.

Q. 8 Derive the transformation matrix for scaling with respect to reference point.

Q. 9 Explain uniform and non-uniform scaling with suitable example.

Q. 10 Derive the transformation matrices for rotation with respect to origin and reference point.

Q. 11 Derive 2D transformation matrix for rotation with respect to origin.

Q. 12 What do you mean by pure rotation matrix?

Q. 13 Derive 2D transformation matrix for rotation with respect to reference point.

Q. 14 Explain in brief : Homogeneous coordinate.

Q. 15 What is the need of homogenous coordinate in transformation?

Q. 16 List the properties of the homogeneous coordinate representation.

Q. 17 Give homogenous coordinate matrix representation for translation, rotation and scaling.

Q. 18 Give matrix representation for 2D translation.

Q.19 Give the homogeneous representation for translation, scaling and rotation.

Q. 20 Give matrix representation for 2D scaling.

Q. 21 Give matrix representation for 2D rotation.

Q. 22 What is composite transformation?

Q. 23 Find out the composite transformation matrix to rotate a given 2D object by an amount θ about the given point (xr, yr).

Q. 24 Derive transformation matrix for general pivot point rotation.

Q. 25 Write a short note on : General pivot point rotation.

Q. 26 Derive transformation matrix for general pivot point scaling.

Q. 27 Write a short note on : General pivot point scaling.

Q. 28 What is reflection? Derive transformation matrices for various cases of reflection.

Q.29 Derive transformation matrix for reflection about X axis.

Q. 30 Write a short note on : Reflection about X axis.

Q. 31 Derive transformation matrix for reflection about Y axis.

Q. 32 Write a short note on : Reflection about Y axis.

Q. 33 Derive transformation matrix for reflection about X = Y axis.

Q. 34 Write a short note on : Reflection about X = Y axis.

Q. 35 Derive transformation matrix for reflection about X = – Y axis.

Q. 36 Write a short note on : Reflection about X = –Y axis.

Q. 37 Derive transformation matrix for reflection about origin.

Q. 38 Write a short note on Reflection about origin.

Q. 39 Derive transformation matrix for reflection about any arbitrary line.

Q. 40 How to perform reflection about a line y = mx + c.

Q. 41 Derive transformation matrices for shearing in X and Y direction.

Q. 42 How to perform shearing in X direction? Derive its transformation matrix.

Q. 43 Derive transformation matrix for shearing with respect to line parallel to X-axis.

Q. 44 How to perform shearing in Y direction? Derive its transformation matrix.

Q. 45 Derive transformation matrix for shearing with respect to line parallel to Y-axis.

Technknowledge Publication

❑❑❑

CHAPTER 6

UNIT III

3D Transformation and Projections

Syllabus

3-D transformations : introduction, 3-D transformations - Translation, scaling, rotation and shear, rotation about an arbitrary axis.

Projections : Parallel (Oblique : Cavalier, Cabinet and orthographic : isometric, diametric, trimetric) and Perspective (Vanishing Points – 1 point, 2 point and 3 point).

6.1 3D Transformation

6.1.1 Introduction

- 3D transformation is an extension of 2D transformation in space.
- In 3D transformation, transformation operations like translation, rotation, scaling etc. take place in space rather than on a plane.
- Like 2D transformation, we will make use of homogenous representation to describe the 3D transformation matrices.

6.1.2 3D Geometry

- In 2D geometry, the point is described by (x, y) coordinates in XY plane, whereas in 3D geometry point (x, y, z) is defined in space with added third dimension z.
- 2D geometry system is described by the intersection of two planes which are perpendicular to each other.
- 3D geometry system is described by the intersection of three planes which are perpendicular to each other.
- Intersecting lines form the axis and the intersection point of the plane forms the origin of the geometry system.
- 2D and 3D geometry system are depicted in Fig. 6.1.1.

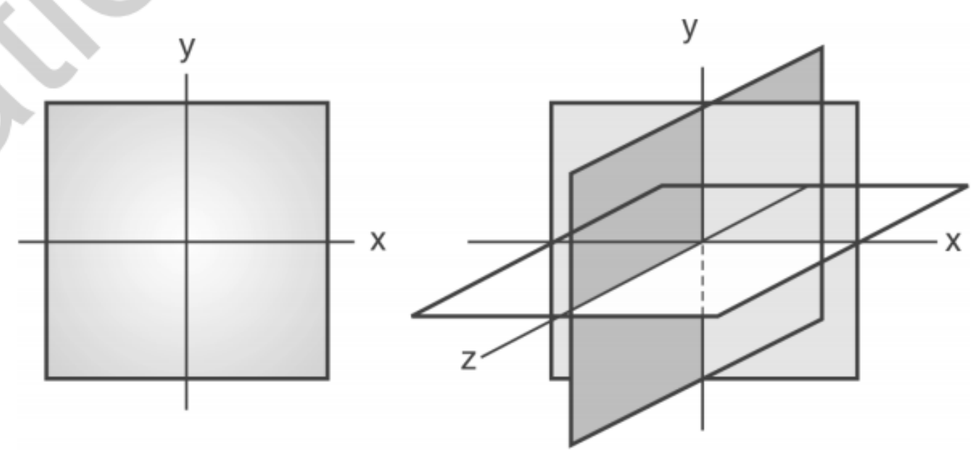


Fig. 6.1.1 : 2D geometry system (left) and 3D geometry system (right)

- 2D geometry divides the coordinate system into four quadrants. 3D geometry divides the coordinate system into eight octants.
- The distance of point is measured from the origin.

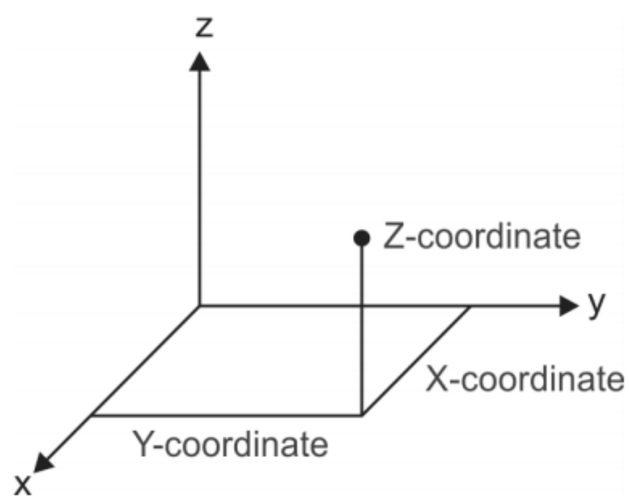


Fig. 6.1.2

Technknowledge Publication

– Distance between two points $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$ is computed by the following equation :

$$d(A, B) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

– When the point is projected on any plane using a perpendicular projection, the third coordinate becomes zero. Any point on the XY plane will be (x, y, 0) and any point on X-axis will be (x, 0, 0). Similar observations can be made for other planes and axes.

6.1.3 Translation

Q. Write transformation matrices for 3D translation.

SPPU - Dec. 18, 2 Marks

– The **three-dimensional translation** is a process of moving an object from one location to another location along the *straight path* in *space*.

– The translation is achieved by adding the required amount of shift in each direction.

– Amount of translation is specified by *translation vector* or *shift vector*, $T = [t_x, t_y, t_z]$.

– Let us consider the original point P (x, y, z), which becomes P′ (x′, y′, z′) after translation. Translated coordinates are computed by following equations :

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

– Equivalent homogeneous matrix representation is :

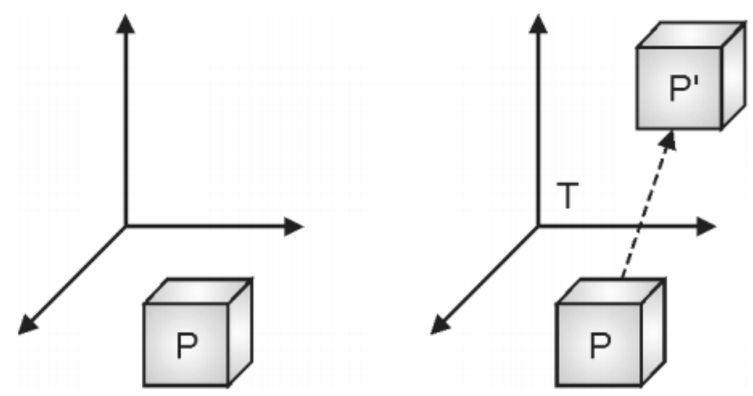
$$P' = T \cdot P$$

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

Inverse Translation Matrix

We can achieve inverse translation by negating the shift parameters. Inverse translation matrix is given as,

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



(a) Original object P **(b) Translated object P'**

Fig. 6.1.3 : Translation of object in space

Ex. 6.1.1 : Translate a triangle with vertices A (2, 2, 2), B (3, 4, 7) and C (8, 9, 12) by translation vector T [2, 4, 5].

Soln. :

Here translation vector T = [2, 4, 5]. Let us consider P = [A, B, C] is a set of original points and P′ = [A′, B′, C′] is the set of transformed coordinates. Transformed coordinates are computed as,

$$P' = T \cdot P$$

$$= \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 5 & 10 \\ 6 & 8 & 13 \\ 7 & 12 & 17 \\ 1 & 1 & 1 \end{bmatrix}$$

Translated vertices are A′ (4, 6, 9), B′ (5, 8, 12) and C′ (10, 13, 17).

Original Coordinates	Transformed Coordinates
A(2, 2, 4)	A′(4, 6, 7)
B(3, 4, 7)	B′(5, 8, 12)
C(8, 9, 12)	C′(10, 13, 17)

TechKnowledge Publications

6.1.4 Scaling

Q. Write matrices in the homogenous coordinate system for the following transformation: 3D Scaling. **SPPU - May 16, 2 Marks**

Q. Write transformation matrices for

(i) 3D rotation about Y-axis

(ii) 3D scaling **SPPU - Dec. 18, 6 Marks**

3D scaling is performed by multiplying all coordinates of the object by some constants called scaling parameters $S = [S_x, S_y, S_z]$. Scaling alters the shape and size of the object; hence it is not rigid-body transformation.

Scaling can be performed in either of the following way :

1. Scaling with respect to origin
2. Scaling with respect to reference point

6.1.4.1 Scaling with Respect to Origin

Let, $S = [S_x, S_y, S_z]$ be a vector of the scaling parameters in all three directions, scaling with respect to the origin of point P is computed as,

$$P' = S \cdot P$$

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

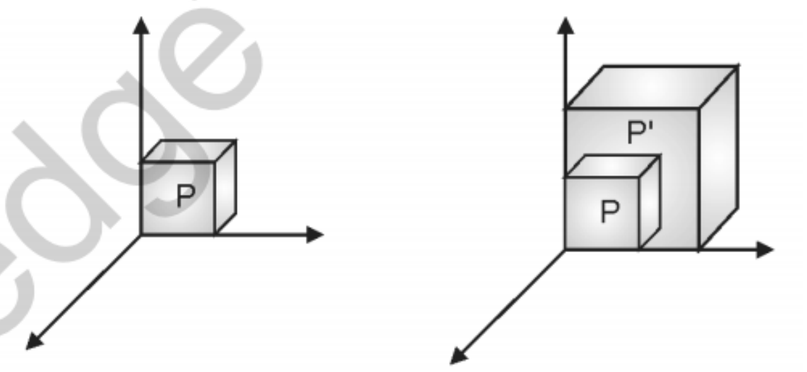
$$z' = S_z \cdot z$$

Matrix Representation of Scaling Transformation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

If all three parameters are the same, it is called **uniform scaling**, which preserves the original shape of the object to be scaled. Otherwise, it is called **non-uniform scaling**.

Fig. 6.1.4 shows the scaling operation with respect to the origin.



(a) Object P to be scaled (b) P' is the scaled object

Fig. 6.1.4 : 3D scaling with respect to the origin

6.1.4.2 Scaling with Respect to Reference Point

Scaling with respect to some reference point (x_r, y_r, z_r) is carried out by following sequence of the steps :

- Translate reference point to the origin.
- Apply scaling transformation with respect to the origin.
- Perform inverse translation to move the reference point back to the original position.

- The sequence of operations to find transformed coordinates is given as,

$$P' = T^{-1} \cdot S \cdot T \cdot P,$$

more specifically

$$P' = T(x_r, y_r, z_r) \cdot S(S_x, S_y, S_z) \cdot T(-x_r, -y_r, -z_r) \cdot P$$

Matrix representation of the operation would be,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & 1 & 0 & -y_r \\ 0 & 0 & 1 & -z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Techknowledge Publication

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & (1-S_x)\,x_r \\ 0 & S_y & 0 & (1-S_y)\,y_r \\ 0 & 0 & S_z & (1-S_z)\,z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Inverse scaling is achieved by replacing scaling parameters with their reciprocal.

Fig. 6.1.5 depicts the scaling operation with respect to the reference point.

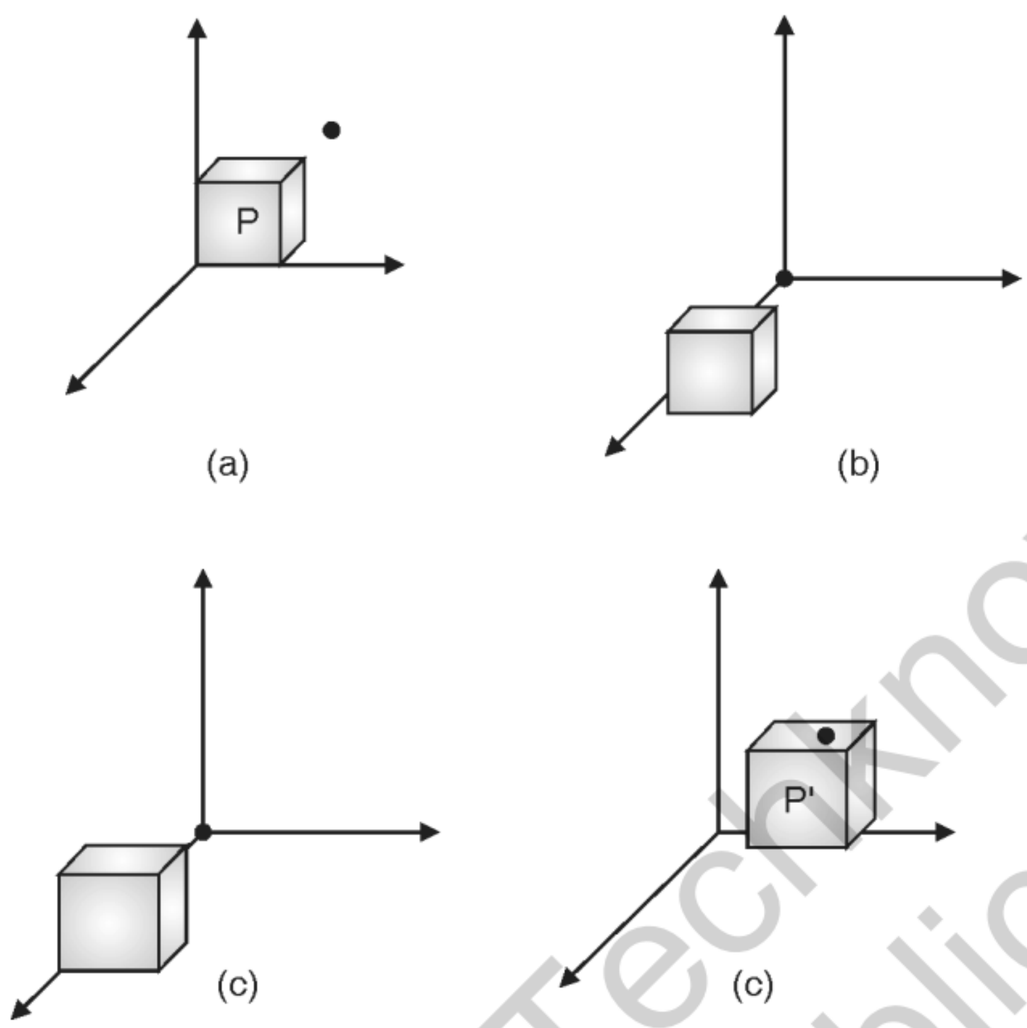


Fig. 6.1.5 : 3D scaling with respect to a reference point

6.1.5 Rotation

- In 3D, rotation is possible in infinite ways. We can rotate the object around three principal axes, or any arbitrary line.
- Here, we will discuss all possible cases of rotation in space.

Note : Like 2D rotation, for anticlockwise rotation, an angle is considered positive, and for clockwise rotation, angle is considered negative

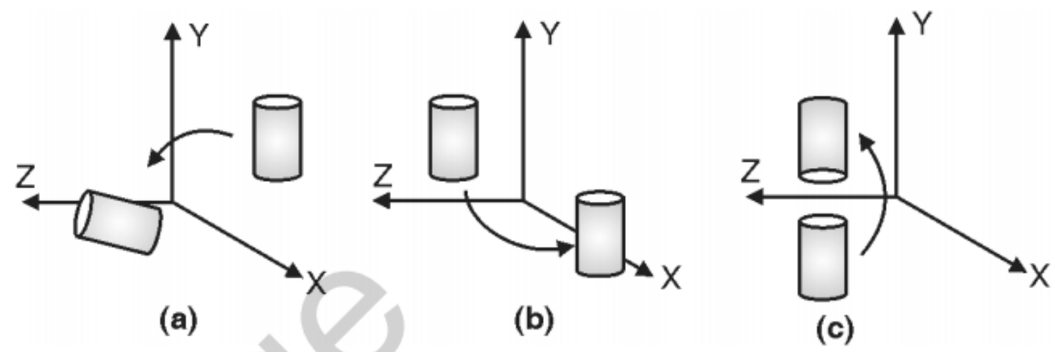
6.1.5.1 Rotation About Principal Axis

Q. Write matrices in a homogenous coordinate system for the following transformation: 3D rotation with respect to Y-axis. **SPPU - May 16, 2 Marks**

Q. Write transformation matrices for 3D rotation about Y-axis. **SPPU - Dec. 18, 2 Marks**

Q. Write transformation matrices for 3D rotation about X-axis. **SPPU - May 19, 2 Marks**

3D rotation is simply an extension of 2D rotation with added dimension. Fig. 6.1.6 depicts the rotation about X, Y and Z-axis.



(a) Rotation about X-axis (b) Rotation about the Y-axis (c) Rotation about Z-axis

Fig. 6.1.6 : 3D rotation about principal axes

Rotation about X-axis

- Rotation of object about X-axis does not alter the X-coordinate. It only affects Y and Z-coordinates. Rotation about X axis is given by,

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

- Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_x(\theta)\cdot P$$

$$\text{Where,} \quad R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Y-axis

- Similarly, rotation of an object about Y-axis does not alter the Y-coordinate. It only affects Z and X-coordinates. Rotation about Y-axis is given by,

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

Techknowledge Publication

– Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_y(\theta) \cdot P$$

$$\text{Where, } R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about Z-axis

– Similarly, rotation of object about Z-axis does not alter the Z-coordinate. It only affects X and Y coordinates. Rotation about Z-axis is given by,

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

– Matrix representation of above equation using homogeneous coordinate is given as follow :

$$P' = R_z(\theta) \cdot P$$

$$\text{Where, } R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverse Rotation Matrix

– Inverse rotation is achieved by negating the rotation angle θ. The transformation matrix for inverse rotation about Z-axis is shown below :

$$R_z(-\theta) = \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) & 0 & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– Cosine is even function and sine is an odd function, hence;

$$\text{Cos}(-\theta) = \cos\theta \text{ and } \sin(-\theta) = -\sin\theta$$

$$\text{Hence, } R_z(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, we can achieve inverse rotation about X and Y-axis by replacing θ by –θ in rotation matrix $R_x(\theta)$ and $R_y(\theta)$, respectively.

6.1.5.2 Rotation About Line Parallel to Principle Axis

– If the rotation axis is not the principal axis, we can make it coincide with one of the principal axes using composite transformation.

– This is done by performing a necessary series of translation and rotation operations.

– If the object is to be rotated about a line parallel to one of the principal axes, we can get the desired rotation with the following steps :

- Perform translation such that the rotation axis coincides with one of the principal axes.
- Perform specified rotation about that principal axis.
- Perform inverse translation to move the rotation axis to its original location.

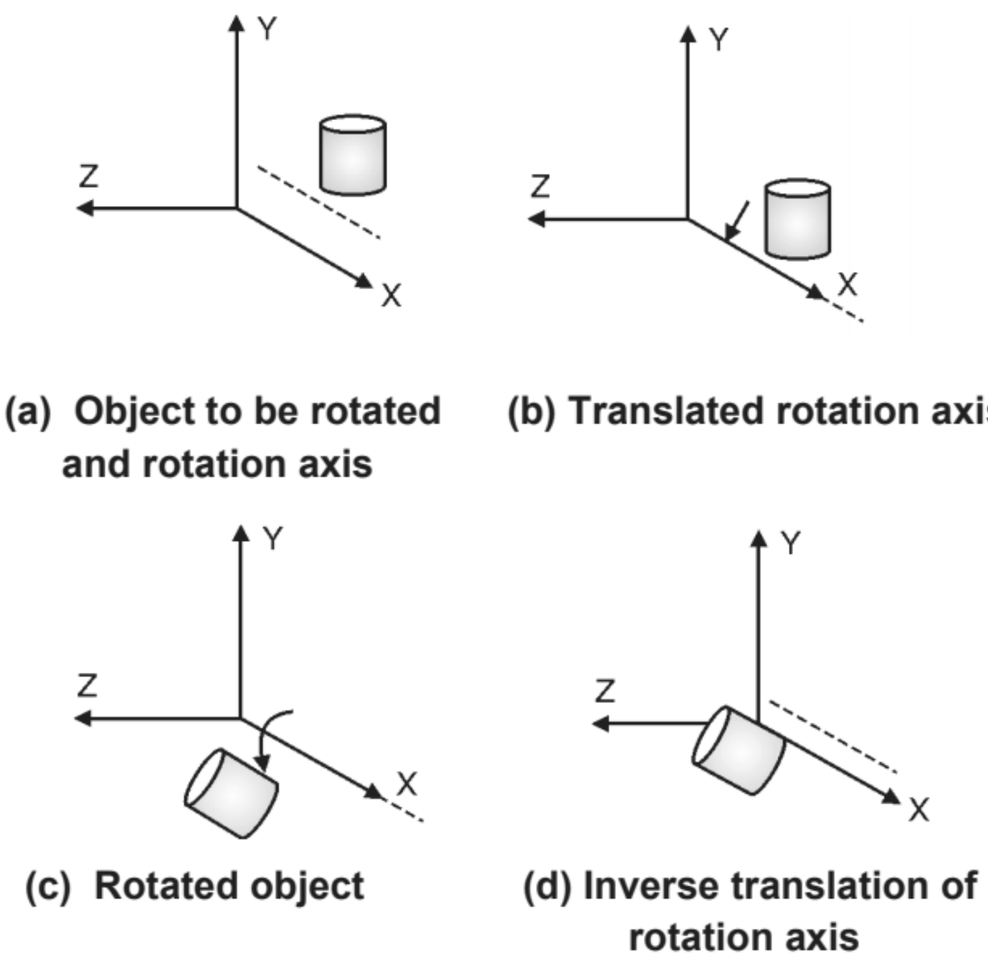
– Above sequence of steps can be written as :

$$P' = T^{-1} \cdot R_\lambda(\theta) \cdot T \cdot P$$

Where $R_\lambda(\theta)$ indicates the rotation about the principal axis with which rotation axis was aligned. The composite transformation matrix is :

$$M = R(\theta) = T^{-1} \cdot R_\lambda(\theta) \cdot T$$

– This operation is the same as 2D rotation about a pivot point in the plane. Fig. 6.1.7 depicts the rotation about a line parallel to X-axis. The dashed line indicates the rotation axis.



(a) Object to be rotated and rotation axis (b) Translated rotation axis

(c) Rotated object (d) Inverse translation of rotation axis

Fig. 6.1.7 : Rotation about a line parallel to X-axis

Techknowledge Publications

– The same concept can be extended for rotation about a line parallel to Y and Z-axis.

6.1.5.3 Rotation About an Arbitrary Line

Q. Explain how to perform rotation about an arbitrary axis in 3D with a diagram.

SPPU - May 17, 4 Marks

Q. Explain briefly rotation about an arbitrary axis in 3D. **SPPU - May 18, Dec. 19, 6 Marks**

– In real-world applications, rotation about the principal axis is a rare case. Application programmer needs to rotate the object around any arbitrary axis in space.

– When the object is to be rotated around a line which is not parallel to the principal axis, it requires more efforts. This case is more complicated than the previous cases.

– Let the rotation axis passes from the points P_1P_2 as shown in Fig. 6.1.8 (a). Desired rotation is achieved by the composite transformation. Following steps should be performed for that :

Step 1 : Translate the rotation axis such that it passes through the origin (Refer Fig. 6.1.8(b)).

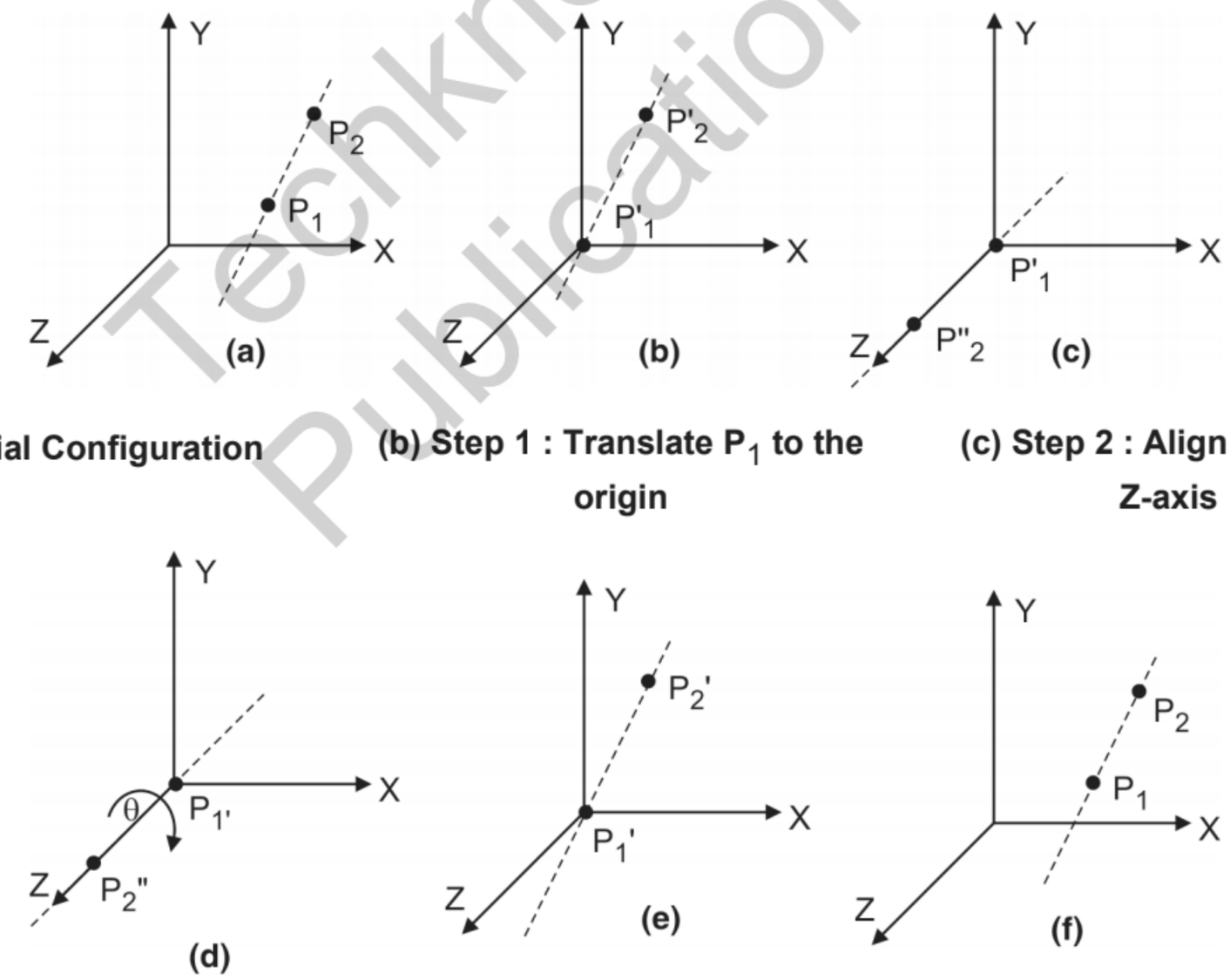
Step 2 : Rotate the translation axis such that it coincides one of the principal axes (Refer Fig. 6.1.8(c)).

Step 3 : Perform the actual rotation operation (Refer Fig. 6.1.8(d)).

Step 4 : Perform inverse rotation to bring rotation axis to the original orientation (Refer Fig. 6.1.8(e)).

Step 5 : Perform inverse translation to move back the rotation axis to its original location (Refer Fig. 6.1.8(f)).

– The sequence of the operations to perform the desired rotation is described in Fig. 6.1.8.



(a) Initial Configuration (b) Step 1 : Translate P_1 to the origin (c) Step 2 : Align $P_1'P_2'$ to Z-axis

(d) Step 3 : Rotation about Z-axis by given angle θ (e) Step 4 : Rotate the axis to the original orientation (f) Step 5 : Translate the axis to the original location

Fig. 6.1.8 : Rotation about an arbitrary line

Technknowledge Publications

– We can align the rotation axis with any of the principal axes. For the sake of discussion, we will align it with the Z-axis.

Let $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$ be the endpoints of the rotation axis depicted in Fig. 6.1.9.

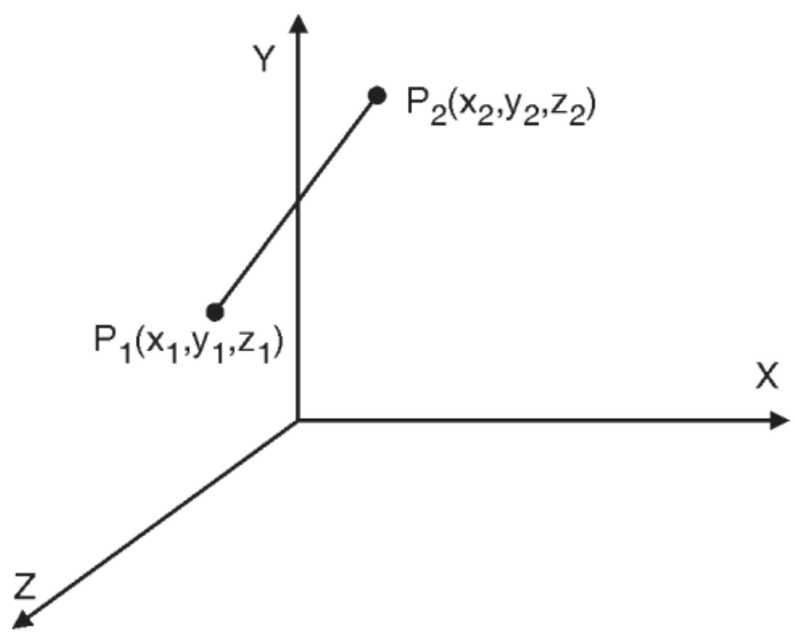


Fig. 6.1.9 : Orientation of rotation axis P_1P_2

Direction of rotation axis P_1P_2 is given by :

$$\overrightarrow{P_1P_2} = P_2 - P_1 = V$$

$$= (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

Unit vector in direction of rotation axis is defined as :

$$u = \frac{V}{|V|}$$

$$= \frac{x_2 - x_1, y_2 - y_1, z_2 - z_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}}$$

$$= (a, b, c)$$

Where, $a = \frac{x_2 - x_1}{|V|}$, $b = \frac{y_2 - y_1}{|V|}$, $c = \frac{z_2 - z_1}{|V|}$

and, $|V| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$

– If the rotation axis is not passing through the origin, we should translate any of the end points (P_1 or P_2) to the origin. Omit this step if rotation axis is already passing through the origin.

– For the given case, let us translate the point P_1 to the origin. This is done by applying the following translation matrix to the endpoints of the rotation axis.

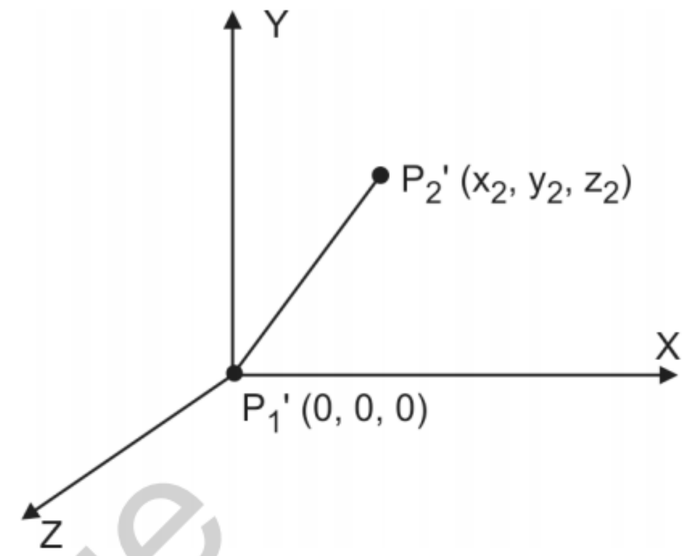


Fig. 6.1.10 : Position of rotation axis after translation by point P_1

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– This transformation moves the rotation axis as shown in Fig. 6.1.10, P_1' and P_2' are the translated coordinates of the original endpoints. Length of the rotation axis is already normalized to simplify the mathematical computation. So, normalized coordinates of u would be (a, b, c) as computed before.

– Now align the rotation axis with the Z-axis, to do so rotate the vector u around the X-axis in anticlockwise direction by the angle α such that unit vector falls in XZ plane.

– Then rotate unit vector in XZ plane around the Y-axis in a clockwise direction by the angle β to align it with the Z-axis.

– To align the vector u with Z-axis, first put the torch on X-axis such that the shadow of vector u falls in YZ plane, call the shadow u_1.

– As this would be parallel projection on the YZ plane, x coordinate will become 0 and the remaining two coordinates will not alter.

Techknowledge Publication

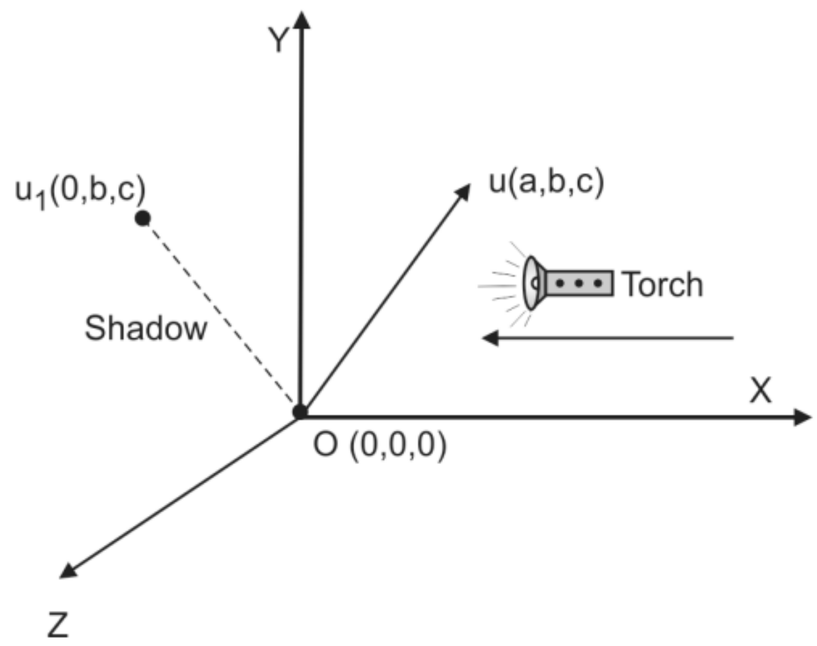


Fig. 6.1.11 : Shadow of vector u on YZ plane

– As shown in Fig. 6.1.11, parallel projection of vector u on YZ plane would be u′(0, b, c). Parallel projection on the YZ plane only eliminates the x coordinate; it does not affect remaining coordinates. It is easier to find out an angle α with projected vector u_1, compared to u.

– As shown in Fig. 6.1.12 Vector u_1 makes an angle α with the Z-axis. If we rotate u and u_1 by the angle α around X-axis, u′ will get align with Z-axis and u will fall in XZ plane.

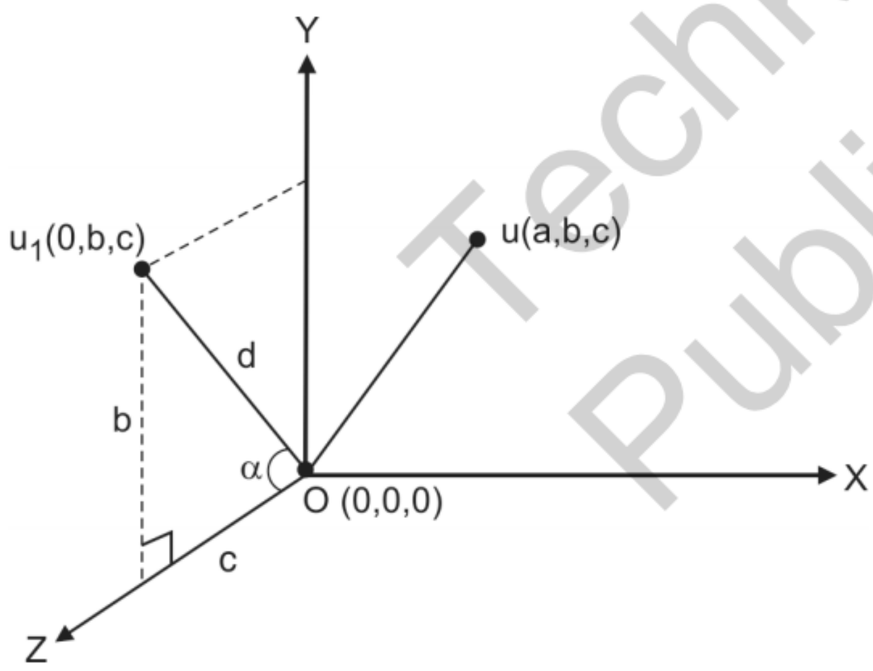


Fig. 6.1.12 : Computation of angle α from projected vector u_1

– From Fig. 6.1.12, sin and cosine angles between u_1 and Z axis is computed using trigonometric rule as,

$$\sin\alpha = \frac{\text{Opposite side}}{\text{Hypotenuse}} = \frac{b}{d}$$

$$\cos\alpha = \frac{\text{Adjacent side}}{\text{Hypotenuse}} = \frac{c}{d}$$

$$d = \sqrt{b^2 + c^2}$$

– The rotation matrix around the X-axis is,

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– This transformation matrix moves unit vector u from space to XZ 2D plane as shown in Fig. 6.1.13. Here, u′ is the vector after rotation of u around the X-axis by an angle α.

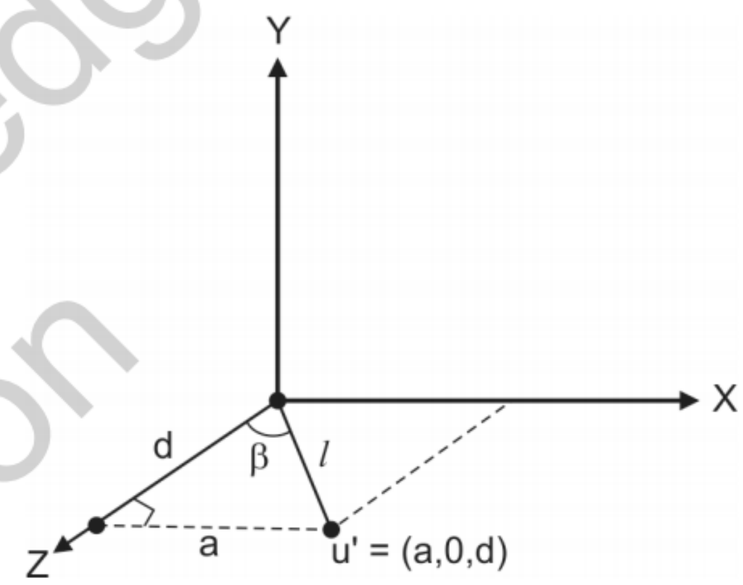


Fig. 6.1.13 : Position of vector u after rotation about X-axis by the angle α

– As rotation was performed about X-axis, the x component of u won't change. And u′ is in XZ plane, so its y component is zero, and z component is d, we already computed. Rotation does not alter the length and hence the length of vector u' will remain 1.

$$\ell = \sqrt{a^2 + (b^2 + c^2)} = 1$$

– If we rotate u′ by the angle β about Y-axis in a clockwise direction, it will be aligned with the Z-axis.

– From Fig. 6.1.13, sin and cosine angles between u′ and Z-axis is computed using the trigonometric rule as,

$$\sin\beta = \frac{\text{Opposite side}}{\text{Hypotenuse}} = \frac{a}{\ell} = a$$

$$\cos\beta = \frac{\text{Adjacent side}}{\text{Hypotenuse}} = \frac{d}{\ell} = d$$

TechKnowledge Publications

– So transformation matrix for rotation about Y-axis would be,

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– Above three steps, translation, rotation about X-axis and rotation about Y-axis would align line P_1P_2 to the Z-axis.

– The actual rotation about line P_1P_2 by the angle θ is achieved by rotation about Z-axis using the following rotation matrix.

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– To move the rotation axis to its actual position, perform the first three steps in reverse order. The composite transformation matrix would be,

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) \cdot R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & y_1 \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & \frac{-b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, $$R_y(\beta) \cdot R_x(\alpha) = \begin{bmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ 0 & \frac{b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & \frac{-ab}{d} & \frac{-ac}{d} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ a & b & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_v$$

And $$R_x^{-1}(\alpha) \cdot R_y^{-1}(\beta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{c}{d} & \frac{b}{d} & 0 \\ 0 & \frac{-b}{d} & \frac{c}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} d & 0 & a & 0 \\ \frac{-ab}{d} & \frac{c}{d} & b & 0 \\ \frac{-ac}{d} & \frac{-b}{d} & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = A_v^T$$

$$\therefore \quad R(\theta) = T^{-1} \cdot A_v^T(\beta) \cdot R_z(\theta) \cdot A_v \cdot T$$

This is the desired transformation sequence.

Technical Knowledge Publication

Note: If the rotation axis is passing from the origin, then translation and inverse translation steps are not required.

– **Assumption:** Length of rotation axis is normalized, i.e. $\ell = |v| = 1$.

6.1.6 Shear

Shearing transformation alters the shape of object. Shearing is used in 3D viewing to derive projection. Shearing transformation matrix for X, Y and Z axis is shown below :

$$SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SH_y = \begin{bmatrix} 1 & c & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & d & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$SH_z = \begin{bmatrix} 1 & 0 & e & 0 \\ 0 & 1 & f & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Parameters a, b, c, d, e and f can take any real value.

Generalized matrix for 3D shearing is described as,

$$SH = \begin{bmatrix} 1 & c & e & 0 \\ a & 1 & f & 0 \\ b & d & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Off diagonal values specify the shear amount in a particular direction. Transformed coordinates after shearing can be computed as,

$$P' = M \cdot P = SH \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & c & e & 1 \\ a & 1 & f & 1 \\ b & d & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Sheared coordinates would be :

$$P' = [(x + cy + ez) \quad (ax + y + fz) \quad (bx + dy + z)]$$

6.1.7 Reflection

Q. Write transformation matrix for 3-D reflection w.r.t. x-axis. **SPPU - May 19, 2 Marks**

– In 3D, reflection can be performed either with respect to the selected axis or with respect to the selected plane. Reflection about an axis is equivalent to rotation by 180° about that axis.

– Transformation matrices for reflection about X = 0, Y = 0 and Z = 0 planes are shown below :

$$Ref_{(X=0)} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ref_{(Y=0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ref_{(Z=0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

– We can compute the reflected coordinate of the object about a plane X = 0 by following transformation sequence.

$$P' = M \cdot P = Ref_{(X=0)} \cdot P$$

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

$$x' = -x$$

$$y' = y$$

$$z' = z$$

Technknowledge Publications

– When we take reflection about X = 0 plane, only X-coordinate of the object vertices would be flipped, there won't be any change in Y and Z coordinates. We can observe the similar effect for reflection about other planes. Fig. 6.1.14 describes the reflection about X = 0 plane. Grey object is the reflected one.

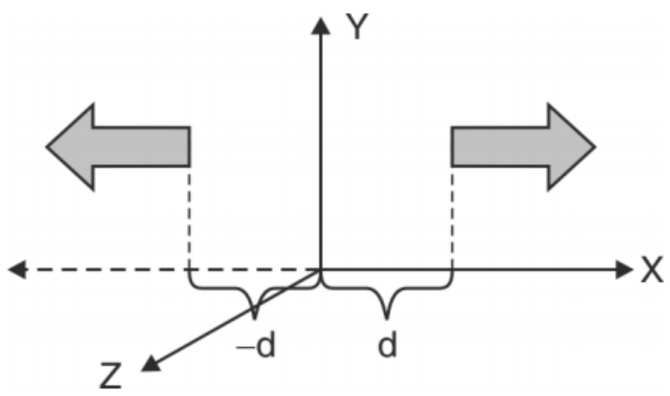


Fig. 6.1.14 : Reflection about X = 0 plane

Reflection through the General Plane

Above matrices are valid for reflection through any of the principal planes. We can extend the concept for reflection through arbitrary plane using rotation, translation and reflection.

Plane is uniquely identified by two parameters: the reference point from which plane is passing and the normal vector of the plane. Let us find the transformation matrix for reflection through a plane passing from reference point R_0 and having normal vector N = ai + bj + ck. The transformation is achieved by performing the following steps :

1. Translate the reference point to the origin.
2. Align the normal vector N to Z-axis by performing the following steps.
   (i) Rotate normal N by the angle α about X-axis such that it falls in XY plane.
   (ii) Rotate the vector in XY plane about Y-axis by the angle β such that it gets aligned with the Z-axis.
3. Perform reflection through Z = 0 plane.
4. Inverse rotation about Y axis by angle β.
5. Inverse rotation about X-axis by angle α.
6. Inverse translation to reposition the reference point to the original location.

So, the transformation matrix would be,

$$M = T^{-1} \cdot R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot Ref_{(Z=0)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)} \cdot T$$

If reference point is specified at the origin, then we do not need to perform step 1 and 6. In that case,

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot Ref_{(Z=0)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)}$$

Ex. 6.1.2 : Find the transformation matrix for mirror reflection with respect to the plane passing through the origin and having normal vector in direction N = i + j + k.

Soln. :

The geometrical representation of plane passing through origin and having normal vector in direction N = i + j + k is depicted in Fig. P. 6.1.2.

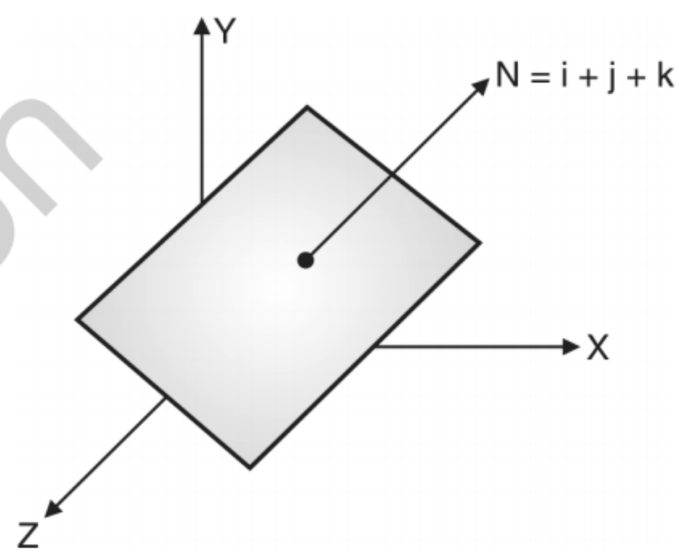


Fig. P. 6.1.2

Perform the following steps to carry out the desired reflection.

1. Rotate normal vector N about X-axis by the angle α so that it falls in XZ plane.

   Let us consider the unit normal such that

$$N = ai + bj + ck$$

$$\text{where } (a, b, c) = (1, 1, 1).$$

$$\cos\alpha = \frac{c}{\sqrt{b^2 + c^2}},$$

$$\sin\alpha = \frac{b}{\sqrt{b^2 + c^2}}$$

$$\text{Let us take } \lambda = d = \sqrt{b^2 + c^2}$$

$$\text{and } |V| = \sqrt{a^2 + b^2 + c^2}$$

Technknowledge Publication

$$\lambda = d = \sqrt{1+1} = \sqrt{2}$$

$$\text{and } |V| = \sqrt{1+1+1} = \sqrt{3}$$

2. Rotation about Y-axis in clockwise direction by angle β so that vector gets align with Z-axis.

$$\cos\beta = \frac{\sqrt{b^2+c^2}}{\sqrt{a^2+b^2+c^2}},$$

$$\sin\beta = \frac{a}{\sqrt{a^2+b^2+c^2}}$$

3. Reflection through XY plane.

4. Inverse rotation about Y-axis by angle β.

5. Inverse rotation about X-axis by angle α.

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot RF_{(z=0)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)}$$

$$= A^T_v \cdot Ref_{(z=0)} \cdot A_v$$

$$A_v = \begin{bmatrix} \frac{d}{|v|} & \frac{-ab}{d|v|} & \frac{-ac}{d|v|} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ \frac{a}{|v|} & \frac{b}{|v|} & \frac{c}{|v|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Normal, N = i + j + k, so a = b = c = 1

$$d = \sqrt{1^2+1^2} = \sqrt{2}$$

$$|v| = \sqrt{1^2+1^2+1^2} = \sqrt{3}$$

$$\therefore A_v = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$RF_{(z=0)} \cdot A_v = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = A^T_v \cdot Ref_{(z=0)} \cdot A_v = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} \frac{1}{3} & \frac{-2}{3} & 0 & 0 \\ \frac{-2}{3} & \frac{1}{3} & 0 & 0 \\ \frac{-2}{\sqrt{3}} & \frac{-2}{3} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

Ex. 6.1.3 : Find the transformation matrix for reflection through the plane passing from reference point $R(x_r, y_r, z_r)$ and having direction vector N = i + j + k.

Soln :

The geometrical representation of plane passing through origin and having normal vector in direction N = i + j + k is depicted in Fig. P. 6.1.3

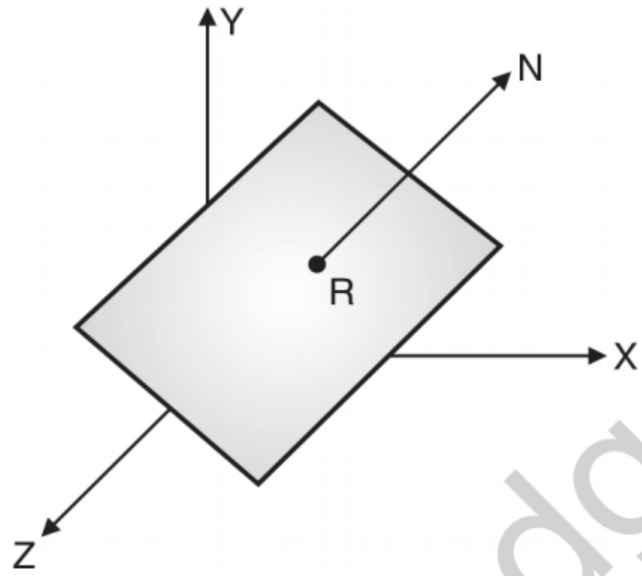


Fig. P. 6.1.3

To perform the desired reflection perform the following steps :

1. Translate reference point to the origin.

$$T = (x_r, y_r, z_r)$$

2. After translation, this case becomes identical to the previous example. To perform steps mentioned in the previous example to derive intermediate transformation matrix.

3. Inverse translation.

$$M = \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{-2}{3} & 0 & 0 \\ \frac{-2}{3} & \frac{1}{3} & 0 & 0 \\ \frac{-2}{3} & \frac{-2}{3} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & 1 & 0 & -y_r \\ 0 & 0 & 1 & -z_r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & \frac{-2}{3} & 0 & \frac{-1}{3}x_r + \frac{2}{3}y_r \\ \frac{-2}{3} & \frac{1}{3} & 0 & \frac{-2}{3}x_r + \frac{1}{3}y_r \\ \frac{-2}{3} & \frac{-2}{3} & 1 & \frac{2}{3}x_r + \frac{2}{3}y_r - z_r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

$$= \begin{bmatrix} \frac{1}{3} & \frac{-2}{3} & 0 & \frac{2}{3}(x_r + y_r) \\ \frac{-2}{3} & \frac{1}{3} & 0 & \frac{1}{3}(-2x_r + 4y_r) \\ 0 & 0 & 1 & \frac{2}{3}(x_r + y_r) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.1.8 Solved Examples

Ex. 6.1.4 : A cube is defined by 8 vertices : A(0, 0, 0), B(2, 0, 0), C(2, 2, 0), D(0, 2, 0), E(0, 0, 2), F(0, 2, 2), G(2,0,2), H(2, 2, 2). Perform following 3D transformations on the cube.

(i) Translation by $T = [t_x, t_y, t_z] = [5, 3, 4]$

(ii) Scaling by $S = [s_x, s_y, s_z] = [1, 2, 0.5]$

(iii) Rotation about X-axis by 90° in clockwise direction

Soln. :

Given cube is shown in Fig. P. 6.1.4.

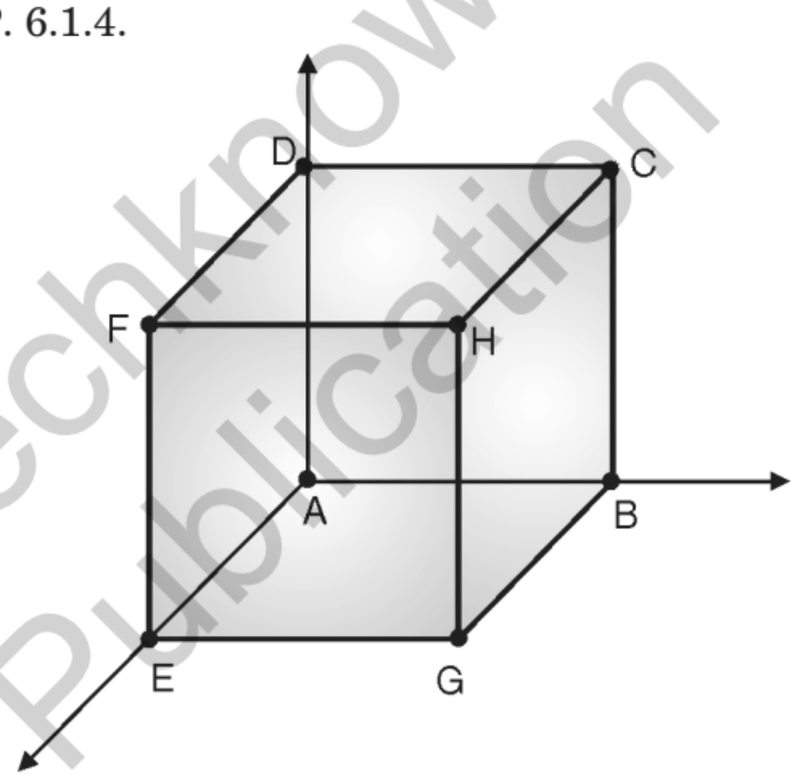


Fig. P. 6.1.4 : 2 × 2 × 2 Cube with A(0, 0, 0) at origin

Let us compute the coordinates for asked operations.

1. Translation by T = [5, 3, 4]

Coordinates of translated cube are computed as,

$$P' = T \cdot P = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 5 & 7 & 7 & 5 & 5 & 5 & 7 & 7 \\ 3 & 3 & 5 & 5 & 3 & 5 & 3 & 5 \\ 4 & 4 & 4 & 4 & 6 & 6 & 6 & 6 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

2. Scaling by S = [1, 2, 0.5]

Coordinates of Scaled cube are computed as,

$$P' = S \cdot P$$

$$= \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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

$$P = \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 4 & 4 & 0 & 4 & 0 & 4 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

3. Rotation about X-axis by 90° in clockwise direction

Here, rotation is in clockwise direction, so $\theta = -90°$

Coordinates of Scaled cube are computed as,

$$P' = R_{x(\theta = -90^o)} \cdot P$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(-90^\circ) & -\sin(-90^\circ) & 0 \\ 0 & \sin\theta(-90^\circ) & \cos(-90^\circ) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Technknowledge Publication

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

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

Ex. 6.1.5 : Find a sequence of transformations required to rotate a solid object w.r.t. a line y = mx + C in anticlockwise direction by angle θ.

Soln. :

Step 1 : Line y = mx + c is shown in Fig. P. 6.1.5. Here, c is the y-intercept translating the line by the amount [0, – c, 0].

We can make the rotation axis pass from the origin by following translation operation.

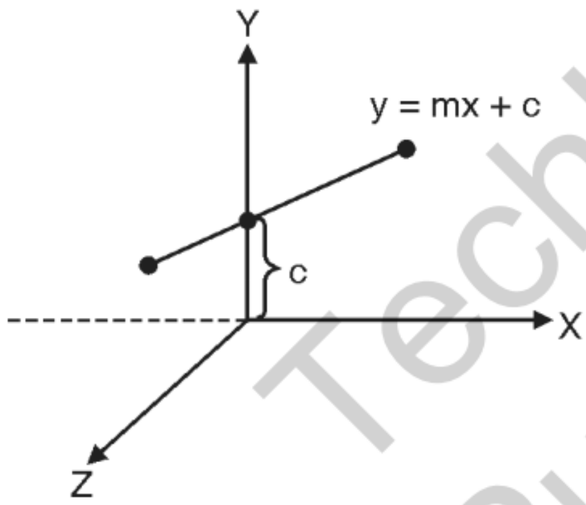


Fig. P. 6.1.5

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

Step 2 : Rotate line about X-axis by the angle α in the anticlockwise direction such that it falls in XZ plane.

$$\therefore \quad R_{x(\alpha)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3 : Now rotate line about Y-axis by the angle β in a clockwise direction. Such that it gets aligned with the Z-axis.

$$\therefore \quad R_{y(\beta)} = \begin{bmatrix} \cos(-\beta) & 0 & \sin(-\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(-\beta) & 0 & \cos(-\beta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Step 4 : Now line y = mx + c is aligned with z-axis, to perform actual rotation about z-axis by angle θ

$$\therefore \quad R_{z(\theta)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 5 : Inverse rotation about Y axis by angle β

$$\therefore \quad R^{-1}_{y(\beta)} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Techknowledge Publication

Step 6 : Inverse rotation about X-axis by the angle α to put a line in its original orientation.

$$\therefore \quad R^{-1}_{x(\alpha)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 7 : Inverse translation of line by vector.

[0, c, 0] to place it on its actual position.

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

$\therefore$ Desired transformation sequence to rotate an object about line y = mx + c by angle θ is,

$$M = T^{-1} \cdot R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)} \cdot T$$

Ex. 6.1.6 : Rotate a triangle ABC with vertices A(2, 2, 2), B(3, 4, 7) and C(8, 9, 12) about Y-axis in anticlockwise direction by angle 90°.

Soln. :

The rotation angle is 90° and rotation is to be performed in an anticlockwise direction, so $\theta = +90°$. Rotation is to be performed about the Y-axis, so transformed coordinates are computed as,

$$P' = R_y(\theta) \cdot P = \begin{bmatrix} \cos 90^\circ & 0 & \sin 90^\circ & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 90^\circ & 0 & \cos 90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 8 \\ 2 & 4 & 9 \\ 2 & 7 & 12 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 7 & 12 \\ 2 & 4 & 9 \\ -2 & -3 & -8 \end{bmatrix}$$

Rotated vertices are A′ (2, 2, – 2), B′ (7, 4, – 3) and C′ (12, 9, – 8).

Original Coordinates	Transformed Coordinates
A(2, 2, 2)	A'(2, 2, – 2)
B(3, 4, 7)	B'(7, 4, – 3)
C(8, 9, 12)	C'(12, 9, – 8)

Ex. 6.1.7 : Find transformation matrix for rotation about a line parallel to the X-axis and passing through the reference point P (x_r, y_r, z_r).

Soln. :

Mentioned composite transformation is achieved by performing the sequence of the following steps :

1. Translation by T ($-x_r$, $-y_r$, $-z_r$)

2. Rotation about X-axis by angle θ.

3. Inverse translation.

$$\therefore \quad M = T^{-1} \cdot R_x(\theta) \cdot T = \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & 1 & 0 & -y_r \\ 0 & 0 & 1 & -z_r \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & x_r \\ 0 & 1 & 0 & y_r \\ 0 & 0 & 1 & z_r \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_r \\ 0 & \cos\theta & -\sin\theta & -y_r\cos\theta + z_r\sin\theta \\ 0 & \sin\theta & \cos\theta & -y_r\sin\theta - z_r\cos\theta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & z_r\sin\theta + y_r(1-\cos\theta) \\ 0 & \sin\theta & \cos\theta & z_r(1-\cos\theta) - y_r\sin\theta \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ex. 6.1.8 : The pyramid with coordinates A (0, 0, 0), B (1, 0, 0), C (0, 1, 0) and D (0, 0, 1) is to be rotated by about line L that has direction vector v = j + k and passing through point (0, 1, 0). Find the coordinates of the transformed pyramid.

Soln. :

The orientation of said pyramid is shown in Fig. P. 6.1.8.

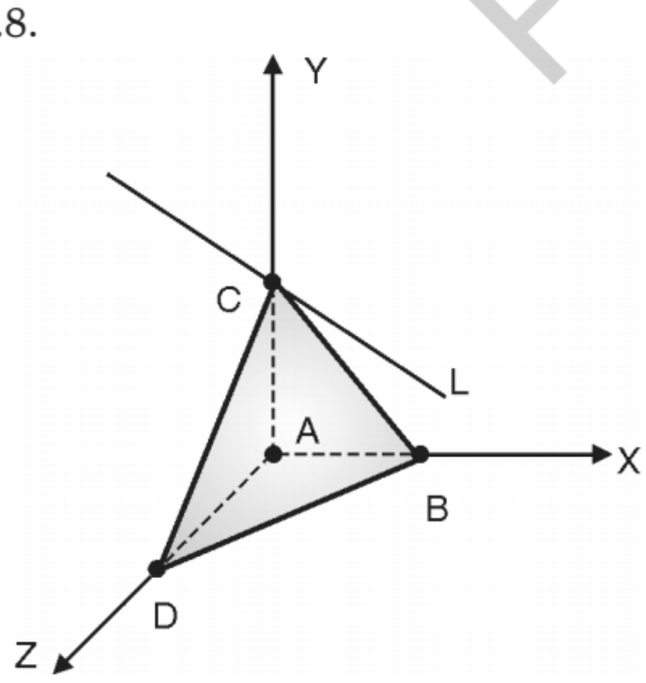


Fig. P. 6.1.8

To rotate the pyramid about line L, we need to align it with one of the principal axes. Let us align it with Z-axis.

We should perform the following steps to achieve the desired operation.

1. Translate line by a vector (0, – 1, 0) so that it passes through the origin.

   $\therefore \quad T = [0 \; -1 \; 0]$

2. Rotate this translated line by an angle α about X-axis in an anticlockwise direction, so that it gets align with Z-axis.

   Line L is in direction of vector v = j + k, so it makes an angle of α = 45° with Y and Z axis and the line is already in YZ plane. So only one rotation is sufficient to align the line with the Z-axis.

3. Perform rotation about Z-axis by angle θ = 90°.

4. Inverse rotation by angle α.

5. Inverse translation by vector T.

Tech Knowledge Publications

$$M = T^{-1} \cdot R_{x(-\alpha)} \cdot R_z(\theta) \cdot R_{x(\alpha)} \cdot T$$

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

Multiplying first two and last two matrices,

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

Multiplying last two matrices,

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

Transformed coordinates are computed as,

$$P' = M \cdot P = \begin{bmatrix} 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} \\ \frac{-1}{\sqrt{2}} & \frac{1}{2} & \frac{1}{2} & \frac{-1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\therefore P' = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & \frac{2}{\sqrt{2}} \\ \frac{-1}{2} & \frac{\sqrt{2}-1}{2} & 0 & 0 \\ \frac{-1}{2} & -\left(\frac{\sqrt{2}+1}{2}\right) & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Technknowledge Publication

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	$A'\left(\frac{1}{\sqrt{2}}, \frac{-1}{2}, \frac{-1}{2}\right)$
B(1, 0, 0)	$B'\left(\frac{1}{\sqrt{2}}, \frac{\sqrt{2}-1}{2}, -\frac{\sqrt{2}+1}{2}\right)$
C(0, 1, 0)	C′ (0, 0, 0)
D(0, 0, 1)	$D'\left(\frac{2}{\sqrt{2}}, 0, 0\right)$

Ex. 6.1.9 : Find out the 3D transformation matrix to rotate a given 3D object by an amount 60° about a line passing from point(1,1,1) and the direction vector V = 2i + 2j + 2k.

Soln. :

Here $\theta = 60°$

Line is passing from (1, 1, 1) and its direction vector is V = 2i + 2j + 2k. So line interpolates all points having the same x, y and z components hence line also passes through the origin.

$$V = 2i + 2j + 2k = i + j + k = ni + nj + nk$$

$$\text{So, } (a, b, c) = (1, 1, 1)$$

$$\text{As derived in the previous section, } d = \sqrt{b^2 + c^2} = \sqrt{1+1} = \sqrt{2}$$

$$\text{And Length of the vector, } \ell = |v| = \sqrt{a^2 + b^2 + c^2} = \sqrt{1+1+1} = \sqrt{3}$$

Transformation matrix to rotate an object about the arbitrary line is given as,

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)}$$

$$= Av^{-1} \cdot R_{z(\theta)} \cdot Av$$

Length of vector V is not normalized, so

$$R_{x(\beta)} \cdot R_{x(\alpha)} = A_v = \begin{bmatrix} \frac{d}{|v|} & \frac{-ab}{(d \cdot |v|)} & \frac{-ac}{(d \cdot |v|)} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ \frac{a}{|v|} & \frac{b}{|v|} & \frac{c}{|v|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

$$R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} = A_v^T = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{z(\theta = 60°)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{3} & -\sqrt{3} & 0 & 0 \\ \sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = A_v^T \cdot R_{z(\theta = 60°)} \cdot A_v$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{3} & -\sqrt{3} & 0 & 0 \\ \sqrt{3} & \sqrt{3} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & \frac{-1}{\sqrt{2}\sqrt{3}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices,

$$= \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{2}\sqrt{3}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{2}\sqrt{3}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{2} & \frac{-1-\sqrt{3}}{\sqrt{2}} & \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ \sqrt{2} & \frac{\sqrt{3}-1}{\sqrt{2}} & \frac{\sqrt{3}-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} \frac{2\sqrt{3}+1}{3} & -\frac{\sqrt{3}-2}{3} & \frac{4-\sqrt{3}}{3} & 0 \\ \frac{4-\sqrt{3}}{3} & \frac{\sqrt{3}+3}{6} & \frac{7\sqrt{3}-7}{6} & 0 \\ -\frac{\sqrt{3}-2}{3} & \frac{11-5\sqrt{3}}{6} & -\frac{4-2\sqrt{3}}{6} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ex. 6.1.10 : Rotate a triangle with vertices A(0, 2, 1), B(2, 3, 0) and C(1, 2, 1) by 45° around a line passing through the points (0, 0 0) and (1, 1, 1).

Tech Knowledge Publications

Soln. :

As derived in previous section,

$$D = \sqrt{b^2 + c^2} = \sqrt{(0-1)^2 + (0-1)^2}\sqrt{1+1} = \sqrt{2}$$

And Length of the rotation axis,

$$\ell = |v| = \sqrt{a^2 + b^2 + c^2} = \sqrt{(0-1)^2 + (0-1)^2 + (0-1)^2} = \sqrt{1+1+1} = \sqrt{3}$$

The rotation axis is already passing through the origin, so there is no need for translating it. So the desired transformation sequence can be achieved by,

$$M = R^{-1}_{x(\alpha)} \cdot R^{-1}_{y(\beta)} \cdot R_{z(\theta)} \cdot R_{y(\beta)} \cdot R_{x(\alpha)} = Av^{-1} \cdot R_{z(\theta)} \cdot Av$$

Where,
$$Av = \begin{bmatrix} \frac{d}{|v|} & \frac{-ab}{(d \cdot |v|)} & \frac{-ac}{(d \cdot |v|)} & 0 \\ 0 & \frac{c}{d} & \frac{-b}{d} & 0 \\ \frac{a}{|v|} & \frac{b}{|v|} & \frac{c}{|v|} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{z(\theta = 45°)} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos 45° & -\sin 45° & 0 & 0 \\ \sin 45° & \cos 45° & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = Av^{-1} \cdot R_{z(\theta)} \cdot Av$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & 0 & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 0.815 & 0 & 0.578 & 0 \\ -0.408 & 0.709 & 0.578 & 0 \\ -408 & -0.709 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.709 & -0.709 & 0 & 0 \\ 0.709 & 0.709 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.815 & -0.408 & 0.408 & 0 \\ 0 & 0.709 & 0.709 & 0 \\ 0.578 & 0.578 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying last two matrices,

$$M = \begin{bmatrix} 0.815 & 0 & 0.578 & 0 \\ -0.408 & 0.709 & 0.578 & 0 \\ -408 & -0.709 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.578 & -0.792 & 0.213 & 0 \\ 0.578 & 0.213 & -0.792 & 0 \\ 0.578 & 0.578 & 0.578 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Techknowledge Publication

$$M = \begin{bmatrix} 0.805 & -0.311 & 0.508 & 0 \\ 0.508 & 0.808 & -0.0314 & 0 \\ -0.312 & 0.506 & 0.809 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates,

$$P' = M \cdot P = \begin{bmatrix} 0.805 & -0.311 & 0.508 & 0 \\ 0.508 & 0.808 & -0.0314 & 0 \\ -0.312 & 0.506 & 0.809 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 2 & 1 \\ 2 & 3 & 2 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -0.114 & 0.677 & 0.691 \\ 1.302 & 3.440 & 1.810 \\ 1.821 & 0.894 & 1.509 \\ 1 & 1 & 1 \end{bmatrix}$$

Original Coordinates	Transformed Coordinates
A(0, 2, 1)	A′ (– 0.114, 1.302, 1.821)
B(2, 3, 0)	B′ (0.677, 3.440, 0.894)
C(1, 2, 1)	C′ (0.691, 1.810, 1.509)

Ex. 6.1.11 : For origin centered unit square, rotate 45° clockwise, scale by a factor 2 in the x-direction. Find resultant coordinates of the square (write required matrices). **SPPU - Dec. 17, 4 Marks**

Soln. :

The square with size 2 and center at the origin with its vertex coordinates is shown below :

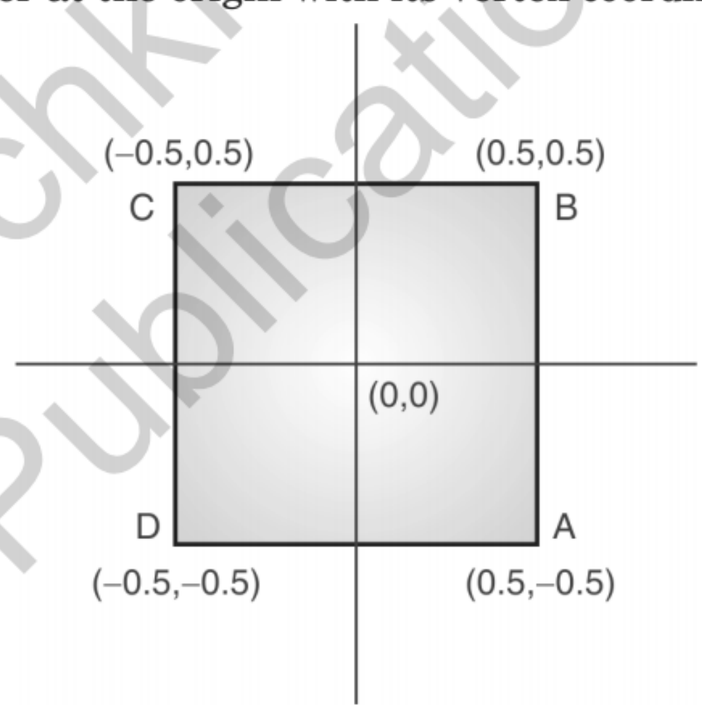


Fig. P. 6.1.11

The square is to be rotated by 45° in the clockwise direction followed by scaled by factor 2 in the x-direction. So $\theta = -45°$, $S_x = 2$ and $S_y = 1$.

The coordinates of the rotated square are computed as,

$$P' = R \cdot S \cdot P = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 0.5 & 0.5 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.71 & 0.71 & 0 \\ -0.71 & 0.71 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 0.5 & 0.5 & -0.5 & -0.5 \\ -0.5 & 0.5 & 0.5 & -0.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.71 & 0.71 & 0 \\ -0.71 & 0.71 & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 1 & -1 & -1 \\ -0.5 & 0.5 & 0.5 & -0.5 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0.355 & 1.065 & -0.355 & -1.065 \\ -1.065 & -0.355 & 1.065 & 0.355 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Thus,

Original Coordinates	Transformed Coordinates
A(0.5, – 0.5)	A′(0.355, – 1.065)
B(0.5, 0.5)	B′(1.065, – 0.355)
C(– 0.5, 0.5)	C′(– 0.355, 1.065)
D(– 0.5, –0.5)	D′(– 1.065, 0.355)

Ex. 6.1.12 : Rotate origin centered square with 2 unit length of each side, in the clockwise direction with a rotation angle of 90°. **SPPU - May 18, 3 Marks**

Soln. :

The square with size 2 and center at the origin with its vertex coordinates is shown below :

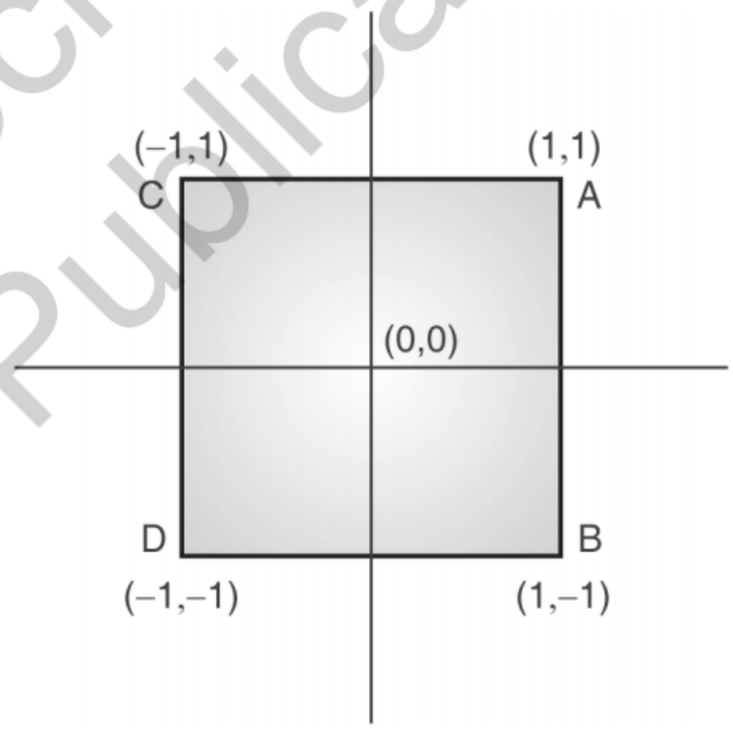


Fig. P. 6.1.12

The square is to be rotated in a clockwise direction, so angle θ = – 90°.

The coordinates of a rotated square are computed as,

$$P' = M \cdot P = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

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

Thus,

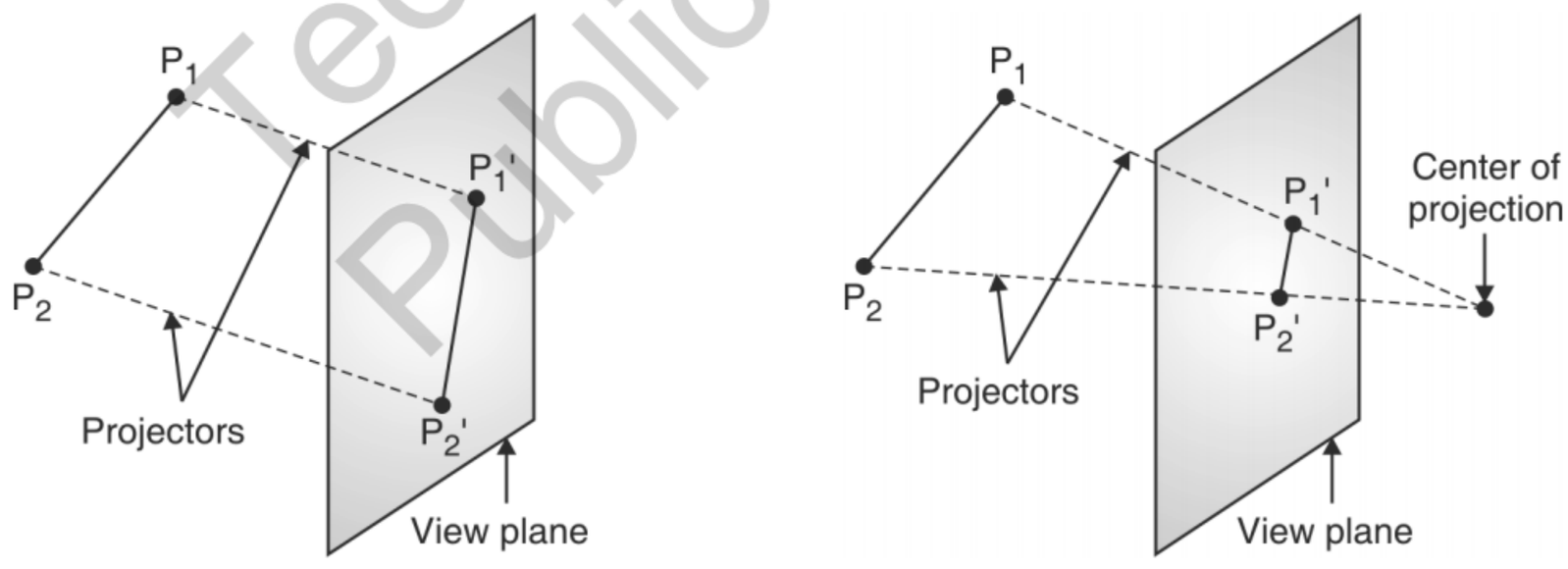
Original Coordinates	Transformed Coordinates
A(1, 1)	A′(1, – 1)
B(1, – 1)	B′(– 1, – 1)
C(– 1, 1)	C′(1, 1)
D(– 1, – 1)	D′(1, 1)

6.2 Projections

6.2.1 Introduction

Q. What are the types of projection and write in brief about each type of projections. **SPPU - May 17, May 19, 6 Marks**

- **Projection** is the process of transforming an object representation from n-dimensional space to less than n-dimensional space.
- A 3D object to 2D projection is a vital operation in CAD-CAM and engineering applications.
- **View plane** is the plane on which the object is projected. It is also known as a **projection plane** or plane of projection.
- Projection is broadly classified into two categories : Parallel projection and perspective projection.



(a) Parallel projection **(b) Perspective projection**

Fig. 6.2.1 : Types of projection

- In parallel projection, coordinate positions of the object are transformed to view plane by parallel rays/projectors (Refer Fig. 6.2.1 (a)).
- In perspective projection, rays are fired from a point source, called center of projection, which intersects the object coordinates and projects it on view plane (Refer Fig. 6.2.1 (b)).

- Fig. 6.2.1 demonstrates parallel and perspective projections. P_1P_2 is the original line in space. $P_1'P_2'$ is projected line on 2D view plane.

Tech Knowledge Publications

- In the case of parallel projection, projectors are parallel to each other, whereas in case of perspective projection rays are inclined and meeting at a point called **center of projection**.
- We can summarize types of projections as follows :

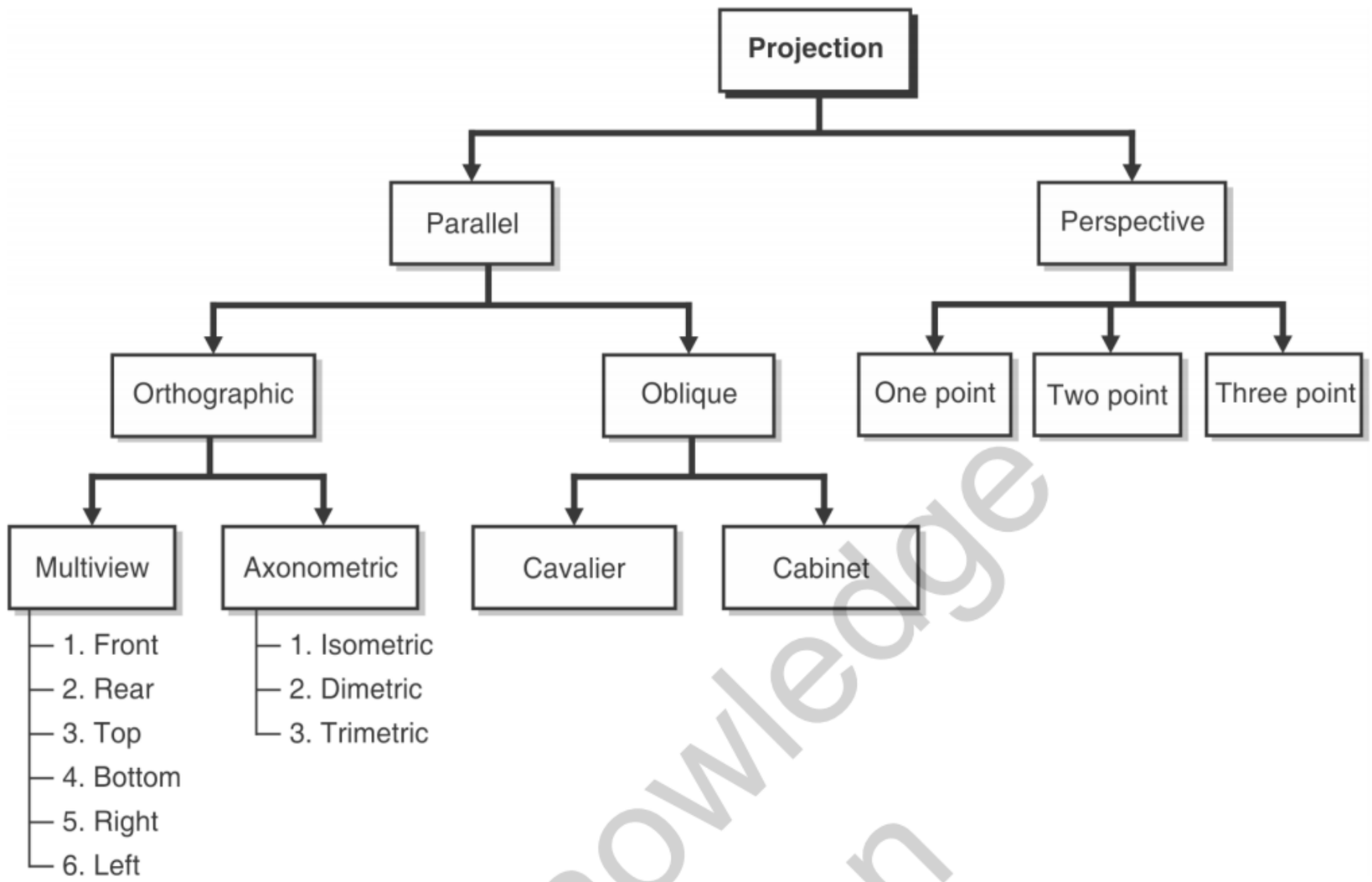
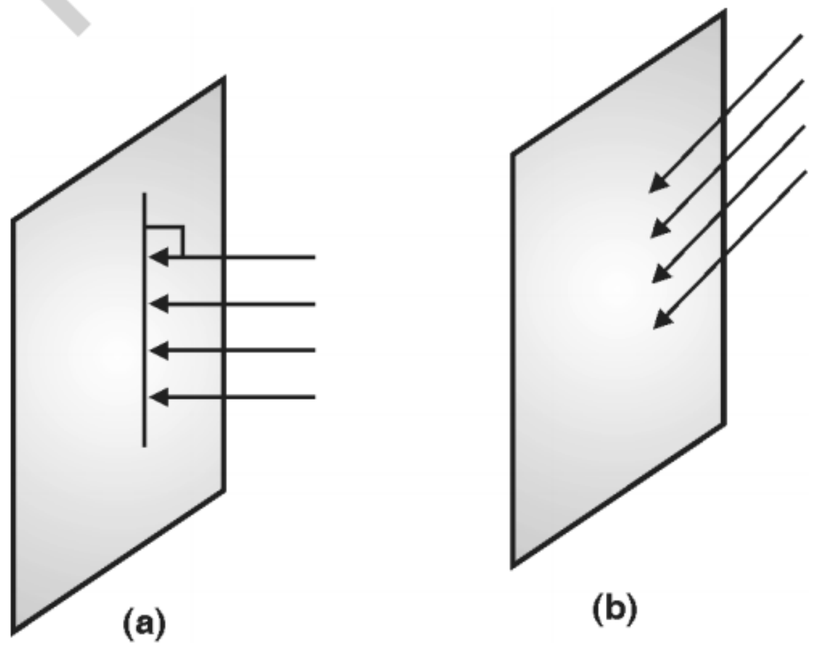


Fig. 6.2.2 : Taxonomy of projection

6.2.2 Parallel Projection

Q.	Explain parallel projection with example.	**SPPU - Dec. 15, 2 Marks**
Q.	Explain the following term with an example : Parallel projection	**SPPU - Dec. 18, 2 Marks**

- Parallel projection is achieved by passing parallel rays from the object vertices and projecting the object on view plane.
- In parallel projection, all projection vectors are parallel to each other.



(a) Orthographic parallel projection **(b) Oblique parallel projection**

Fig. 6.2.3 : Types of parallel projection

TechKnowledge Publications

- Parallel projection preserves true shape and size of the object on view plane.
- However, it fails to capture depth information and hence cannot produce a realistic view.
- Based on the direction of the projection vectors and their relation with view plane, parallel projection is further classified as an orthographic parallel projection or oblique parallel projection.
- When all projectors are parallel to each other and perpendicular to the view plane, it is called **orthographic parallel projection**. If projectors are parallel to each other but are not perpendicular to the view plane, it is called **oblique parallel projection**. Both types of projections are illustrated in Fig. 6.2.3.
- Classification of parallel projection is depicted in Fig. 6.2.4.

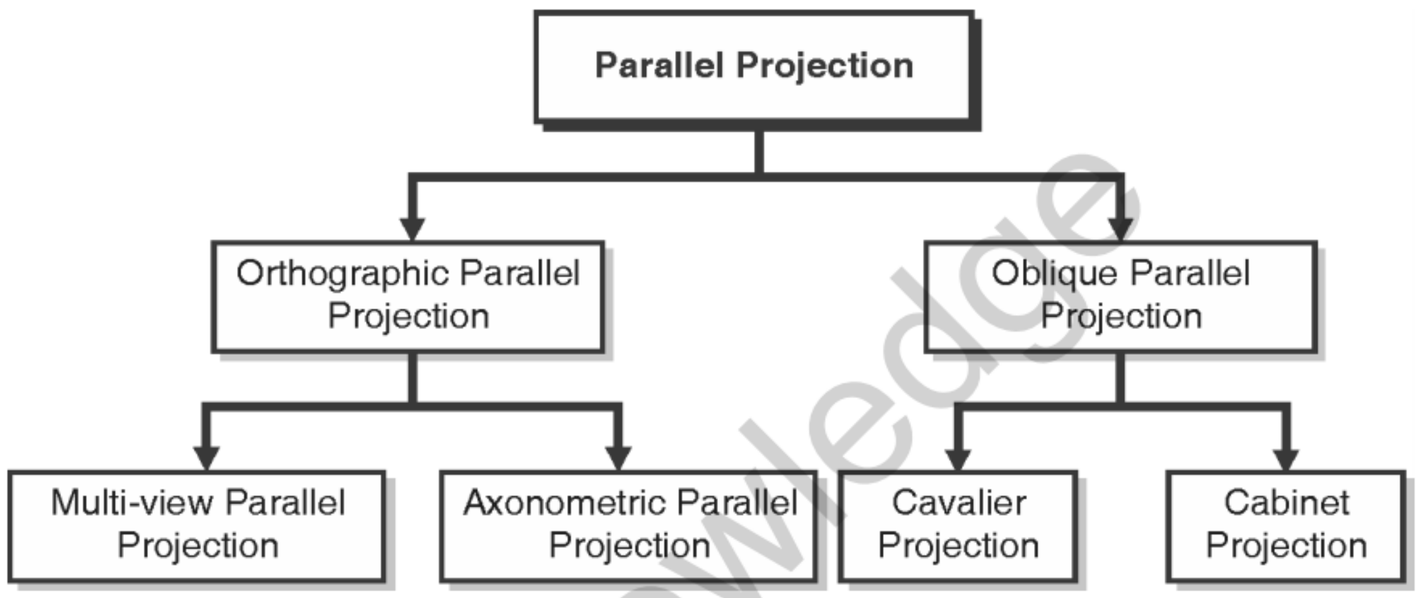


Fig. 6.2.4 : Types of parallel projection

6.2.2.1 Oblique Parallel Projection

- As we discussed earlier, in **oblique projections**, projectors are parallel to each other and they are not perpendicular to the view plane. View in oblique projection is controlled by two parameters ϕ and α as shown in Fig. 6.2.5.

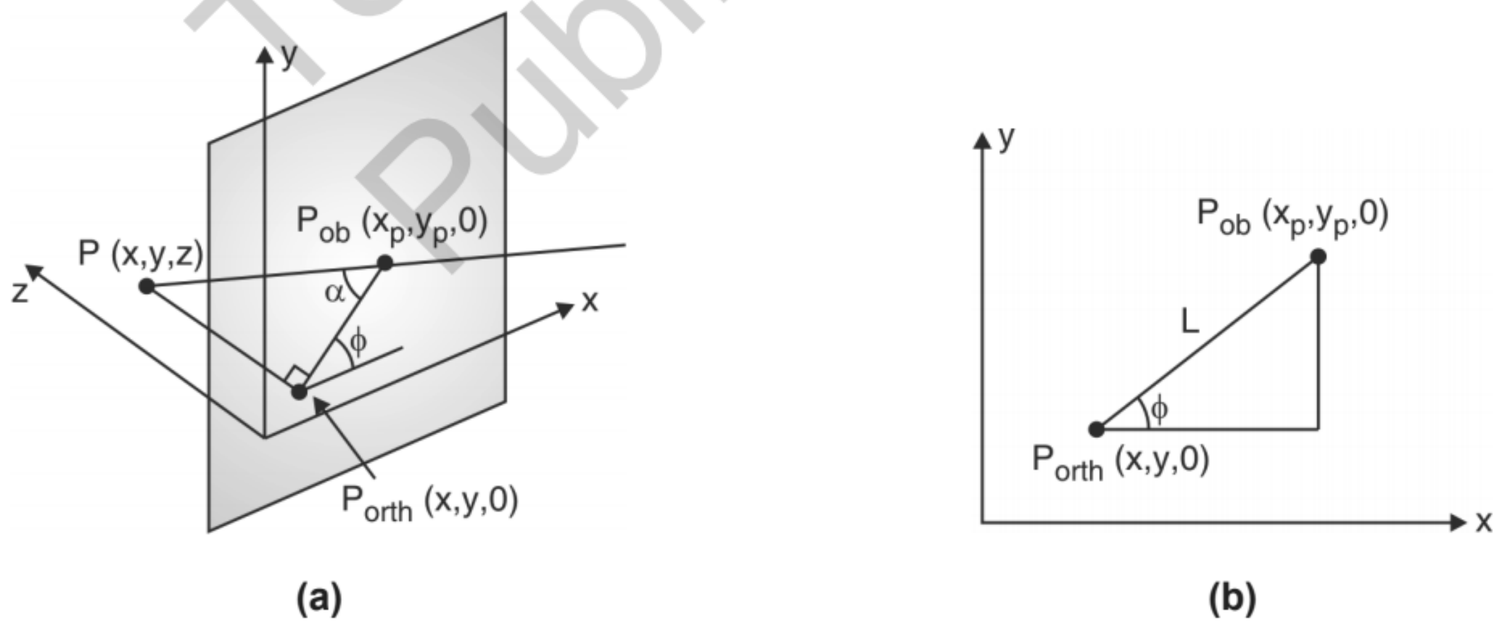


Fig. 6.2.5 : Oblique parallel projection

- In orthographic parallel projection, projectors are perpendicular to the projection plane and hence its corresponding coordinate value becomes zero and rest of the coordinates will not alter. Whereas in case of oblique projection, projectors are inclined with the projection plane, which distorts the projected coordinates.
- Let, P(x, y, z) be the point in space. Orthographic projection of point P on view plane XY is $P_{orth}(x, y, 0)$.
- Oblique projection of P on the same plane is let's say $P_{ob}(x_p, y_p, 0)$.

- Oblique projection vector passing through points P and P_{ob} makes an angle α with view plane. Let the length of line joining points P_{orth} and P_{ob} is L.

  From Fig. 6.2.5(b), values of oblique projection of point P(x, y, z) on XY view plane is given by following equations :

$$x_p = x + L \cos \phi$$
$$y_p = y + L \sin \phi$$

From Fig. 6.2.5 (a),

$$\tan \alpha = \frac{z}{L}$$

$$\therefore \quad L = \frac{z}{\tan \alpha} = zL_1$$

By solving the equations of x_p and y_p for L,

$$x_p = x + zL_1 \cos \phi$$
$$y_p = y + zL_1 \sin \phi$$

So the transformation matrix for any parallel projection on view plane X_vY_v is written as,

$$M = \begin{bmatrix} 1 & 0 & L_1 \cos \phi & 0 \\ 0 & 1 & L_1 \sin \phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

L_1 is the foreshortening factor.

Properties

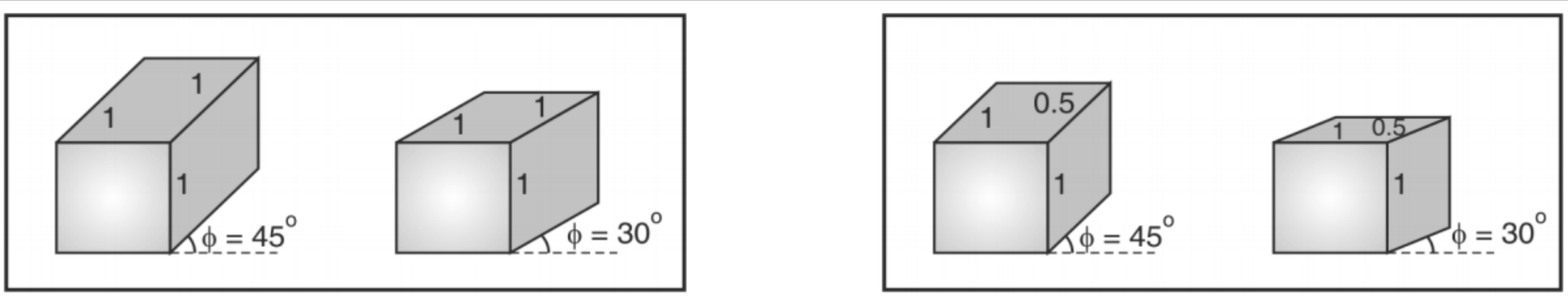
- α is the angle between the oblique projector and plane.
- ϕ is the angle between the horizontal and projected Z-axis.
- $L_1 = 1$ when $\alpha = 90°$, which produces orthographic projection.
- ϕ is a free parameter, but the common choice for ϕ are 30° and 45°.
- Common choice of α are 45° ($\tan \alpha = 1$) and 63.4° ($\tan \alpha = 2$).
- According to the value of α, oblique projection is classified into two categories :
  - Cavalier projection
  - Cabinet projection

1. Cavalier Projection

- If $\alpha = 45°$ (angle between projectors and view plane), the projection is called **cavalier projection**.
- Foreshortening factor for the lines perpendicular to the projection plane would be 1.
- In other words, if the projection angle is 45°, there won't be any change in the projected length of lines which are perpendicular to the view plane.
- Resulting figure in cavalier projection appears thick. 3D nature of the object can be captured but the shape seems to be distorted for lines not parallel to the principal axis (Refer Fig. 6.2.6(a)).

2. Cabinet Projection

- If α is set to 63.4°, the type of projection is known as **cabinet projection**.
- Foreshortening factor for the lines perpendicular to the projection plane would be 0.5.
- In other words, if the projection angle is 63.4°, the projected length of lines perpendicular to the view plane would be half of the original length.
- Cabinet projection is more realistic than cavalier projection.
- Fig. 6.2.6 shows the effect of angle for cabinet and cavalier projection.



(a) Cavalier projection (b) Cabinet projection

Fig. 6.2.6 : Cabinet and cavalier projection

Techknowledge Publication

The relation between angle, projection factor and type of projection is summarized in Table 6.2.1.

Table 6.2.1

The angle between the projector and the view plane(α)	Foreshortening factor	Type of projection
45.0°	1.0	Cavalier
63.4°	0.5	Cabinet
90.0°	0.0	Orthographic

6.2.2.2 Orthographic Parallel Projection

Based on the relation between principal plane and the sides of the object, we can further classify the orthographic parallel projection in two categories :

1. Multi-view parallel projection (Sides of object parallel to principal plane)
2. Axonometric parallel projection (Sides of object are not parallel to principal plane)

1. Multiview Parallel Projection

– Multi-view orthographic projection is used in engineering drawings. It is most often used to generate the front view, side view and top view. It is possible to construct the 3D view of an object from these three views.

– In **multi-view parallel projection**, the plane of projection is parallel to the surface of the object. We get only one surface projected on view plane in multi-view projection.

– The front view is called **elevation** and the top view is called **plan**. Generally, the most descriptive face of an object is selected for elevation. Sometimes, a cross-sectional view is also used to get more details.

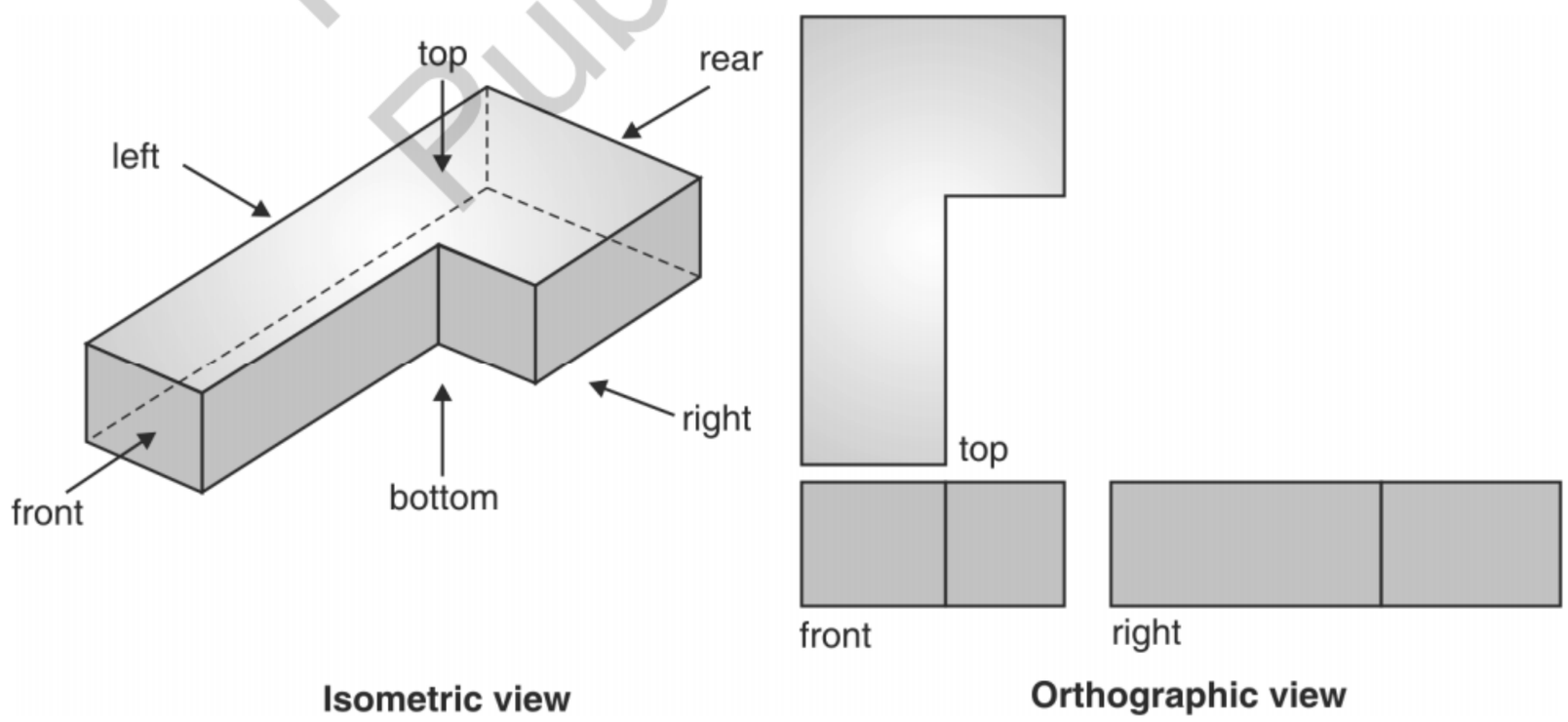


Fig. 6.2.7 : Multi-view parallel projection

– On paper, the top view is placed exactly above elevation. Left and right side views are kept on the respective side of the front view.

- Orthographic parallel projection preserves true length and angle between lines, so used to measure the actual dimension of an object.
- Fig. 6.2.7 shows the isometric 3D view and its orthographic parallel projection with front, top and right side view.

2. Axonometric Parallel Projection

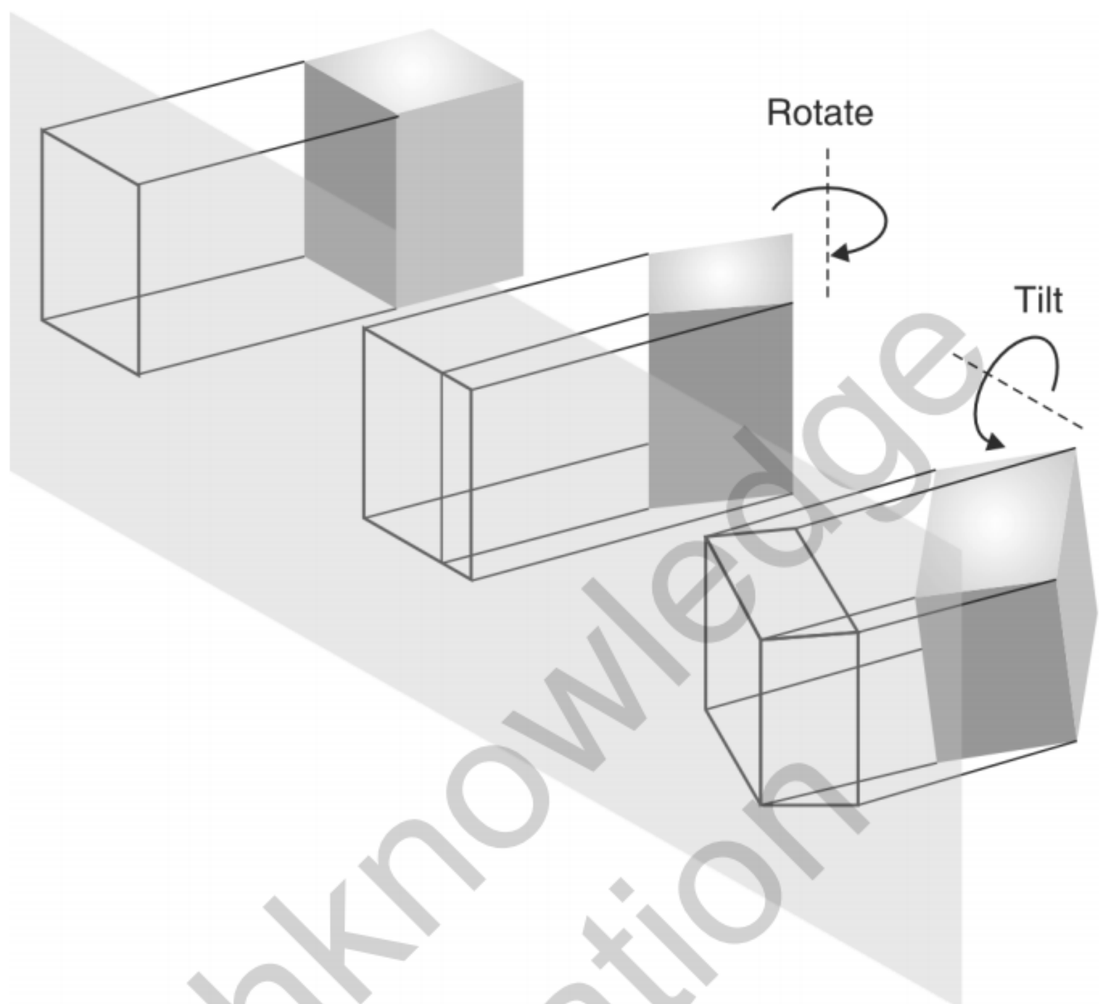


Fig. 6.2.8 : Displaying three surfaces by rotation and tilting

- As shown in Fig. 6.2.8, by performing the rotation of an object, we can display more than one surfaces of an object. Such projection is called **axonometric orthographic projection**.
- Axonometric projection is further classified in isometric, diametric or trimetric by aligning projection plane with three, two or one principal axis respectively.
- Axonometric projection does not preserve the true size of the object. The ratio of projected length to original length is called **foreshortening factor.**
- Isometric projection is achieved by aligning projection vector with the cube diagonal. It preserves the same foreshortening factor for all three axes. For diametric projection, foreshortening factors for any two axes are the same. And for trimetric projection, foreshortening factors for all three principal axes are different.
- Thus, diametric is a special case of trimetric and isometric is a special case of diametric.
- Fig. 6.2.9 explains all three cases of axonometric projection. If diagonal of a unit cube is aligned such that it makes an equal angle with all three axes, projection foreshortens the line with the same factor in all the directions. This is termed as isometric projection.

Tech Knowledge Publications

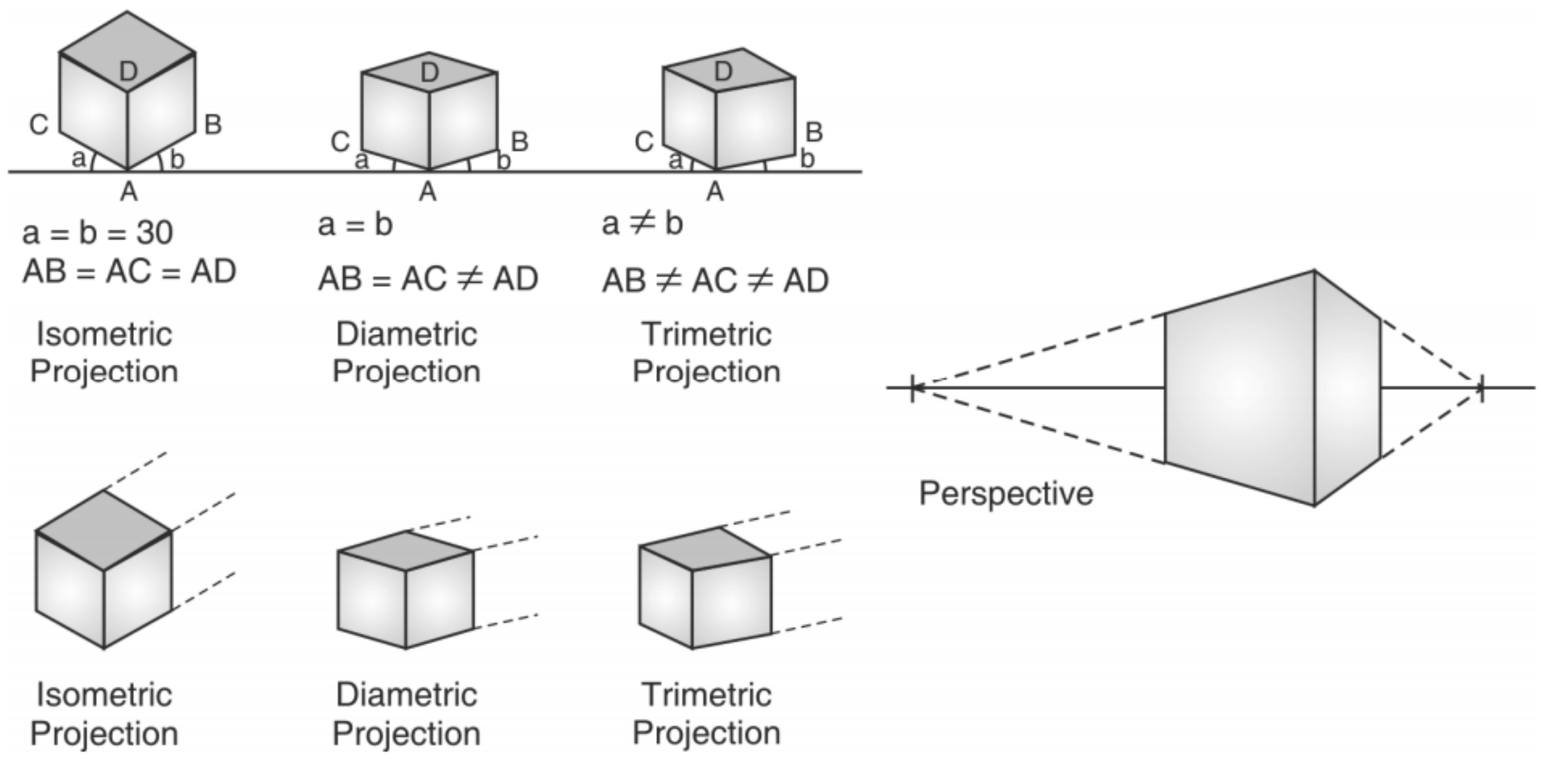


Fig. 6.2.9 : Types of axonometric projection

- In diametric projection, any two sides of cube make equal angle and foreshortening in those two directions would be the same.
- In trimetric projection, all three sides of cube make different angle and their foreshortening factors are also different.
- If the projection plane is placed at Z_{vp} distance on the Z-axis and parallels to XY plane, parallel projection gives the coordinates,

$$x_p = x$$

$$y_p = y$$

- Original z-coordinate value is preserved for depth cueing and for visible surface determination.
- Following are the types of axonometric projection :

i. Trimetric projection

In axonometric projection, at least three faces of an object must be visible. This can be achieved by a first rotating object about Y-axis by angle followed by a rotation about X-axis and then taking projection on z = 0 plane.

The transformation matrix for this would be,

$$M = P_z \cdot P_x \cdot P_y(\phi)$$

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

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

Techknowledge Publications

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

Foreshortening ratios are obtained by multiplying a unit vector with the transformation matrix

$$\begin{aligned} U &= M \cdot I \\ &= \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ \sin\theta \cdot \sin\phi & \cos\theta & -\sin\theta \cdot \cos\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ \sin\theta \cdot \sin\phi & \cos\theta & -\sin\theta \cdot \cos\phi \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{aligned}$$

$$f_x = \sqrt{\cos^2\theta + \sin^2\theta \cdot \sin^2\phi} \quad \ldots(1)$$

$$f_y = \sqrt{\cos^2\theta} \quad \ldots(2)$$

$$f_z = \sqrt{\sin^2\theta + \sin^2\theta \cdot \cos^2\phi} \quad \ldots(3)$$

Where f_x, f_y and f_z defines foreshortening factors in x, y and z-direction respectively.

For trimetric projection, foreshortening factor for all three directions is different. Trimetric projection is least restrictive.

ii. Dimetric projection

Dimetric projection has more restriction than trimetric projection. In diametric projection, foreshortening factor for any two axes are same, so suppose $f_x = f_y$.

$$\therefore \cos^2\theta = \cos^2\phi + \sin^2\theta \cdot \sin^2\phi$$

$$\therefore 1 - \sin^2\theta = 1 - \sin^2\phi + \sin^2\theta \cdot \sin^2\phi$$

$$\sin^2\phi - \sin^2\theta \sin^2\phi = \sin^2\theta$$

$$\sin^2\phi\,(1 - \sin^2\theta) = \sin^2\theta$$

$$\sin^2\phi = \frac{\sin^2\theta}{1 - \sin^2\theta} \quad \ldots(4)$$

From Equation (3),

$$\begin{aligned} f_z^2 &= \sin^2\phi + \sin^2\theta \cdot \cos^2\phi \\ &= \sin^2\phi + \sin^2\theta \cdot (1 - \sin^2\phi) \\ &= \sin^2\phi + \sin^2\theta - \sin^2\theta \cdot \sin^2\phi \end{aligned}$$

Putting Equation (4) in this,

$$f_z^2 = \left(\frac{\sin^2\theta}{1 - \sin^2\theta}\right) + \sin^2\theta - \sin^2\theta \left(\frac{\sin^2\theta}{1 - \sin^2\theta}\right)$$

$$\therefore f_z^2\,(1 - \sin^2\theta) = \sin^2\theta + \sin^2\theta\,(1 - \sin^2\theta) - (\sin^4\theta)$$

$$\therefore f_z^2 - f_z^2 \sin^2\theta = \sin^2\theta + \sin^2\theta - \sin^4\theta - \sin^4\theta$$

$$\therefore 2\sin^4\theta - \sin^2\theta\,(f_z^2 + 2) - f_z^2 = 0$$

Solving it for quadratic equation

$$ax^2 + bx + c = 0\,,\ a = 2,\ b = -(f_z^2 + 2),$$

$$c = f_z^2\,,\ x = \sin^2\theta$$

$$\begin{aligned} \Delta &= b^2 - 4ac \\ &= (f_z^2 + 2)^2 - 4(2)\,(f_z^2) \\ &= f_z^2 + 4 f_z^2 + 4 - 8 f_z^2 \\ &= f_z^2 - 4 f_z^2 + 4 = (f_z^2 - 2)^2 \end{aligned}$$

TechKnowledge Publications

$$r_1, r_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$= \frac{(-f_z^2 + 2) \pm \sqrt{(f_z^2 - 2)^2}}{4}$$

$$r_1 = \frac{f_z^2 + 2 + f_z^2 - 2}{4} = \frac{f_z^2}{2}$$

$$r_2 = \frac{f_z^2 z^2 + 2 - f_z^2 + 2}{4} = 1$$

But from Equation (4), $\sin^2\theta = 1$ is not possible

$$\therefore \sin^2\theta = \frac{f_z^2}{2}$$

$$\therefore \theta = \sin^{-1}\left(\pm\frac{f_z}{\sqrt{2}}\right)$$

From Equation (4),

$$\sin^2\phi = \frac{\sin^2\theta}{1 - \sin^2\theta} = \frac{\frac{f_z^2}{2}}{1 - \frac{f_z^2}{2}} = \frac{f_z^2}{2 - f_z^2}$$

$$\phi = \sin^{-1}\left(\pm\frac{f_z}{\sqrt{2 - f_z^2}}\right)$$

iii. Isometric projection

Isomeric projection is the most restrictive. In isometric projection, foreshortening factor in all three dimensions must be same.

$$f_x = f_z = f_z$$

From Equation (2) and (3),

$$f_y = f_z$$

$$\cos^2\theta = \sin^2\phi + \cos^2\phi \cdot \sin^2\theta$$

$$1 - \sin^2\theta = \sin^2\phi + (1 - \sin^2\phi) \cdot \sin^2\theta$$

$$1 - \sin^2\theta = \sin^2\phi + \sin^2\theta - \sin^2\phi \cdot \sin^2\theta$$

$$1 - 2\sin^2\theta = \sin^2\phi\ (1 - \sin^2\theta)$$

$$\therefore \sin^2\phi = \frac{1 - 2\sin^2\theta}{1 - \sin^2\theta} \quad \ldots(5)$$

By comparing R.H.S of Equation (4) and (5),

$$\frac{\sin^2\theta}{1 - \sin^2\theta} = \frac{1 - 2\sin^2\theta}{1 - \sin^2\theta}$$

$$\therefore \sin^2\theta = 1 - 2\sin^2\theta$$

$$\therefore \sin^2\theta = \frac{1}{3}$$

$$\therefore \theta = \sin^{-1}\left(\pm\frac{1}{\sqrt{3}}\right) = \pm 35.26$$

From Equation (4),

$$\sin^2\phi = \frac{\sin^2\theta}{1 - \sin^2\theta} = \frac{1/3}{1 - 1/3} = \frac{1/3}{2/3} = \frac{1}{2}$$

$$\therefore \phi = \sin^{-1}\left(\pm\frac{1}{2}\right) \pm 45°$$

$$\therefore f_y = f_z$$

$$\therefore f_z = \sqrt{\cos^2\theta}$$

$$= \sqrt{1 - \sin^2\theta} = \sqrt{1 - \frac{1}{3}}$$

$$= \frac{\sqrt{2}}{3} = 0.8165$$

Thus isometric is special case of diametric projection with $f_x = f_y = f_z = 0.8165$.

Isomeric projection is often used in engineering drawing to construct the 3D view of the object.

6.2.3 Perspective Projection

Q. Explain perspective projection with example.

SPPU - Dec. 15, 2 Marks

Q. Explain perspective projection and its types in brief.

SPPU - May 18, 3 Marks

- Perspective projection preserves the depth information but it does not preserve the true shape and size of the object. Size of the projected object depends on the distance between the view plane, the center of projection and the object.
- Projection of a distant object is smaller than that of the nearer object. Fig. 6.2.10 describes this property of perspective projection.

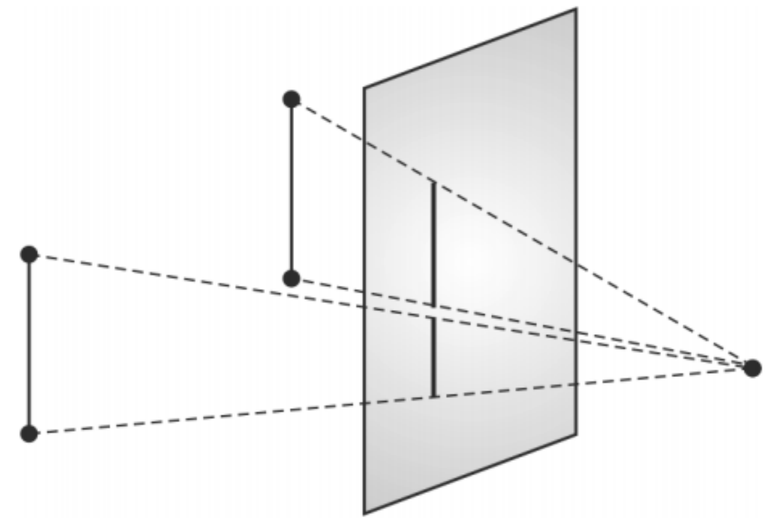


Fig. 6.2.10 : Effect of perspective projection

Technknowledge Publication

Let us derive the transformation matrix for the perspective projection of point P(x, y, z) on view plane Z = 0 (i.e. XY plane). Assume that the center of projection (COP) is at distance d from the origin on Z-axis as shown in Fig. 6.2.11.

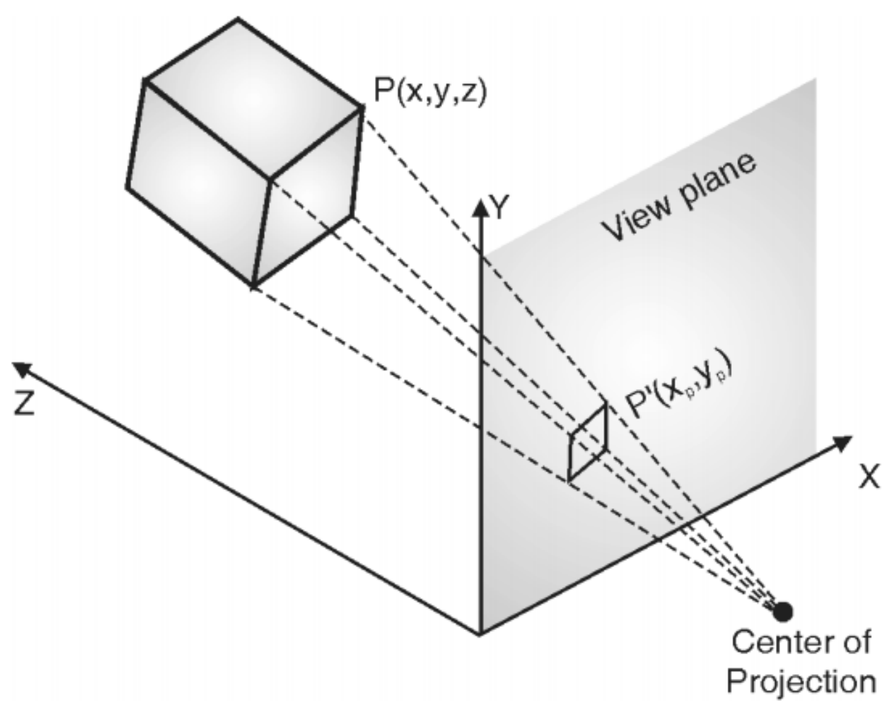


Fig. 6.2.11 : Perspective projection of P(x, y, z) on XY plane

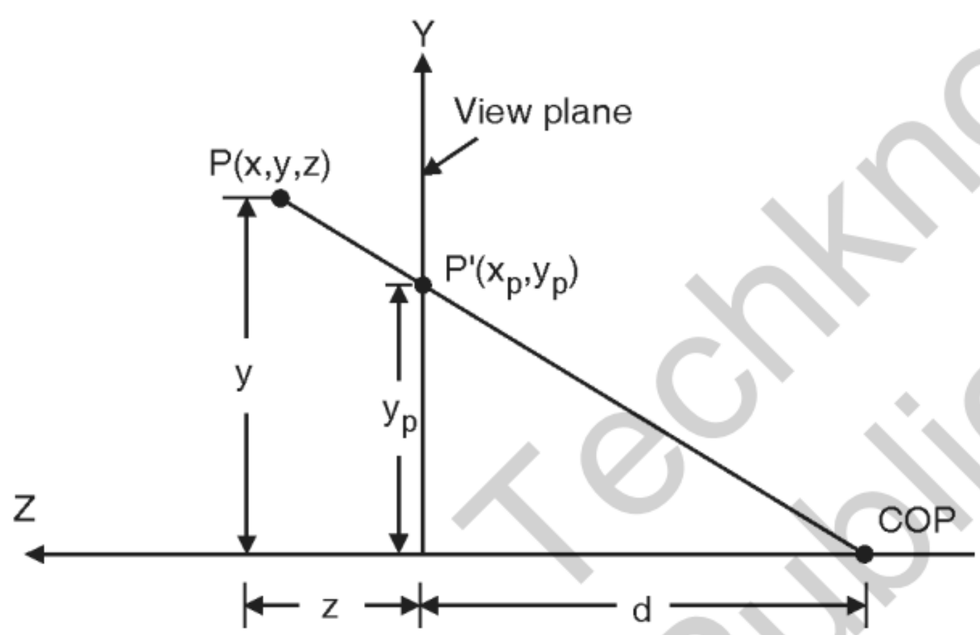


Fig. 6.2.12 : View from X-axis

– For this particular case, we are considering that the object is on positive Z-axis and centre of projection is behind the projection plane i.e. on d distance on negative Z-axis. Although, this can be generalized to any valid position of object, view-plane and COP.

If we look from the X-axis, Fig. 6.2.11 looks like Fig. 6.2.12.

From the triangle equality rule,

$$\frac{y_p}{d} = \frac{y}{z+d}$$

$$y_p = \frac{y \cdot d}{z+d}$$

If we look from the Y-axis, Fig. 6.2.11 looks as Fig. 6.2.12.

From the triangle equality rule,

$$\frac{x_p}{d} = \frac{x}{z+d}$$

$$\therefore x_p = \frac{x \cdot d}{z+d}$$

As view plane is positioned at z = 0, projected z coordinate would be 0.

$$z_p = 0$$

– From this, we can define a transformation matrix as,

$$M = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix}$$

Projected Coordinates, $P' = M \cdot P$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

– If view plane is not XY plane or it is not parallel to XY plane in that case, we need to perform composite transformation such that the normal of the plane get aligned with Z-axis.

– Other cases of perspective projections are discussed at the end of this section.

6.2.3.1 Vanishing Point

– In perspective projection, parallel lines of the object which are not parallel to projection plane are projected into a converging point. In perspective projection, the point where projected parallel lines converge to a single point is called the **vanishing point**.

Techknowledge Publication

- Projection of railway track forms a vanishing point. It seems they are meeting at a single point at a very far distance. A scene can have any number of vanishing points depending on the orientation of parallel lines in the object.
- If the set of parallel lines are parallel to any of the principal axis, vanishing point for such set is called the **principal vanishing point**.
- We can control the numbers of principal vanishing points by rotating the view plane. The intersection of the view plane with a number of principal axes defines the number of principal vanishing points.

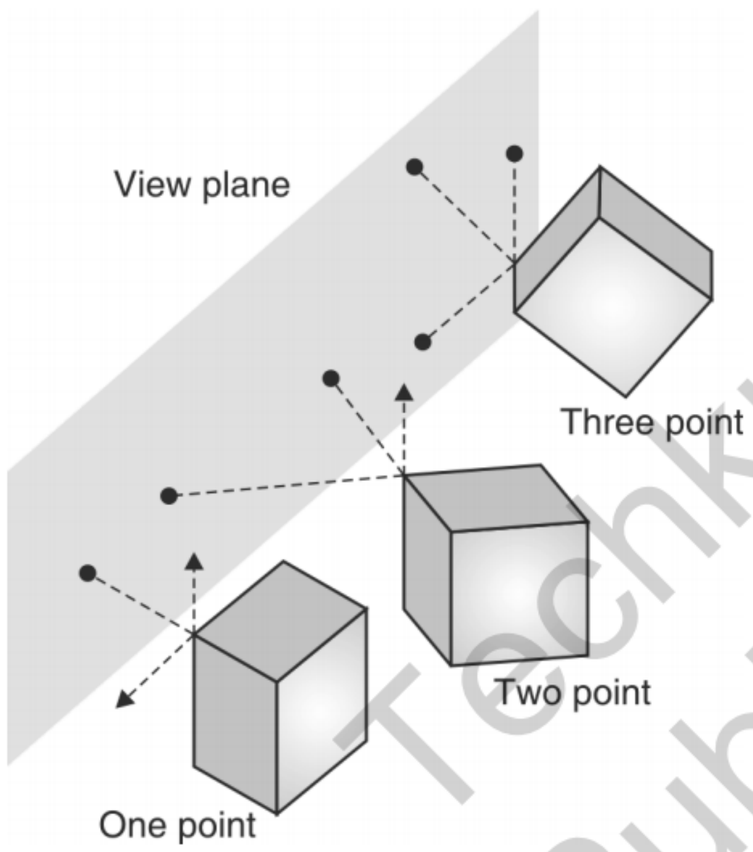


Fig. 6.2.13 : Types of perspective projection

- If view plane intersects to one, two or three principal axes, we have a single point, two points or three-point perspective projection respectively. Fig. 6.2.13 shows the orientation of the cube to achieve one, two and three-point perspective projections.
- Types of perspective projection are given based on a number of the vanishing point. There are mainly three types of perspective projections :
  1. One-point perspective projection
  2. Two-point perspective projection
  3. Three-point perspective projection

1. Single-point Perspective Projection

- One-point perspective projection is achieved by aligning view plane with two principal axes.
- In other words, the one-point perspective projection is achieved by setting up one of the principal axis perpendiculars to the view plane.
- One-point perspective projection contains only one vanishing point at the horizon. Images like straight roads, railway tracks, buildings etc. have such one-point perspective projection.
- Fig. 6.2.14 explains the one, two and three-point perspective projections.

Tech Knowledge Publications

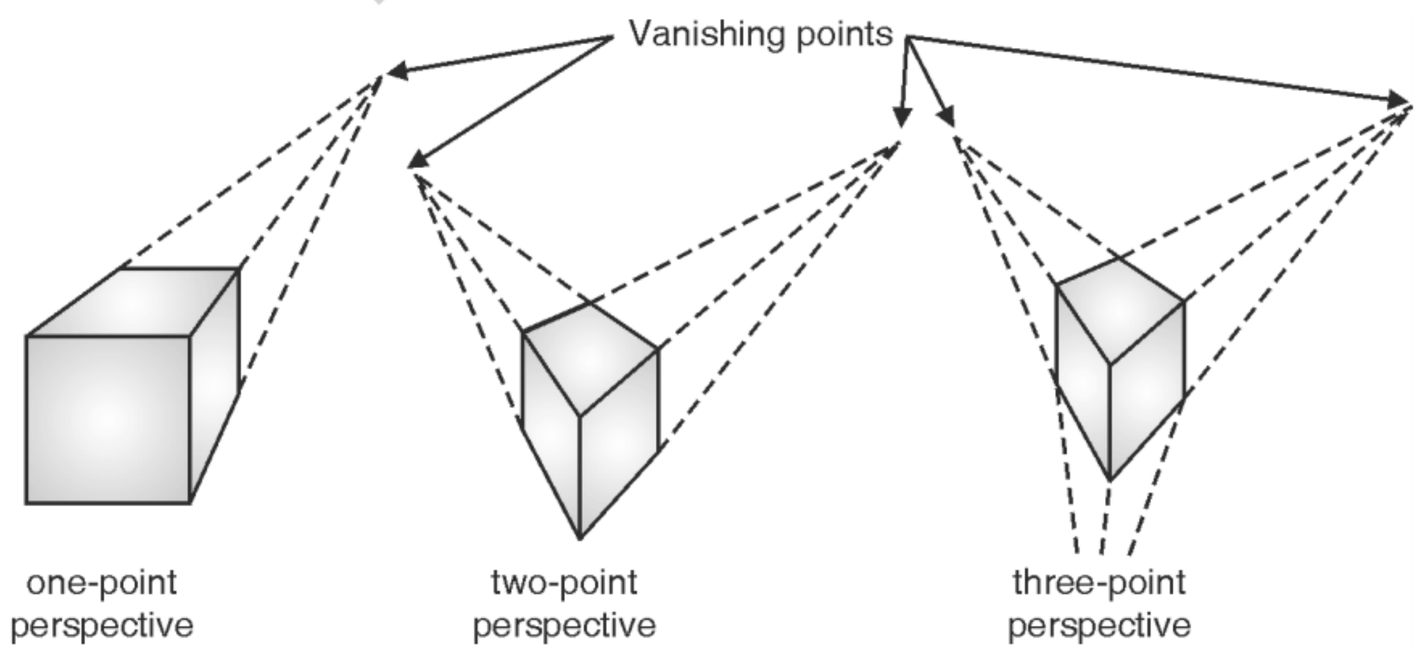


Fig. 6.2.14 : Perspective projection is achieved by projecting the result of the perspective transformation to any of the principal plane

X-direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= [x \;\; y \;\; z \;\; px + 1]$$

$$x' = \frac{x}{px + 1}$$

$$y' = \frac{y}{px + 1}$$

$$z' = \frac{z}{px + 1}$$

Y direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & q & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= [x \;\; y \;\; z \;\; qy + 1]$$

$$x' = \frac{x}{qy + 1}$$

$$y' = \frac{y}{qy + 1}$$

$$z' = \frac{z}{qy + 1}$$

Z direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= [x \;\; y \;\; z \;\; rz + 1]$$

$$x' = \frac{x}{rz + 1}$$

$$y' = \frac{y}{rz + 1}$$

$$z' = \frac{z}{rz + 1}$$

2. Two-point Perspective Projection

- Two-point perspective projection occurs when view plane is parallel to one of the principal axes or if view plane intersects exactly two principal axes.
- Two-point perspective projection is depicted in Fig. 6.2.14.
- When view plan intersects X and Y axis,
- X-direction perspective projection is given by,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= [x \;\; y \;\; z \;\; (px + qy + 1)]$$

$$x' = \frac{x}{px + qy + 1}$$

$$y' = \frac{y}{px + qy + 1}$$

$$z' = \frac{z}{px + qy + 1}$$

- In a similar way, we can derive the value of (x', y', z') for the other two cases.

3. Three-point Perspective Projection

- Three-point perspective projection happens when view plane is not parallel to any of the principal axes
- Three-point perspective projection is depicted in Fig. 6.2.14.
- For Three-point perspective projection,

$$P' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p & q & r & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= [x \;\; y \;\; z \;\; (px + qy + rz + 1)]$$

$$x' = \frac{x}{px + qy + rz + 1}$$

$$y' = \frac{y}{px + qy + rz + 1}$$

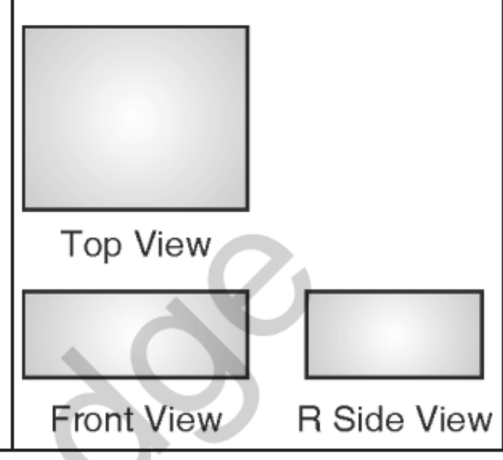
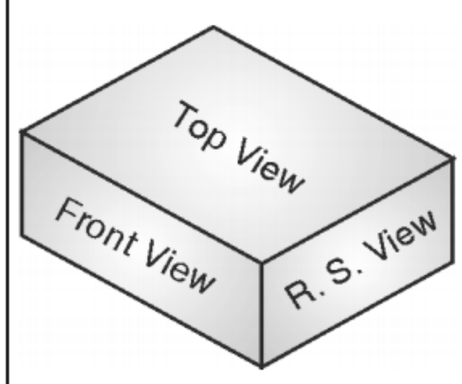
$$z' = \frac{z}{px + qy + rz + 1}$$

Techknowledge Publications

6.2.4 Difference between Parallel and Perspective Projection

Sr. No.	Parallel Projection	Perspective Projection
1.	Projectors are parallel to each other.	Projectors are not parallel.
2.	Need to specify the direction of projection.	Need to specify the center of projection.
3.	Center of projection is at infinite distance.	Center of projection is at a finite distance.
4.	Does not produce a realistic view.	Produces realistic view.
5.	Depth information is lost.	Depth information is preserved.
6.	Preserve relative proportion of object.	Does not Preserve relative proportion of object.
7.	Subtypes: Orthographic projection, oblique projection.	Subtypes: Single point, two points, three-point projection

6.2.5 Difference betweem Orthographic and Isometric Projection

Sr. No.	Orthographic Projection	Isometric Projection
1.	Provides a 2D view of the object	Provides a 3D view of the object
2.	Each view of orthographic projection shows only one side of the object	Isometric projection displays at least three sides of the object
3.	In orthographic projection, the projection plane is parallel to one of the principal plane	In isometric projection, the projection plane is not parallel to any of the principal plane
4.	Does not preserve depth	Dose include depth
5.	True shape and size of an object is preserved	The projected object is foreshortened equally in all three directions.
6.	Example : 	Example : 

6.2.6 Solved Examples

Ex. 6.2.1 : Derive the transformation matrix for parallel projection of point P (x, y, z) on XY plane in direction $V = a_i + b_j + c_k$.

Soln. :

$V = a_i + b_j + c_k$ is the direction of projection. Let us assume that parallel projection of point P (x, y, z) on XY plane in direction of V is $P'(x', y', z')$.

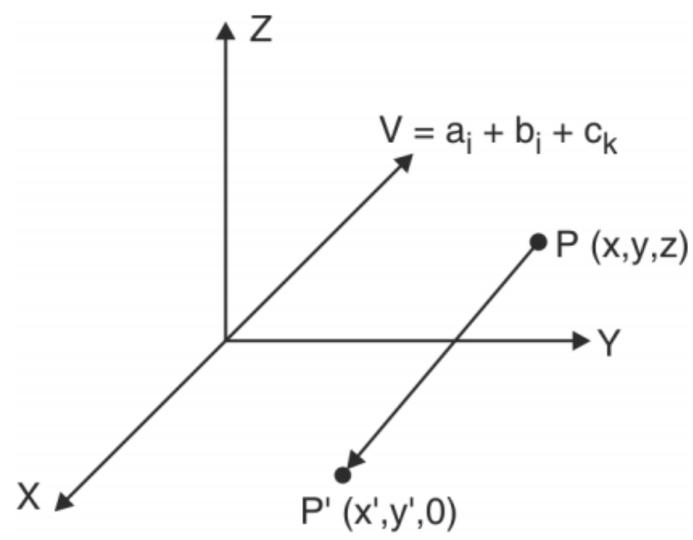


Fig. P. 6.2.1

Projection is in XY plane, so $z' = 0$.

Directions of V and PP' are same. So,

$$\overrightarrow{PP'} = k \cdot \vec{V}$$

$$((x' - x), (y' - y), (z' - z)) = k \cdot (a, b, c)$$

$$x' = x + k \cdot a$$

$$y' = y + k \cdot b$$

Technknowledge Publication

$$z' = z + k \cdot c$$

but, $$z' = 0$$

$$\therefore \quad z + k \cdot c = 0$$

$$\therefore \quad k = -\frac{z}{c}$$

$$\therefore \quad x' = x - \frac{a}{c} \cdot z$$

$$\therefore \quad y' = y - \frac{b}{c} \cdot z$$

Transformation matrix for this projection would be,

$$M = \begin{bmatrix} 1 & 0 & -a/c & 0 \\ 0 & 1 & -b/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrix representation of transformed coordinates is given as,

$$P' = M \cdot P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a/c & 0 \\ 0 & 1 & -b/c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Ex. 6.2.2 : Derive value of rotation angle ϕ and θ for diametric and trimetric axonometric projection.

Soln. : In axonometric projection, at least three faces of an object must be visible. This can be achieved by a first rotating object about Y-axis by angle followed by a rotation about X-axis and then taking projection on z = 0 plane.

The transformation matrix for this would be,

$$M = P_z \cdot P_x \cdot P_y(\phi)$$

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

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

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

Foreshortening ratios are obtained by multiplying a unit vector with the transformation matrix

$$U = M \cdot I = \begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ \sin\theta \cdot \sin\phi & \cos\theta & -\sin\theta \cdot \cos\phi & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ \sin\theta \cdot \sin\phi & \cos\theta & -\sin\theta \cdot \cos\phi \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

TechKnowledge Publications

$$f_x = \sqrt{\cos^2\theta + \sin^2\theta \cdot \sin^2\phi} \quad \ldots(1)$$

$$f_y = \sqrt{\cos^2\theta} \quad \ldots(2)$$

$$f_z = \sqrt{\sin^2\theta + \sin^2\theta \cdot \cos^2\phi} \quad \ldots(3)$$

Where f_x, f_y and f_z defines foreshortening factors in x, y and z-direction respectively.

Foreshortening Factor for Diametric Projection

In diametric projection, foreshortening factor for any two axes are same, so suppose $f_x = f_y$.

$$\therefore \cos^2\theta = \cos^2\phi + \sin^2\theta \cdot \sin^2\phi$$

$$\therefore 1 - \sin^2\theta = 1 - \sin^2\phi + \sin^2\theta \cdot \sin^2\phi$$

$$\sin^2\phi - \sin^2\theta \sin^2\phi = \sin^2\theta$$

$$\sin^2\phi\,(1 - \sin^2\theta) = \sin^2\theta$$

$$\sin^2\phi = \frac{\sin^2\theta}{1 - \sin^2\theta} \quad \ldots(4)$$

From Equation (3),

$$f_z^2 = \sin^2\phi + \sin^2\theta \cdot \cos^2\phi$$

$$= \sin^2\phi + \sin^2\theta \cdot (1 - \sin^2\phi)$$

$$= \sin^2\phi + \sin^2\theta - \sin^2\theta \cdot \sin^2\phi$$

Putting Equation (4) in this,

$$f_z^2 = \left(\frac{\sin^2\theta}{1 - \sin^2\theta}\right) + \sin^2\theta - \sin^2\theta \left(\frac{\sin^2\theta}{1 - \sin^2\theta}\right)$$

$$\therefore f_z^2\,(1 - \sin^2\theta) = \sin^2\theta + \sin^2\theta\,(1 - \sin^2\theta) - (\sin^4\theta)$$

$$\therefore f_z^2 - f_z^2 \sin^2\theta = \sin^2\theta + \sin^2\theta - \sin^4\theta - \sin^4\theta$$

$$\therefore 2\sin^4\theta - \sin^2\theta\,(f_z^2 + 2) - f_z^2 = 0$$

Solving it for quadratic equation

$$ax^2 + bx + c = 0\,,\ a = 2,\ b = -(f_z^2 + 2),\ c = f_z^2\,,$$

$$x = \sin^2\theta$$

$$\Delta = b^2 - 4ac$$

$$= (f_z^2 + 2)^2 - 4(2)\,(f_z^2)$$

$$= f_z^2 + 4 f_z^2 + 4 - 8 f_z^2$$

$$= f_z^2 - 4 f_z^2 + 4 = (f_z^2 - 2)^2$$

$$r_1, r_2 = \frac{-b + \sqrt{\Delta}}{2a}$$

$$= \frac{(-f_z^2 + 2) \pm \sqrt{(f_z^2 - 2)^2}}{4}$$

$$r_1 = \frac{f_z^2 + 2 + f_z^2 - 2}{4} = \frac{f_z^2}{2}$$

$$r_2 = \frac{f_z^2\, z^2 + 2 - f_z^2 + 2}{4} = 1$$

But from Equation (4), $\sin^2\theta = 1$ is not possible

$$\therefore \sin^2\theta = \frac{f_z^2}{2}$$

$$\therefore \theta = \sin^{-1}\left(\pm\frac{f_z}{\sqrt{2}}\right)$$

From Equation (4),

$$\sin^2\phi = \frac{\sin^2\theta}{1 - \sin^2\theta} = \frac{\frac{f_z^2}{2}}{1 - \frac{f_z^2}{2}} = \frac{f_z^2}{2 - f_z^2}$$

$$\phi = \sin^{-1}\left(\pm\frac{f_z}{\sqrt{2 - f_z^2}}\right)$$

Foreshortening Factor for Isometric Projection

In isometric projection, foreshortening factor in all three dimensions must be same.

$$f_x = f_z = f_z$$

From Equation (2) and (3),

$$f_y = f_z$$

$$\cos^2\theta = \sin^2\phi + \cos^2\phi \cdot \sin^2\theta$$

$$1 - \sin^2\theta = \sin^2\phi + (1 - \sin^2\phi) \cdot \sin^2\theta$$

$$1 - \sin^2\theta = \sin^2\phi + \sin^2\theta - \sin^2\phi \cdot \sin^2\theta$$

$$1 - 2\sin^2\theta = \sin^2\phi\ (1 - \sin^2\theta)$$

$$\therefore \sin^2\phi = \frac{1 - 2\sin^2\theta}{1 - \sin^2\theta} \quad \ldots(5)$$

By comparing R.H.S of Equation (4) and (5),

$$\frac{\sin^2\theta}{1 - \sin^2\theta} = \frac{1 - 2\sin^2\theta}{1 - \sin^2\theta}$$

$$\therefore \sin^2\theta = 1 - 2\sin^2\theta$$

$$\therefore \sin^2\theta = \frac{1}{3}$$

Tech Knowledge Publications

$$\therefore \quad \theta = \sin^{-1}\left(\pm\frac{1}{\sqrt{3}}\right)$$

$$= \pm 35.26$$

From Equation (4),

$$\sin^2\phi = \frac{\sin^2\theta}{1-\sin^2\theta}$$

$$= \frac{1/3}{1-1/3} = \frac{1/3}{2/3} = \frac{1}{2}$$

$$\therefore \quad \phi = \sin^{-1}\left(\pm\frac{1}{2}\right) \pm 45°$$

$$\therefore \quad f_y = f_z$$

$$\therefore \quad f_z = \sqrt{\cos^2\theta}$$

$$= \sqrt{1-\sin^2\theta} = \sqrt{1-\frac{1}{3}}$$

$$= \frac{\sqrt{2}}{3} = 0.8165$$

Thus isometric is special case of diametric projection with $f_x = f_y = f_z = 0.8165$.

For trimetric projection, foreshortening factor for all three directions is different. Trimetric projection is less restrictive.

Ex. 6.2.3 : Find transformed co-ordinate of a unit cube with one vertex at origin for,

(i) Cavalier projection with $\phi = 45°$

(ii) Cabinet projection with $\phi = 30°$

Soln. :

Let us assume that vertex A of unit cube is positional at the origin.

From Fig. P. 6.2.3,

$$A = (0, 0, 0)$$

$$B = (1, 0, 0)$$

$$C = (1, 1, 0)$$

$$D = (0, 1, 0)$$

$$E = (0, 0, 1)$$

$$F = (1, 0, 1)$$

$$G = (1, 1, 1)$$

$$H = (0, 1, 1)$$

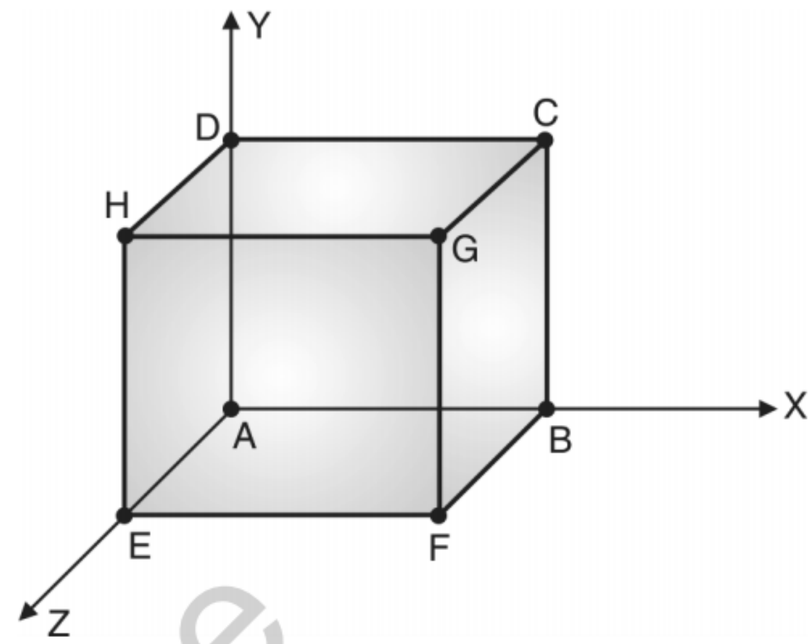


Fig. P. 6.2.3

(i) Cavalier Projection

For cavalier projection, foreshortening factor $L_1 = 1$

Transformation matrix for cavalier and cabinet projection is,

$$M = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{With } \phi = 45° \text{ and } L_1 = 1$$

$$M = \begin{bmatrix} 1 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformed points are computed as,

$$P' = M \cdot P$$

$$M = \begin{bmatrix} 1 & 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 1 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Techknowledge Publications

By solving above matrices,

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	A′ (0, 0, 0)
B(1, 0, 0)	B′ (1, 0, 0)
C(1, 1, 0)	C′ (1, 1, 0)
D(0, 1, 0)	D′ (0, 1, 0)
E(0, 0, 1)	$E'\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)$
F(1, 0, 1)	$F'\left(\frac{\sqrt{2}+1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0\right)$
G(1, 1, 1)	$G'\left(\frac{\sqrt{2}+1}{\sqrt{2}}, \frac{\sqrt{2}+1}{\sqrt{2}}, 0\right)$
H(0, 1, 1)	$H'\left(\frac{1}{\sqrt{2}}, \frac{\sqrt{2}+1}{\sqrt{2}}, 0\right)$

(ii) Cabinet Projection

For cabinet projection, foreshortening factor,

$$F = L_1 = \frac{1}{2}$$

Given angle $\phi = 30°$

$$\text{So, } M = \begin{bmatrix} 1 & 0 & L_1\cos\phi & 0 \\ 0 & 1 & L_1\sin\phi & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{\sqrt{3}}{4} & 0 \\ 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformed coordinates of the cube are computed as,

$$P' = M \cdot P$$

$$= \begin{bmatrix} 1 & 0 & \frac{\sqrt{3}}{4} & 0 \\ 0 & 1 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

By solving the above matrices,

Original Coordinates	Transformed Coordinates
A(0, 0, 0)	A′ (0, 0, 0)
B(1, 0, 0)	B′ (1, 0, 0)
C(1, 1, 0)	C′ (1, 1, 0)
D(0, 1, 0)	D′ (0, 1, 0)
E(0, 0, 1)	$E'\left(\frac{\sqrt{3}}{4}, \frac{1}{4}, 0\right)$
F(1, 0, 1)	$F'\left(\frac{4+\sqrt{3}}{4}, \frac{1}{4}, 0\right)$
G(1, 1, 1)	$G'\left(\frac{4+\sqrt{3}}{4}, \frac{5}{4}, 0\right)$
H(0, 1, 1)	$H'\left(\frac{\sqrt{3}}{4}, \frac{5}{4}, 0\right)$

Ex. 6.2.4 : Derive a transformation matrix for perspective projection when the center of projection is at origin and plane of projection is at distance d on the Z-axis.

Soln. :

Spatial orientation of the viewplane is shown in Fig. P. 6.2.4.

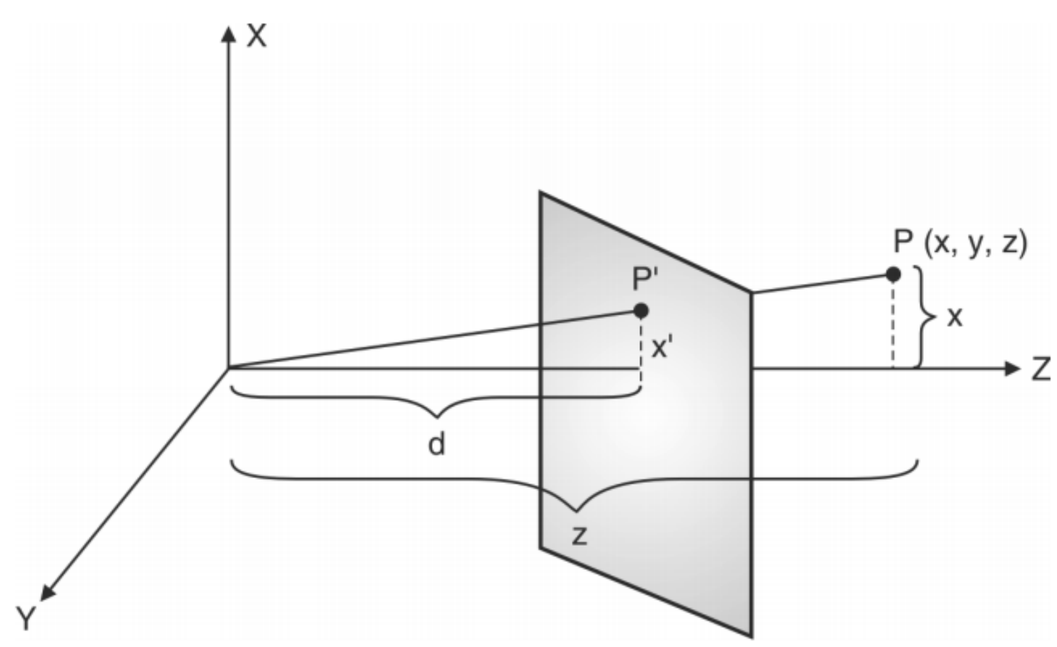


Fig. P. 6.2.4

Suppose projection of point P (x, y, z) on given view plane is P′ (x′, y′, z′).

Technknowledge Publication

From the rule of triangle equality,

$$\frac{x}{z} = \frac{x'}{d}$$

$$\therefore \quad x' = \frac{x}{z} \cdot d$$

Similarly, $\frac{y}{z} = \frac{y'}{d}$

$$\therefore \quad y' = \frac{y}{z} \cdot d$$

And z′ would be d because view plane is on Z-axis at distance d.

$$\therefore \quad z' = d$$

Transformation matrix would be,

$$M = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P' = M \,.\, P$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & d & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Ex. 6.2.5 : Using origin as the center of projection, derive transformation matrix for perspective projection on a view plane passing through reference point R_0 (x_0, y_0, z_0) and having normal vector $\overline{N} = n_1 i + n_2 j + n_3 k$.

Soln. :

Spatial orientation of the viewplane is shown in Fig. P. 6.2.5.

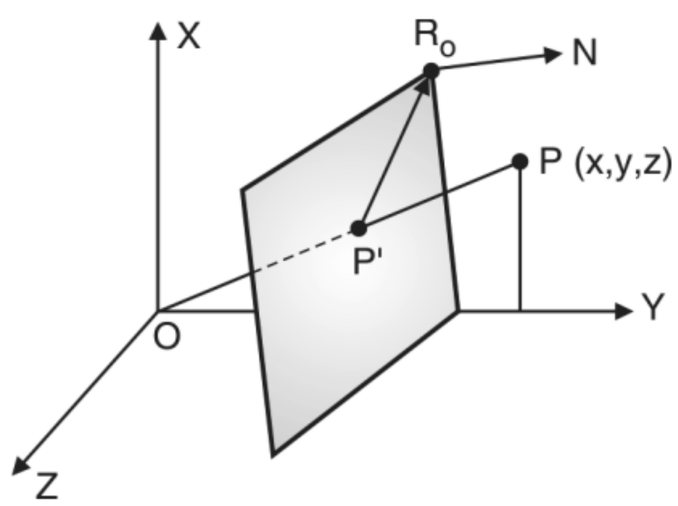


Fig. P. 6.2.5

Let us assume that projection of point P (x, y, z) on the given view plane is P′ (x′, y′, z′).

From Fig. P. 6.2.5, OP′ and OP both vectors have the same direction but different magnitude.

$$\therefore \quad \overrightarrow{OP'} = \alpha \overrightarrow{OP}$$

$$(P' - O) = \alpha (P - O)$$

$$\therefore \quad (x', y', z') = \alpha (x, y, z)$$

$$x' = \alpha x \,;\; y' = \alpha y \,;\; z' = \alpha z$$

We need to find out α to find projected co-ordinates.

R_0 and P′ are on the view plane and $\overline{N}$ is normal to the plane. The dot product of two perpendicular vectors is always zero. So,

$$(\overrightarrow{R_0 P'}) \cdot \overline{N} = 0$$

$$(P' - R_0) \cdot \overline{N} = 0$$

$$(x', y', z') - (x_0, y_0, z_0) \cdot (n_1, n_2, n_3) = 0$$

$$(x' - x_0)\, n_1 + (y' - y_0)\, n_2 + (z' - z_0)\, n_3 = 0$$

$$n_1 x' + n_2 y' + n_3 z' = n_1 x_0 + n_2 y_0 + n_3 z_0$$

Put, $x' = \alpha x$, $y' = \alpha y$ and $z' = \alpha z$ in above equation.

$$n_1 \alpha x + n_2 \alpha y + n_3 \alpha z = n_1 x_0 + n_2 y_0 + n_3 z_0$$

$$\alpha (n_1 x + n_2 y + n_3 z) = n_1 x_0 + n_2 y_0 + n_3 z_0$$

$$\alpha = \frac{n_1 x_0 + n_2 y_0 + n_3 z_0}{n_1 x + n_2 y + n_3 z}$$

$$\text{Let, } d_0 = n_1 \cdot x_0 + n_2 \cdot y_0 + n_3 \cdot z_0$$

$$\alpha = \frac{d_0}{n_1 x + n_2 y + n_3 z}$$

$$x' = \alpha x = \frac{d_0}{n_1 x + n_2 y + n_3 z} \,.\, x$$

$$y' = \alpha y = \frac{d_0}{n_1 x + n_2 y + n_3 z} \,.\, y$$

Techknowledge Publication

$$z' = \alpha z = \frac{d_0}{n_1 x + n_2 y + n_3 z} \cdot z$$

Transformation matrix for $P' = M \cdot P$ would be,

$$M = \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & 0 \\ n_1 & n_2 & n_3 & 0 \end{bmatrix}$$

Ex. 6.2.6 : Derive a general perspective transformation matrix for projection on a a plane passing from reference point R_0 (x_0, y_0, z_0) having a normal vector $N = n_1 i + n_2 j + n_3 k$. Consider centre of projection at C (a, b, c).

Soln. :

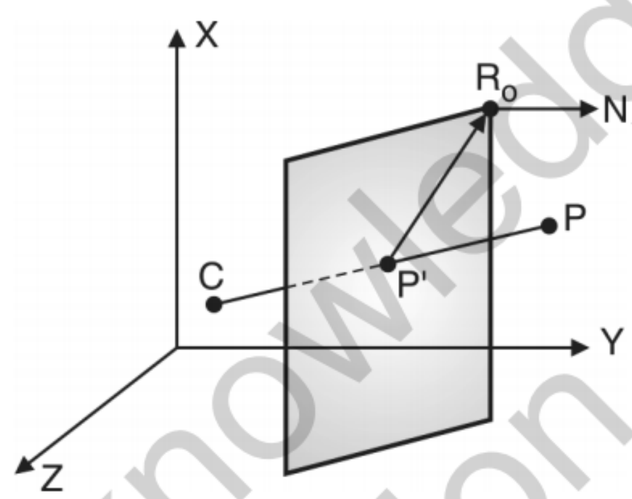


Fig. P. 6.2.6

Spatial orientation of the view plane is shown in Fig. P. 6.2.6.

Coordinate of center of projection is C(a, b, c). Let the point P be (x, y, z) and its projection on view plane is P′ (x′, y′, z′).

From Fig. P. 6.2.6,

$$\overrightarrow{CP'} = \alpha \overrightarrow{CP}$$

$$(P' - C) = \alpha (P - C)$$

$$(x', y', z') - (a, b, c) = \alpha \{ (x, y, z) - (a, b, c)\}$$

$$(x' - a, y' - b, z' - c) = \alpha (x - a, y - b, z - c)$$

$$x' = a + \alpha (x - a)$$

$$y' = b + \alpha (y - b)$$

$$z' = c + \alpha (z - c)$$

Dot product of two perpendicular vectors is zero.

$$\therefore \ (\overrightarrow{R_0 P'}) \cdot N = 0$$

$$(P' - R_0) \cdot N = 0$$

$$\{(x', y', z') - (x_0, y_0, z_0)\} \cdot (n_1, n_2, n_3) = 0$$

$$(x' - x_0)\, n_1 + (y' - y_0)\, n_2 + (z' - z_0)\, n_3 = 0$$

$$x'\, n_1 + y'\, n_2 + z'\, n_3 = n_1\, x_0 + n_2\, y_0 + n_3\, z_0$$

Replacing value of x′, y′ and z′

$$(a + \alpha\,(x - a))\, n_1 + (b + \alpha\,(x - b))\, n_2 + (c + \alpha\,(x - c)) = n_1\, x_0 + n_2\, y_0 + n_3\, z_0$$

Rearranging the terms,

$$\alpha\,(n_1\,(x - a) + n_2\,(y - b) + n_3\,(z - c)) = (n_1\, x_0 + n_2\, y_0 + n_3\, z_0) - (a \cdot n_1 + b \cdot n_2 + c \cdot n_3)$$

$$\text{Let us take } d_0 = n_1\, x_0 + n_2\, y_0 + n_3 z_0$$

$$\text{and, } d_1 = n_1 a + n_2\, b + n_3 c$$

$$\alpha = \frac{d_0 - d_1}{(x - a) \cdot n_1 + (y - b) \cdot n_2 + (z - c) \cdot n_3}$$

And from previous comparison,

$$x' = (x - a) \cdot \alpha + a$$

$$y' = (y - b) \cdot \alpha + b$$

$$z' = (z - c) \cdot \alpha + c$$

From this, transformation matrix is written as,

$$M = \begin{bmatrix} d_0 + an_1 & an_2 & an_3 & (d_0 - a) + a\,((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ bn_1 & d_0 + bn_2 & bn_3 & (d_0 - b) + b\,((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ cn_1 & cn_2 & d_0 + cn_3 & (d_0 - c) + c\,((n_1 - a) + (n_2 - b) + (n_3 - c)) \\ n_1 & n_2 & n_3 & (n_1 - a) + (n_2 - b) + (n_3 - c) \end{bmatrix}$$

Second Method

1. Translate C to the origin.
2. This is identical to the previous case. Use that transformation matrix.
3. Inverse translation of C.

$$M = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} d_0 & 0 & 0 & 0 \\ 0 & d_0 & 0 & 0 \\ 0 & 0 & d_0 & c \\ n_1 & n_2 & n_3 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tech Knowledge Publications

Review Questions

Q. 1 How to achieve 3D translation?

Q. 2 Derive 3D transformation matrix for translation operation.

Q. 3 How to achieve 3D rotation?

Q. 4 Derive 3D transformation matrix for various cases of 3D rotation.

Q. 5 Derive 3D transformation matrix for rotation about a principal axis.

Q. 6 How to perform 3D rotation about a line parallel to the principal axis?

Q. 7 Derive transformation matrix for 3D rotation about a line parallel to the principal axis.

Q. 8 Derive transformation matrix for rotation about an arbitrary axis in space.

Q. 9 Write matrices in the homogeneous coordinate system for 3D scaling transformation.

Q. 10 How to achieve 3D scaling?

Q. 11 Derive 3D transformation matrices for scaling with respect to the origin and with respect to a reference point.

Q. 12 Derive 3D transformation matrices for scaling with respect to the origin.

Q. 13 Derive 3D transformation matrices for scaling with respect to a reference point.

Q. 14 What is the projection? Why projection is necessary ?

Q. 15 Define the terms : Projection, view plane, Direction of projection, the center of projection.

Q. 16 Enlist the types of projection.

Q. 17 Write a short note on parallel projection.

Q. 18 Explain orthographic parallel protection.

Q. 19 Explain in detail Oblique projection.

Q. 20 Write a short note on cavalier protection.

Q. 21 Write a short note on cabinet protection.

Q. 22 Write a short note on perspective projection.

Q. 23 Derive a perspective projection of point P (x, y, z) on a view plane positioned at z = 0 and center of projection is on negative z-axis at distance d.

Q. 24 Write a short note on one-point perspective projection.

Q. 25 Write a short note on Two-point perspective projection.

Q. 26 Write a short note on Three-point perspective projection

Q. 27 Differentiate : Parallel projection vs. Perspective projection.

Q. 28 Explain types of Parallel Projection with example.

Technknowledge Publication

❑❑❑

CHAPTER 7

UNIT IV

Color Models

Syllabus

Color Models : Properties of Light, CIE chromaticity Diagram, RGB, HSV, CMY.

7.1 Properties of Light

Q.	Explain the term with an example - Color gamut.	**SPPU - Dec. 17, 1 Mark**
Q.	Explain Properties of Light.	**SPPU - May 18, 3 Marks**
Q.	Write short note on Properties of light.	**SPPU - Dec. 19, 2 Mark**

– Light is an electromagnetic wave. Waves are defined in term of frequency (f) or wavelength (λ). Visible rays - part of the electromagnetic spectrum - define the notion of color.

– Various components of the EM spectrum and their respective frequency and wavelength are shown in Fig. 7.1.1.

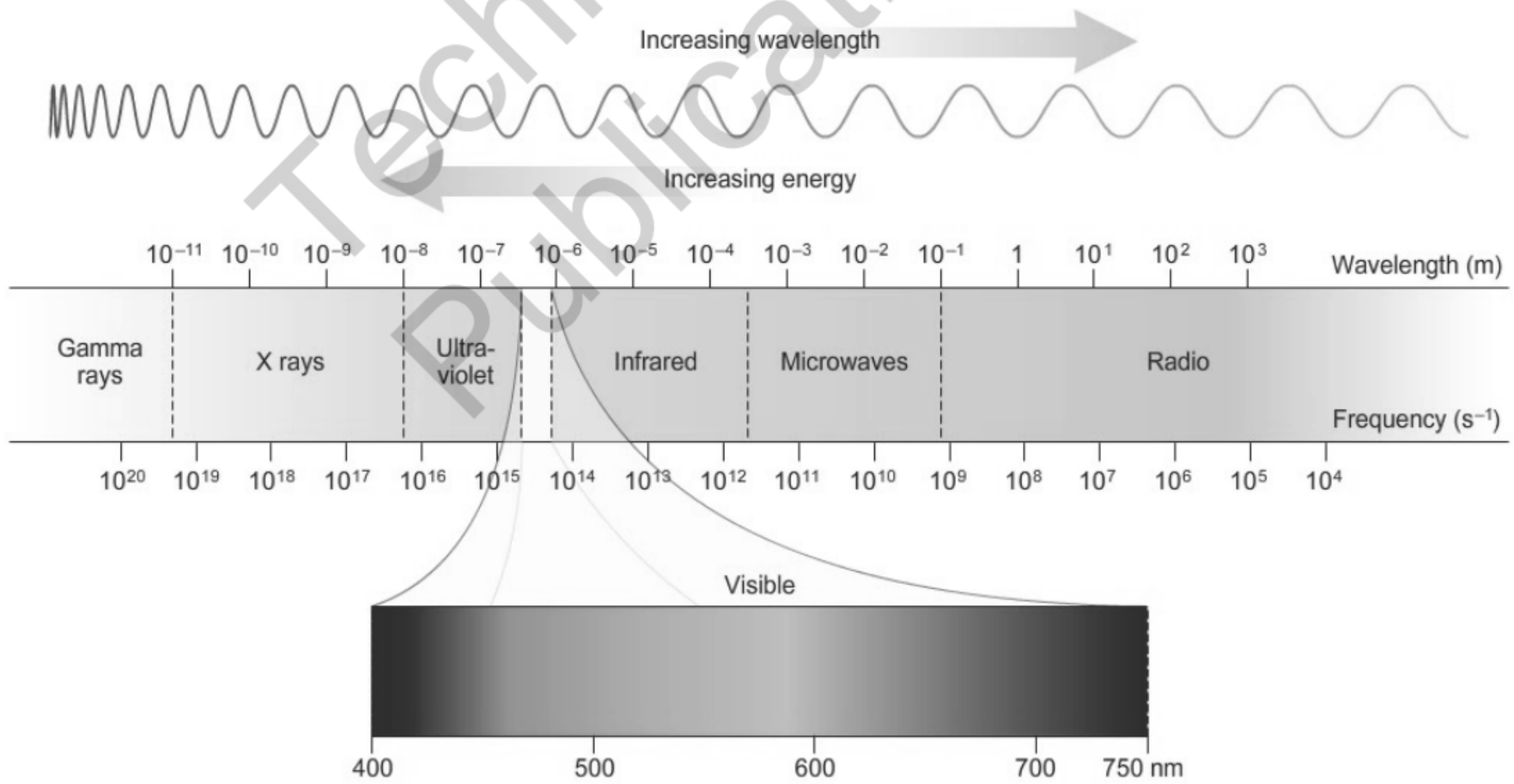


Fig. 7.1.1 : Electro-magnetic spectrum and its various components

– Rays having a wavelength between 400 nm to 750 nm are visible. Color of the object depends on the wavelength of the rays being reflected from the surface of the object.

– Color is a subjective interpretation of the brain, which is ultimately a combination of light of a certain wavelength.

Technical Knowledge Publication

- Color perceived by human eye depends not only on the wavelength of light but it also depends on what wavelength is absorbed by an object or reflected by the object.
- Light plays a significant role in rendering realistic images. With the help of various lighting model, we can simulate the various features of the scene like reflection, shadow, refraction etc.
- Light adds variations in shading and creates more interesting images. That is why photographers and film director seriously put efforts in the proper light setting.
- Light interacts with the material in four different ways. An object can emit the light (Emission), an object can absorb the light (Absorption), the light might bounce off the object (Reflection) or light might pass through the object (Refraction/ Transmission).
- Color of the object in the scene depends on a mixture of reflected frequency from the surface of the object. If the surface reflects lower frequency than object appears reddish. This *dominant lower frequency* decides the color and ultimately **hue** of the color.
- How pure color should appear is decided by the **saturation**. And how bright the scene should appear is decided by the **brightness** parameter.
- Purity is proportional to saturation. When hue and saturation are collectively used to describe the color, it is called **chromaticity**.
- When two different colors are combined in the right proportion they produce neutral color like white, such pair of colors is called **complementary colors**. Red and cyan, green and magenta, blue and yellow are complementary colors.
- By varying proportion of different colors, we can produce a range of colors. By keeping constant hue, and varying saturation, we get different shades of the same color.
- Color gamut is the set of all the visible colors to the human eye. Since the display device can display certain colors only, the color gamut of the display device is specified by the color models. Basic colors used to create color gamut are called **primary color** for that color model.
- In computer system, red, green and blue colors are combined with various intensity to produce all the colors.
- System supporting 24 bit per pixel can produce 17 million different colors using three basic colors red, green and blue.

7.2 CIE Chromaticity Diagram

Q. Explain the CIE chromaticity diagram.

SPPU - May 17, Dec. 18, 4 Marks

Q. Write a short note on CIE chromaticity.

SPPU - May 18, 6 Marks

- The CIE XYZ color space was derived from a series of experiments done by William David and John Guild. Results of their experiments were combined into CIE RGB color space, from which the CIE XYZ color space was derived. CIE XYZ color model is the first mathematically defined color model, which was created by the International Commission on Illumination, in 1931.
- The human eye has three kinds of cone cells which recognize the red, green and blue colors in high brightness condition.
- In low brightness, the vision of cones diminishes and rod cells take over. These three cones have different sensitivity and these three values make tristimulus specification. These three parameters called L, M and S can be represented using three-dimensional space called *LMS color model*.

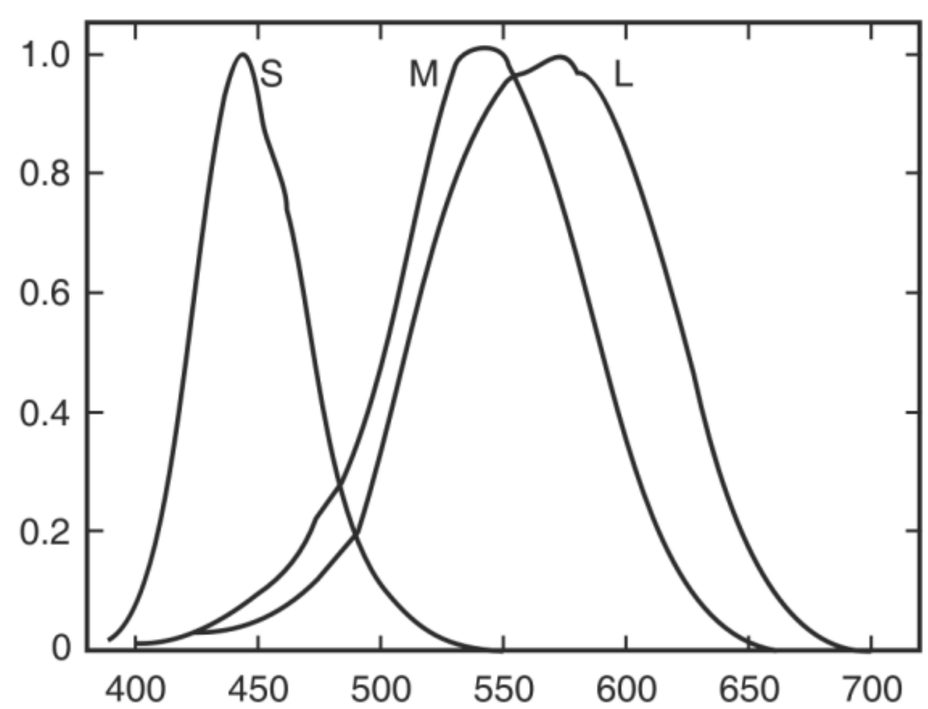


Fig. 7.2.1 : LMS color model

TechKnowledge Publications

– In well light condition, the human eye tends to perceive light in the green part of the spectrum brighter than that of the red and blue part of equal power. The perceived brightness of the different color is roughly analogues to the spectral sensitivity of M cones. CIE standard has modelled Y as luminance and Z as the response of cone. X is a mixture of the cone response curve. Thus the XYZ model is analogues to the LMS model. Any color C can be defined in this model as follow :

$$C = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$$

Where X, Y and Z represents a vector in three-dimensional color space, and **X**, **Y**, **Z** represents the amount of standard primary colors of the model.

– Sometimes XYZ colors are represented by the luminance Y, and chromaticity coordinates x and y, defined by;

$$x = \frac{X}{X+Y+Z},$$

$$y = \frac{Y}{X+Y+Z},$$

$$z = \frac{Z}{X+Y+Z}$$

Where, $x + y + z = 1$

– Parameter x and y are called chromaticity as they depend only on hue and saturation. Generally, color is described with three parameters, x, y and Y. Given x, y and Y, we can derive values of other CIE components as below.

$$X = \frac{x}{y}Y \text{ and } Z = \frac{z}{y}Y$$

– If we plot x and y for all visible colors, we get CIE chromaticity diagram as shown in Fig. 7.2.2.

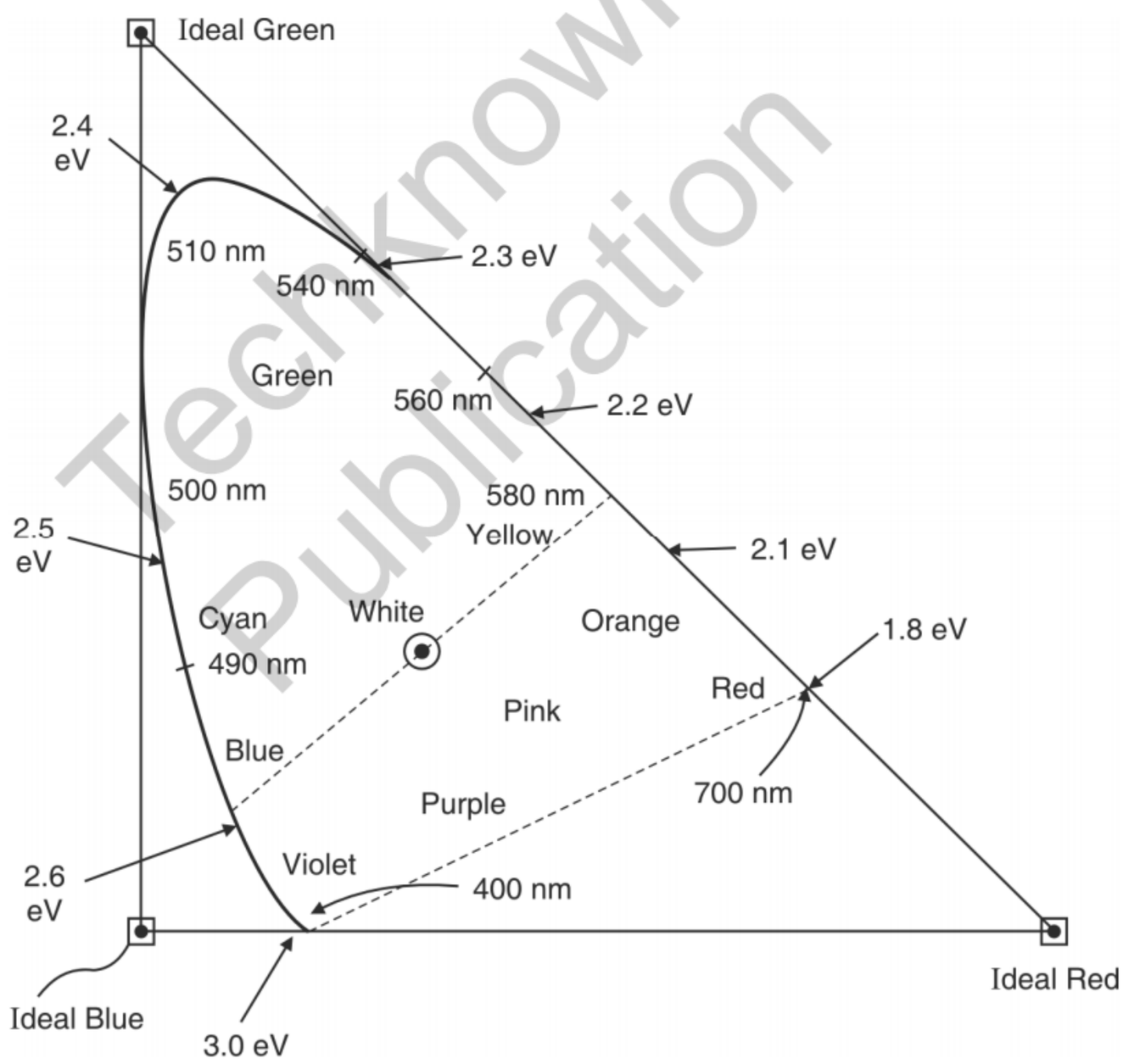


Fig. 7.2.2 : Chromaticity diagram

– Points on the boundary of the tongue-shaped diagram represent pure colors. White color is at the center. The line joining two ends red and violet is known as the purple line, which is not part of color spectrum. CIE diagrams help in many ways.

  - We can find the hue and saturation of any color by matching the color with CIE primaries.

Technknowledge Publication

- It's simple to find complementary color.
- Easy to define color gamut.

– When we add the proper amount of two colors, they produce a white color. Such colors are called the complementary color of each other.

– Color gamuts are represented by a straight line or closed polygon on the chromaticity diagram.

– When we join two colors by a straight line, we get all possible color that can be produced with varying amount of both the colors. Color gamut for three colors is defined by the triangle as shown in Fig. 7.2.3. Triangle covers all the possible colors that can be produced with a mixture of all three colors. Colors outsides the triangle cannot be produced using the same three colors. The gamut of colors must be convex in shape. All colors that can be formed by mixing three colors are found inside the triangle formed by the color points on the chromaticity diagram.

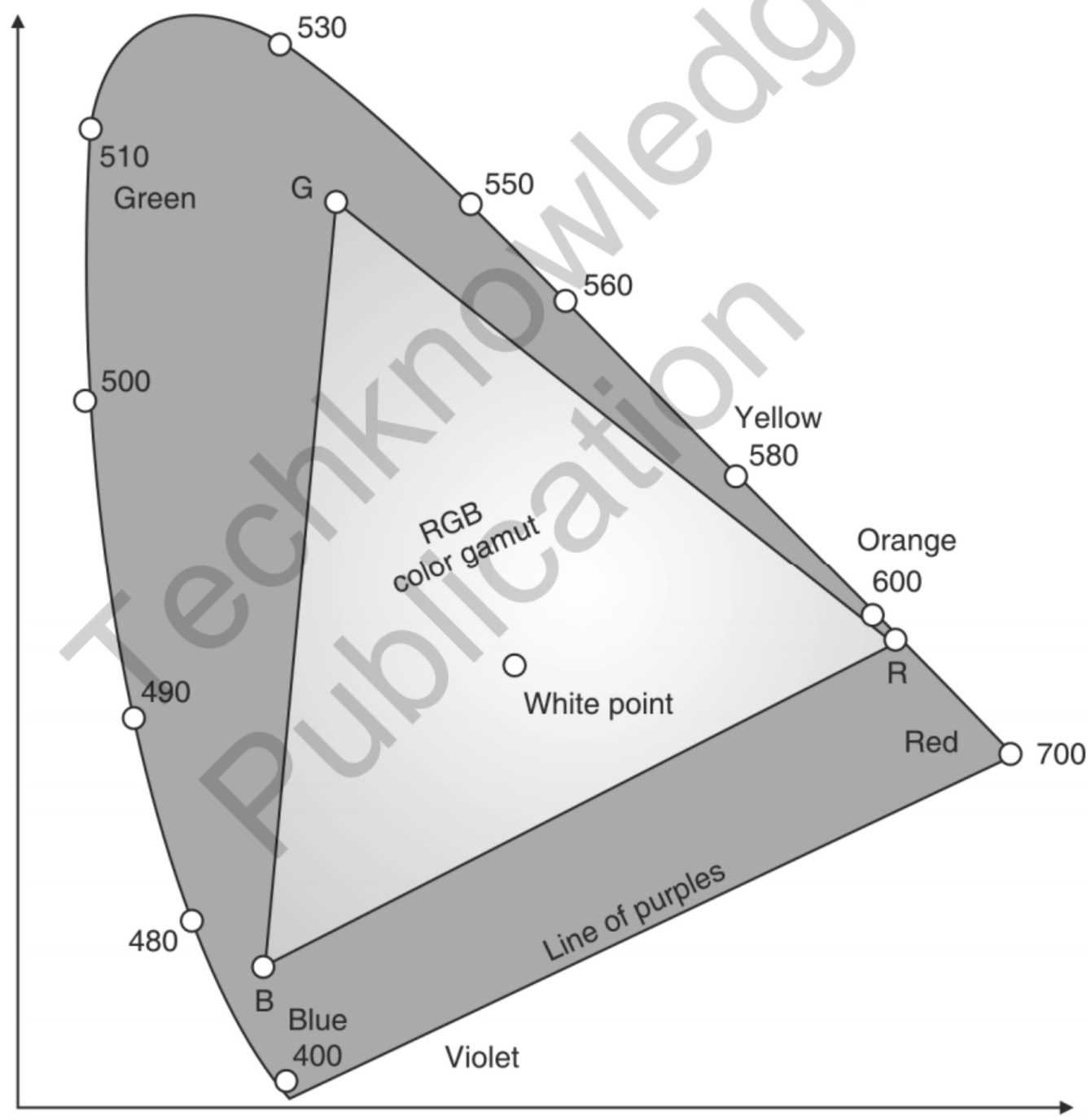


Fig. 7.2.3 : Color Gamut

– Chromaticity diagram represents all the possible color that the human eye can perceive. But it can be seen that given any three colors they cannot cover the gamut of human vision. You cannot define the gamut using three points which form a triangle that includes the entire tongue shape.

– We can even find a dominant wavelength of a given color. By extending the line from white color and passing through given color intersects tongue shape at some point is boundary color. That color is the pure hue of the given color.

– We cannot find the dominant wavelength of the color between white color and purple line, because the purple line is not part of the chromaticity diagram.

7.3 Color Models

Color model is a system to create a wide range of colors from small set of primary colors. Color models are typically represented using some coordinate system, which allows easy mathematical representation of the color. There are two type of color models: additive and subtractive. In this section, we will discuss some popular color models.

7.3.1 RGB Color Model

Q. Explain the RGB color model.

SPPU - Dec. 15, May 16, May 17, May 19, 4 Marks

– RGB color model is an additive color model. It adds red, green and blue color to produce a wide range of colors. RGB color model is used to define colors in monitor, television system, mobile phones, video projectors etc.

– However, it is not suitable for image processing applications. RGB is a device-dependent color model. Since the color elements like phosphors and their response to the individual R, G, and B levels vary from manufacturer to manufacturer, different devices reproduce the same RGB value differently.

– We can define this color model as a unit cube with R, G and B as its three axes. Zero intensity of all three component produce black color and full intensity of them produce the white color. An equal proportion of all three colors between zero to one produce grey shade (diagonal line joining origin and opposite cube corner).

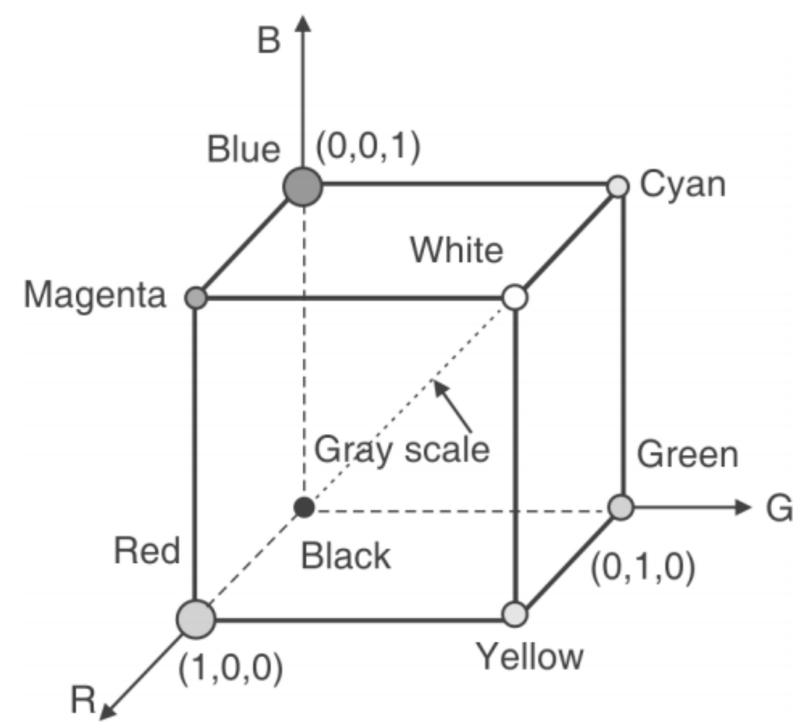


Fig. 7.3.1 : RGB color model

– Vertices on axis represent primary colors and remaining vertices represent the complementary colors.

– When anyone component has a dominant intensity, the color is a hue of primary color, and when two components have the same dominant intensity, then the color is a hue of a secondary color. A secondary color is formed by the sum of two primary colors of equal intensity.

– For example, cyan can be produced by adding blue and green. When a primary and its complementary secondary color are added together, the result is white. The complement of red is cyan, the complement of green is magenta and the complement of green is yellow.

– RGB is an additive color model and any color C can be obtained by addition of R, G and B.

$$C = R\mathbf{R} + G\mathbf{G} + B\mathbf{B}$$

– Value of R, G and B is between 0 and 1. Zero value of all three components produces black color and value one of all three component produces white color. The same value of all three components, other than zero and one produces grey color. And different weights of them produce color in the inside cube.

– The CIE RGB color space is one of many RGB color spaces. RGB color model can produce colors covered by a triangle on the CIE diagram. The RGB chromaticity coordinates for CIE RGB color models are R(0.735, 0.265), G(0.274, 0.717) and B(0.167, 0.009).

Tech Knowledge Publications

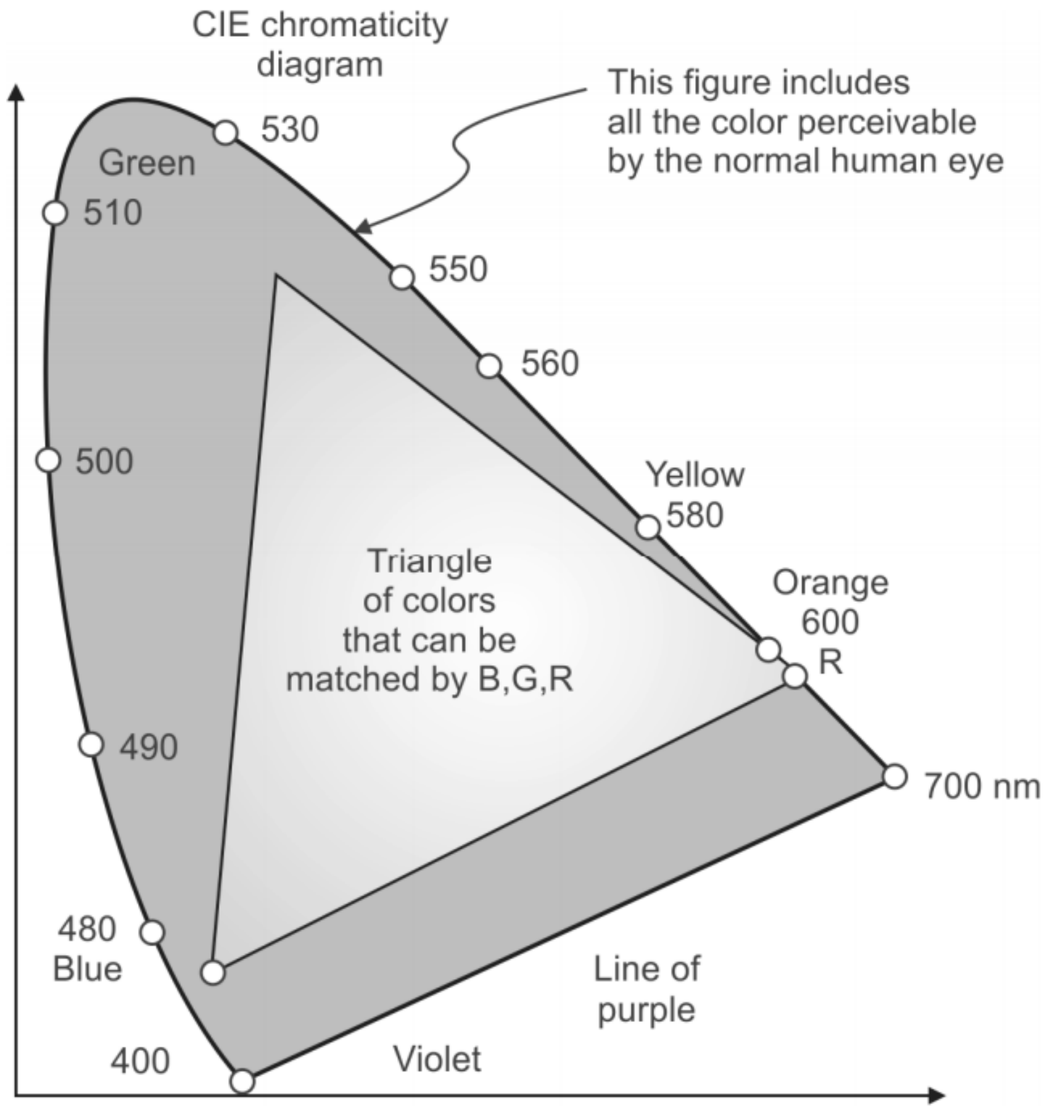


Fig. 7.3.2 : CIE RGB color model

– As CMY colors are complementary of RGB colors, it's very simple to transform color of one color space to color of other space. The transformation from RGB to CMY color space can be achieved as follow.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

7.3.2 HSV Color Model

Q. Explain the HLS color model.

SPPU - May 16, 2 Marks

Q. Explain HSV color models.

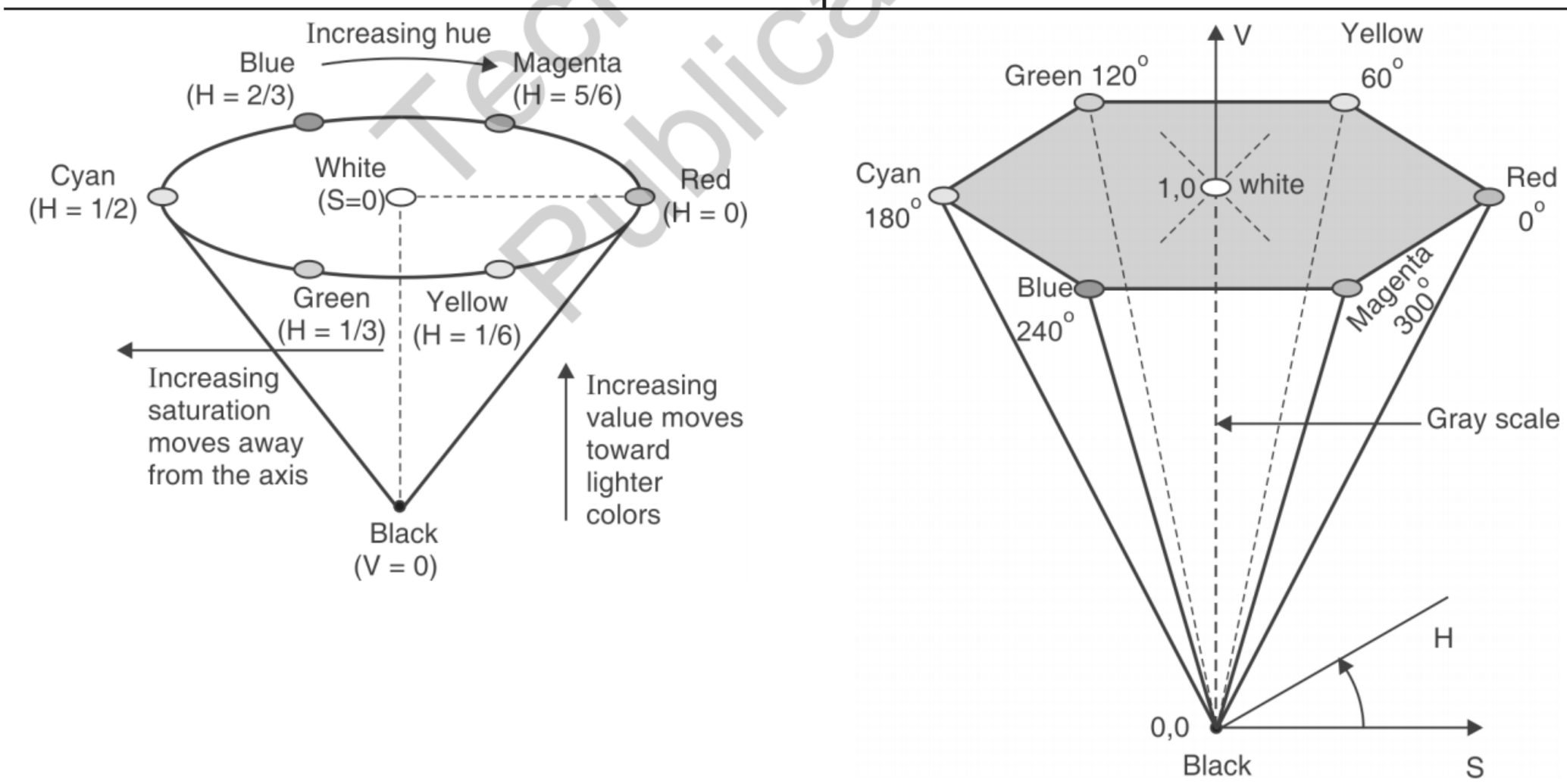
SPPU - May 17, May 18, May 19, 3 Marks

– H, S and V stand for Hue, Saturation and Value, respectively. Hue defines the color, saturation defines the purity of color and Value defines the intensity.

– HSV color model is more popular than the RGB color model in graphics and image processing due to its capability of spatting intensity value from its hue and saturation.

– RGB and CMY used primary colors to create new colors, whereas HSV is close to how human perceives the color.

– RGB and CMY color models are defined over cubic space. HSV model is defined over hexagon or cone shape.



(a) Cone shape representation **(b) Hexagon representation**

Fig. 7.3.3 : Representation of the HSV color model

– Hue range for different colors is expressed in Table 7.3.1.

Table 7.3.1 : Hue range for different colors

Color	Hue Range (in degree)
Red	0 to 60 degree
Yellow	61 to 120 degree
Green	121 to 180 degree
Cyan	181 to 240 degree
Blue	241 to 300 degree
Magenta	301 to 360 degree

– Saturation describes the amount of grey in a particular color, from zero to hundred percent. Saturation range is normalized between 0 and 1. Where 0 is pure grey and 1 is pure color. Reducing the saturation means adding more grey in the given color, which makes the color look like a faded color.

– Value or the brightness works in conjunction with saturation and it describes the brightness of the color in a range of 0 to 100 percent. V = 0 means black and V = 100 means white.

– Pure hue is selected by setting S = V = 1.

7.3.3 CMY Color Model

Q. Write short note on CMY color model.

SPPU – Dec. 19, 2 Marks

– CMY color model is a subtractive color model. The CMY colors are called the **subtractive primaries** because the color specified is obtained from by what is removed from white light.

– Cyan, Magenta and Yellow are the primary colors for this model. The surface which appears cyan absorbs the red light, therefore cyan is white minus red or blue plus green. Similarly Magenta is White minus Green or Red plus Blue and Yellow is White minus Blue or Red plus Green.

– CMY color model is used in the printing industry. Hardcopy devices like plotter employ CMY color space for printing.

– The subspace of Cartesian coordinate system for the CMY model is the same as RGB except instead of black, white color is at the origin. Black color in the RGB color model corresponds to vector (0, 0, 0) and the same color in CMY color space corresponds to vector (1, 1, 1). By subtracting the CMY component from white, we can achieve the desired color.

– However, black color produced by the CMY color model is not pure black, its mud black. So for better printing quality, CMYK color model was proposed, where K represents a black color.

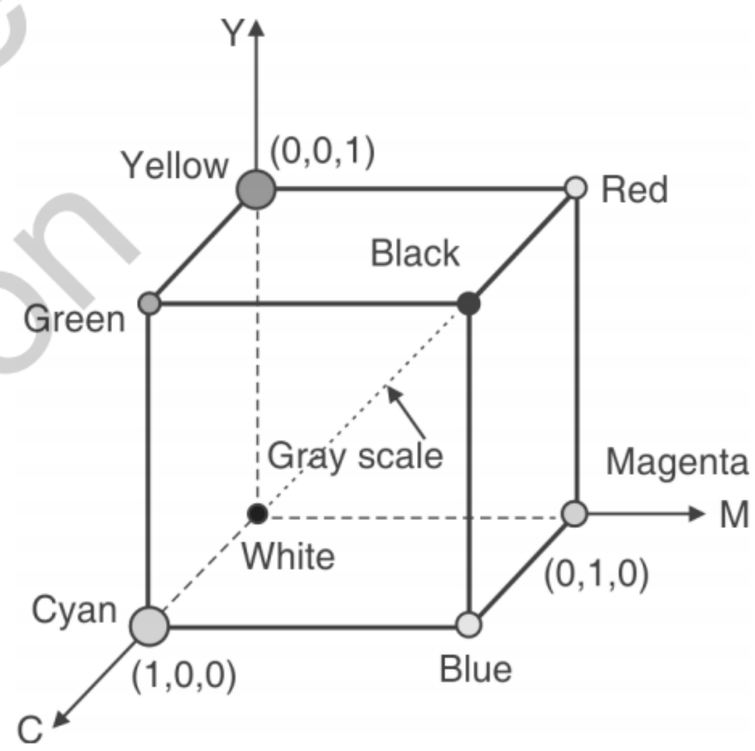


Fig. 7.3.4 : CMY color model

– As CMY colors are complementary of RGB colors, it's very simple to transform color of one color space to color of other space. Transformation from RGB to CMY color space can be achieved as follow.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

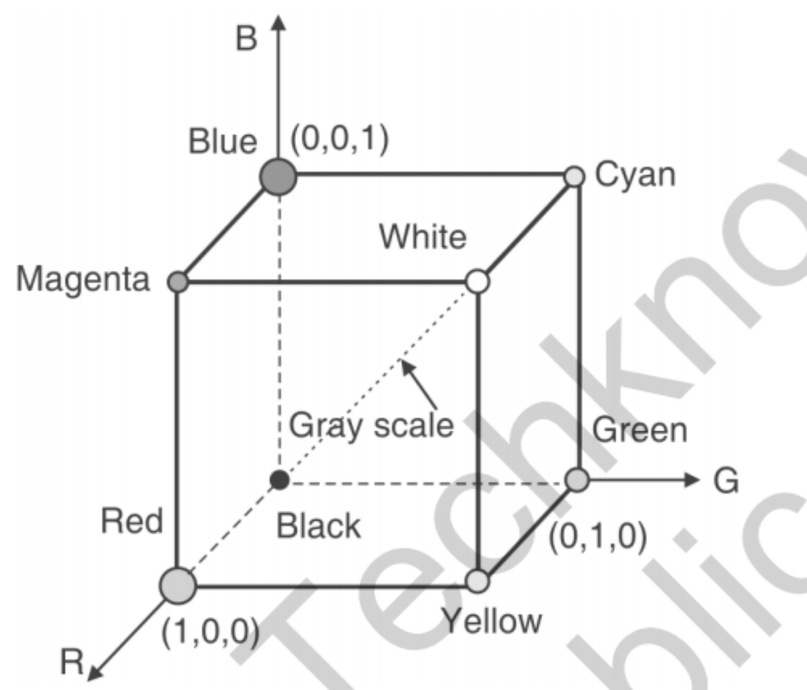
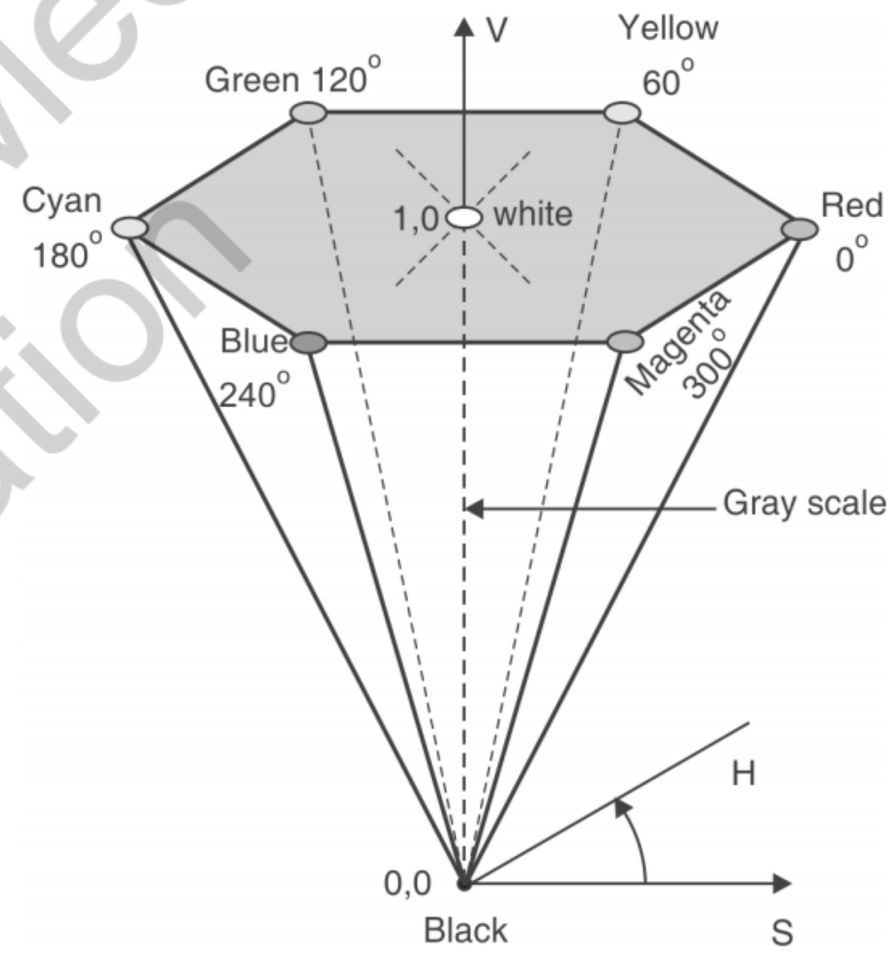
– Same way, the transformation from CMY to RGB color space can be achieved as follow.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

TechKnowledge Publications

7.3.4 Difference between RGB and HSV Color Model

Q. Compare RGB and HSV color model. **SPPU - May 15, 3 Marks**

Sr. No.	RGB Color Model	HSV Color Model
1.	RGB is cubical color model.	HSV is a cylindrical color model.
2.	RGB is an additive color model, where color is described by the addition of all components.	HSV defines the color using parameters called Hue, Saturation and Value.
3.	In color cube, origin represents black and diagonal corner represents white.	V axis represents the intensity of color.
4.	Application: Display device.	Application: Image processor.
5.		

7.3.5 HSL (Hue, Saturation, Lightness) Color Model

- HSL (Hue, Saturation and Lightness) is the extension of HSV color model.
- HSL color model used 50% brightness value around circle boundary in the HSV model.
- Visually, the HSL color model is achieved by putting inverted con on HSV color model and stretching the lightness across the apex of the shape.
- In HSL color space, near-neutral color are considered as fully saturated, which conflicts with the definition of pure color in the HSV model
- Compared to the HSV model, fully saturated colors in the HSL model are achieved by setting L = 0.5 and S = 1.
- HSV model is often known as *hexcone model* and HSL model is known as a *bi-hexcone model*.

TechKnowledge Publications

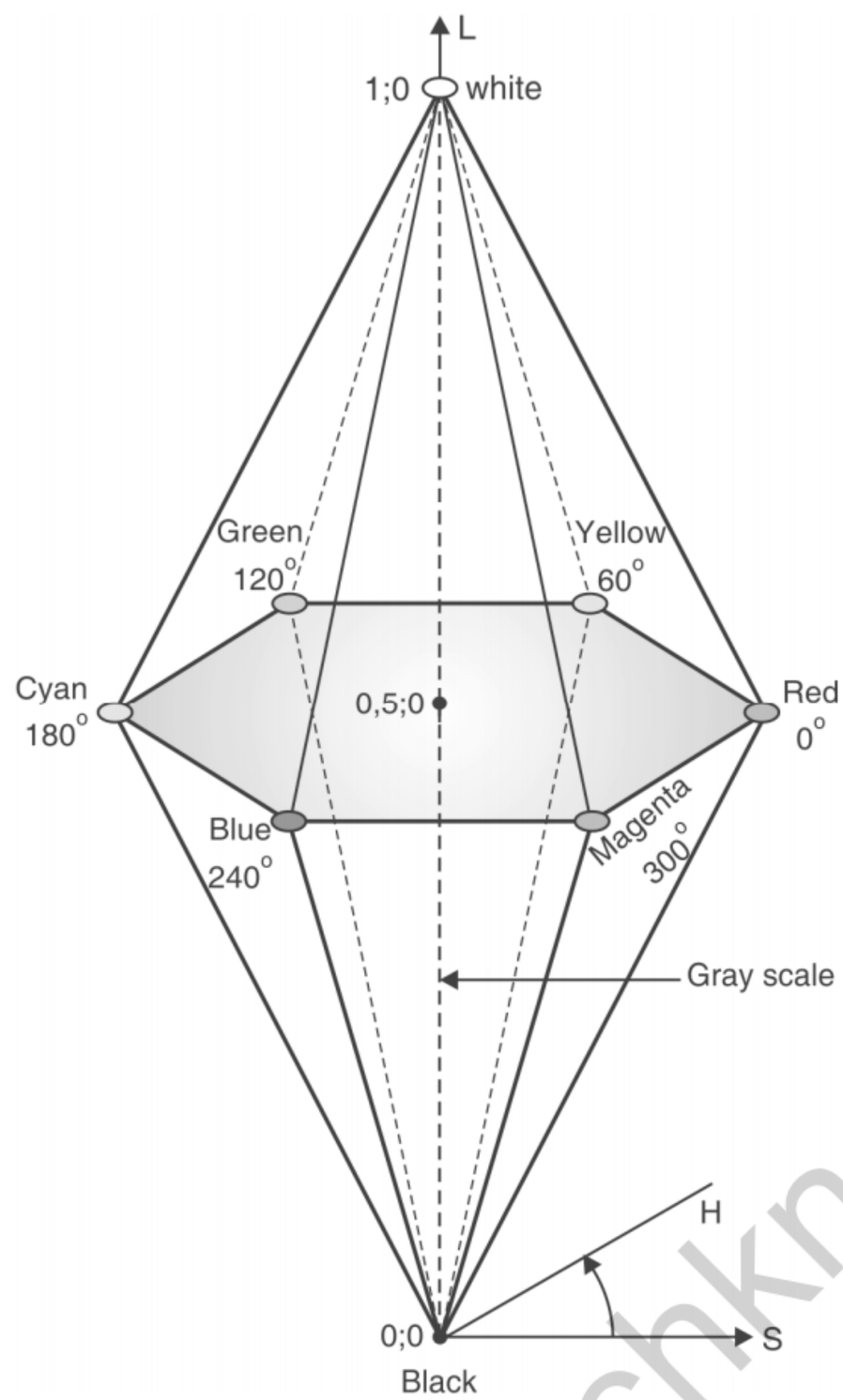


Fig. 7.3.5 : HSL color model

7.4 Applications of Color

One cannot imagine the world without colors. The wise use of colors drastically affects the representation of the concept. One can think the use of color in every area.

– Some of the prominent areas are discussed here :

1. **Art :** One obvious use of color is its application in painting.
2. **Mood :** Colors can also reflect the mood. White color represents the peace while red symbolizes the horror. One can use the colors in a different domain as per his taste and requirements.
3. **Industries :** Colors and graphics are widely used in education, entertainment and advertising industries to represents the concepts more effectively.
4. **Data analysis :** In data analysis, colors are used to represent a different level of information. In satellite image processing, green color represents the area of jungles and vegetation, the blue color represents the area of water etc. Color adds one more dimension of understanding in visualization.
5. **Warning and Highlighting :** Colors are also used to warn or attract users. Green, Red and Orange colors are used in traffic signals to give specific instructions. Similarly, certain colors are used to highlight the products.
6. **Culture and Personality :** Color may also differentiate the culture, personality, gender, age etc. Girls prefer pink over other colors, whereas blue is the choice of man. Different religions also have different representative colors.

Review Questions

Q. 1 Explain Properties of Light.

Q. 2 Define the term : Color gamut, chromaticity.

Q. 3 Explain the CIE chromaticity diagram.

Q. 4 What do you mean by color model?

Q. 5 Explain the RGB color model.

Q. 6 Explain the HLS color model.

Q. 7 Write a short note on : CMY Color Model.

Q. 8 Differentiate RGB and HSV Color Model.

Q. 9 Enlist the applications of Color.

Technical Publication

❑❑❑

CHAPTER 8

UNIT IV

Illumination Models

Syllabus

Illumination Models : Ambient Light, Diffuse reflection, Specular Reflection, and the Phong model, Combined diffuse and Specular reflections with multiple light sources, warn model.

Shading Algorithms : Halftone, Gouraud and Phong Shading.

8.1 Illumination Models

- **Rendering** is the process of generating a realistic scene. Type of light source and direction of the light source plays an important role in rendering. Position of light source decides which part of the scene should appear brighter and which part should be darker.
- Apart from light, there are many other factors like shadow, texture, reflection, refraction, type of projection play an important role in the generation of realistic view. Various **lighting models** are used to calculate intensity at every pixel in the scene. Lighting models are also known as *illumination model* or *shading models*.
- Intensity perceived to the user depends on factors such as type, numbers, distance and orientation of light sources, properties of surface. If the object reflects light, it would be visible. If the object absorbs all the lights, it will not be visible.

8.1.1 Light Sources

Q. Explain light sources.

SPPU - May 19, Dec. 19, 2 Marks

- An opaque object is visible due to reflected light from its surface. This reflected light is the sum of the contribution of light from various light sources as well as from other reflecting surfaces. Even though the object is not directly exposed to the light source, it may still be visible if nearby objects are illuminated.
- Such reflecting surfaces also work as a light source. Thus light sources may be light emitting sources like sun, bulb or it may be reflecting surfaces like wall, table, ceiling etc. We will discuss a few light sources here.
- Fig. 8.1.1 shows how the user perceives the brightness of surface as integration of lights incident from direct light source and the other reflected surface.

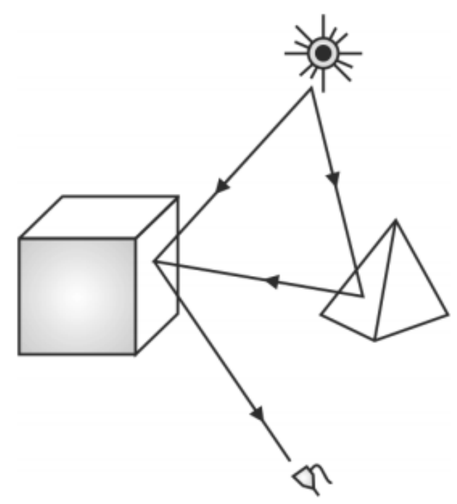


Fig. 8.1.1 : Light source and the reflecting surface

8.1.1.1 Point Light Source

- Point light source is the simplest model of the light emitter. Such light sources are Omni-directional. It emits an equal amount of light in all the directions. The intensity in the scene at any point is a function of distance from the light source. They emit light radially from the center.

Technoknowledge Publication

– Such light sources are generally small in size, like a bulb. With infinite distance from the scene, the sun is also considered a very small point light source.

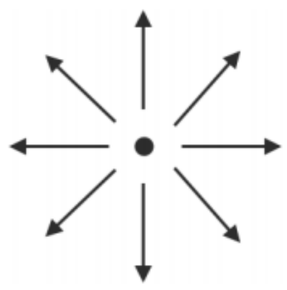


Fig. 8.1.2 : Point light source

8.1.1.2 Distributed Light Source

– Area of point light source is negligible. Its extent is very small. If the area of the light source is large compare to point light source, then it is termed as a **distributed light source**, e.g. tube light.

– Realistic rendering of the scene cannot be achieved with a point light source. It can be modelled more accurately with a distributed light source. Distributor light source span a longer area.

– Fig. 8.1.3 demonstrates the distributed light source.

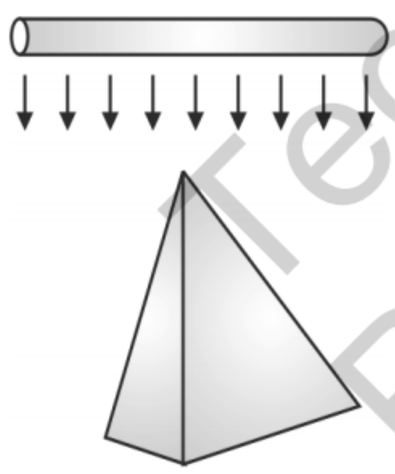


Fig. 8.1.3 : Distributed light source

8.2 Ambient Light

Q. Explain ambient light.

SPPU - May 19, Dec. 19, 2 Marks

– This is perhaps the simplest illumination model. We can model this illumination by assuming that there is no external light source is present. The object is displayed by its intrinsic intensity.

– Even if the object is not directly exposed to the light source, due to reflected rays from the other surfaces, it would be still visible. To form a basic illumination model, we can set a general level of brightness for the scene, this model is known as an **ambient light** for background light.

– Ambient light is constant for all the surfaces and it scatters equally in all direction.

– Objects illuminated using ambient illumination model appears as a monochromatic, silhouette, unless the polygon facets of objects are given different shades.

– Amount of ambient light in the scene is fixed by setting intensity parameter I_a. Though reflected light is constant for all the surfaces, the intensity of reflected light depends on the optical properties of the surface.

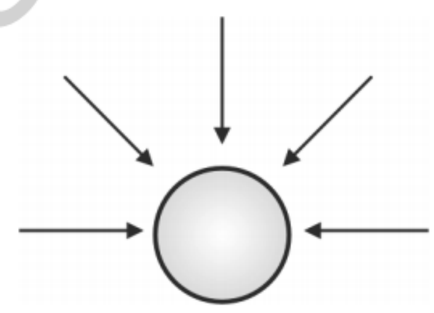


Fig. 8.2.1 : Ambient lighting

– If we assume that the ambient light reflected from all surfaces are the same and in all directions, then the illumination model equation would be,

$$I = I_a \cdot K_a$$

Where, K_a is the *ambient reflection coefficient*, which ranges from 0 to 1. And I_a is the intensity of ambient light.

– Depending upon glossy, transparent, opaque or mettle, the reflected amount of light would be different. Surfaces illuminated by ambient light model looks flat.

8.3 Diffuse Reflection

Q. Explain the concept of diffusion reflection.

SPPU - May 15, May 16, 3 Marks

Q. Describe diffuse illumination.

SPPU - Dec. 15, May 16, Dec. 18, 4 Marks

Q. Explain diffuse reflection.

SPPU - May 19, Dec. 19, 2 Marks

TechKnowledge Publications

- Reflection is the amount of light reflected back to the user from the surface. Reflection of light largely depends on the surface material. The smooth surface reflects more light compare to the rough surface. The shininess of the surface depends on the amount and direction of reflected light.
- On a rough surface like chalk stick, wall, ceiling light reflects equally in all directions as shown in Fig. 8.3.1. Such a surface has equal shininess all over. This is known as **diffuse reflection.**
- When the surface is smooth and shiny, it reflects light in a particular direction only and creates while spot on the surface. This is known as **specular reflection.**

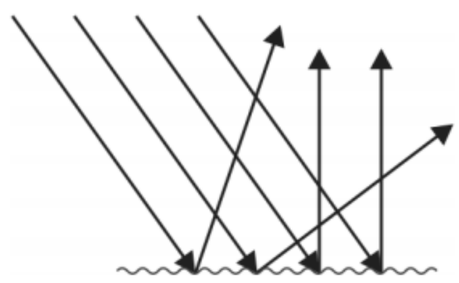


Fig. 8.3.1 : Reflection from rough surface

For example :

(i) When an apple is presented against a light source, some specific region appears bright, that is due to specular reflection.

(ii) Forehead, cheek and nose tip appears bright in good light condition, which is also example of specular reflection.

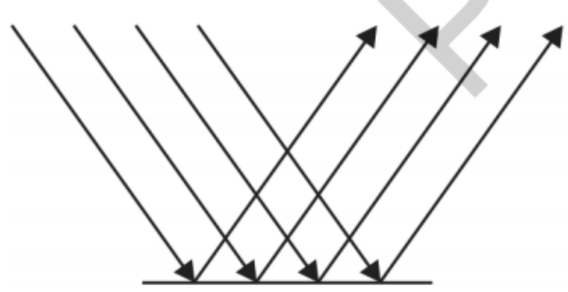


Fig. 8.3.2 : Reflection from the shiny surface

- A light beam can be imagined as a bundle of individual rays packed parallel to each other.
- When this bundle of light rays falls on the smooth surface they all reflect in the same direction which causes a **specular reflection**, and the surface appears bright. And if a bundle of light falls on a rough surface, individual ray reflects in a different direction and cause **diffuse reflection.**
- If no other light source is used and the surface is exposed only to ambient light, the intensity at any point on the surface is computed as,

$$I_{amb,\ diff} = k_d \cdot I_a$$

Where k_d is the diffuse-reflection coefficient and is constant for the entire surface. It takes a value between 0 and 1. For highly reflective surface set k_d to 1, and for the surface which absorbs most of the incident light, set k_d near to 0.

- Ambient light alone produces very flat, boring an unrealistic picture, so often at least one-point light source is added into the scene.
- If diffuse reflection scattered from the surface is equal in all direction, independent of viewing directions, such surfaces are called **ideal diffuse reflector** or **Lambertian reflectors**, since the radiated light energy from any surface is governed by *Lambert's cosine law*.

8.4 Point Source Illumination

Q. Describe point source illumination.

SPPU - Dec. 15, May 16, Dec. 18, 4 Marks

- Though Lambartian surface reflects the same amount of light in all directions, the brightness of reflected light depends on the viewing direction.
- Surface oriented perpendicular to viewing direction looks brighter than the surfaces oriented at some angle.
- As can be seen from Fig. 8.4.1, as the angle between the surface normal and the incident light increases, less light falls on the surface.

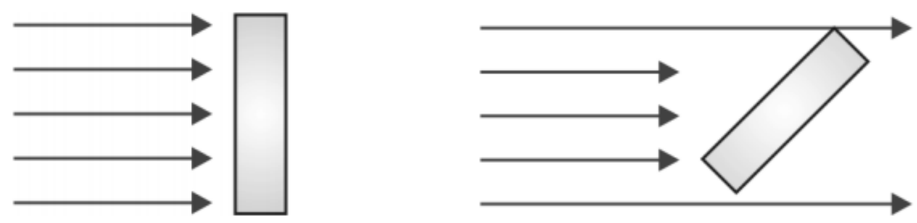


Fig. 8.4.1 : Surface perpendicular to incident light (left), oblique angle (right)

- Let θ be the angle between the incident light and the reflected light, called the **angle of incidence**.
- The area of the perpendicular surface to incident light is cos θ. Thus, the brightness of point depends on the orientation of the surface to the incident light.

Techknowledge Publications

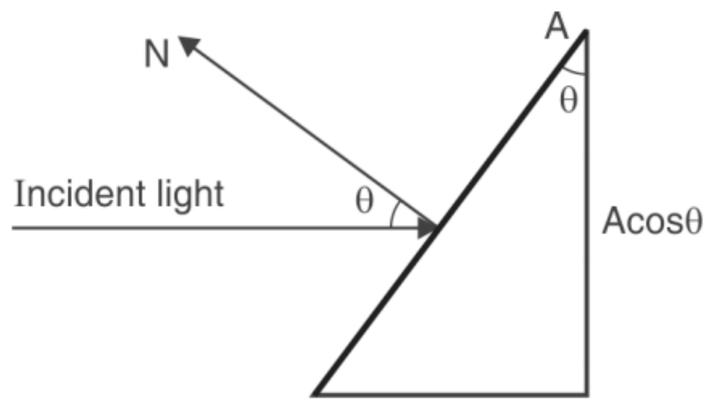


Fig. 8.4.2 : Relation between the incident and reflecting light

- If the surface is perpendicular to incident light than the angle between the incident and reflected light would be 0 and the point will be fully illuminated.
- As this angle increases, the brightness drops off.
- Let I_p denotes the intensity of the point light source. The diffuse reflection at the point is modelled as,

$$I_{diff} = k_d \cdot I_p \cdot \cos\theta$$

- Point is only illuminated if the angle is between 0° to 90 °. When cos θ is negative, the light source is behind the surface.
- Let L and N denotes the unit normal vectors in the direction of point light source and the reflected light respectively, then the cos θ = N.L. So,

$$I_{diff} = k_d \cdot I_p \cdot (N \cdot L)$$

- Let us now combine the ambient and point light source intensity to compute the total diffuse reflection.
- Many graphics packages introduce an *ambient-reflection coefficient* k_a to modify ambient light intensity I_a for each surface. Equation of total diffuse reflection using parameter k_a is written as,

$$I_{diff} = k_d \cdot k_a + (N - L)$$

- Both k_a and k_d takes a value between the range 0 and 1 and depends on the material property.

8.5 Specular Reflection and Phong Model

Q. Write short note on - Phong specular reflection model. **SPPU - May 17, 3 Marks**

Q. Explain the Phong specular reflection model in detail. **SPPU - Dec. 18, 4 Marks**

Q. Explain specular reflection. **SPPU - May 19, Dec. 19, 2 Marks**

- When light falls on a shiny surface, we see a bright spot at a certain viewing direction. This is called **specular reflection**.
- Specular reflection is the result of near-total or total reflection of incident light on the concentrated region.
- The angle of specular reflection is the same as the angle of the incident light.

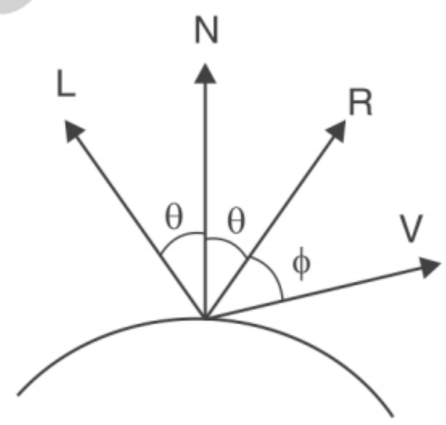


Fig. 8.5.1 : Reflection angle is the same as the incident angle

- In Fig. 8.5.1, L is the direction of light source and R is the direction of specular reflection. V is the viewing direction. θ is the reflection angle with respect to viewing direction V.
- For perfect reflectors like a mirror, specular reflection is achieved when L and R coincide i.e. when θ = 0.
- Phong model sets the intensity of specular reflection proportional to $\cos^n\phi$.
- Angle ϕ varies from 0° to 90°, so specular reflection varies from 0 to 1.
- Value of the specular reflection parameter *n* is determined by the type of material.
- The shiny surface is modelled with a very large value of n (100 or more) and for a dull surface, n is set to very small (near 1).

Tech Knowledge Publications

- For perfect reflector n is infinite.
- For each surface, we can model the monochromatic specular intensity variation using specular reflection coefficient W(θ) as follow.

$$I_{spec} = W(\theta) \cdot I_p \cdot \cos^n \phi$$

Where I_p is the intensity of the point light source.

- Since V and R are unit vectors in the viewing and specular reflection direction, we can compute the cos ϕ as,

$$\text{Cos } \phi = V \cdot R$$

- With the assumption that the specular reflection coefficient is constant (K_s), the intensity of the specular reflection at any point on the surface is calculated as,

$$I_{spec} = K_s \cdot I_p \cdot (V \cdot R)^n$$

8.6 Combined Diffuse and Specular Reflections with Multiple Light Sources

- For a single point light source, we can model the diffuse and specular reflection together as,

$$I = I_{diff} + I_{spec}$$

$$I = k_a \cdot I_a + k_a \cdot I_p (N \cdot L) + k_s \cdot I_p (V \cdot R)^n$$

- If multiple light sources (k) are present in the scene, the light reflection at any point in the scene is obtained by summing the contribution of each light sources.

$$I = k_a \cdot I_a + \sum_{i=1}^{k} k_d I_p (N \cdot L) + k_s \cdot I_p (V \cdot R)^n$$

- On summing up intensity computed from k light sources, the total intensity may exceed the maximum allowable intensity value. To handle this problem, some kind of intensity normalization is applied.
- If calculated intensity exceeds the maximum value, simply set it to the maximum allowed intensity value.
- Another way to compensate the intensity overflow is to normalize the intensity component of each light source by the largest intensity value.
- Another complicated approach is to compute the intensity at each point and scale the value to allowable intensity range.

8.7 Warn Model

Q. Write short note on the Warn model.

SPPU - May 17, 3 Marks

- Discussed all models consider only the point light source. **Warn model** provides a way to model the studio light with varying intensity in different directions.
- Large non point light sources near to objects can be modelled by use of several point light sources arranged in grid.
- Light sources are modelled using the *Phong model*, and the intensity of the light source is controlled by the parameter called *Phong exponent*.
- In addition to that, light controls like "barn doors" and "spotlighting" are simulated using warn model.
- *Flaps* control the amount of light emitted from the light source. For each direction x, y and z, two flaps are used.
- Spotlights are used to control the amount of light emitted within a cone with apex at the position of point light source.
- Multiple light sources can be used to achieve an additive effect. The intensity of emitted light is controlled by flaps.
- Warn model is implemented within PHIGS+ standard.

8.8 Ray Tracing

Q. Explain Ray Tracing.

SPPU - May 15, May 16, 3 Marks

- Ray tracing is a technique for rendering three-dimensional graphics with very complex light interactions.

Tech Knowledge Publications

- Ray tracing is a global illumination model and it accounts the contribution of light coming from non-local light sources and reflection from other objects in the scene. Ray tracing is very useful in realistic scene rendering. It is useful in rendering, shadow, shading, visible surface determination etc. Working principle of ray tracing largely depends on principles of geometric optics.
- Ray tracing is highly used in games, entertainment, simulation and film industries. It is an image space approach.
- Formally, ray tracing is a technique of generating an image by tracing the path of light through a pixel in an image plane and its reflection from the object surface. Process of ray tracing is illustrated in Fig. 8.8.1.

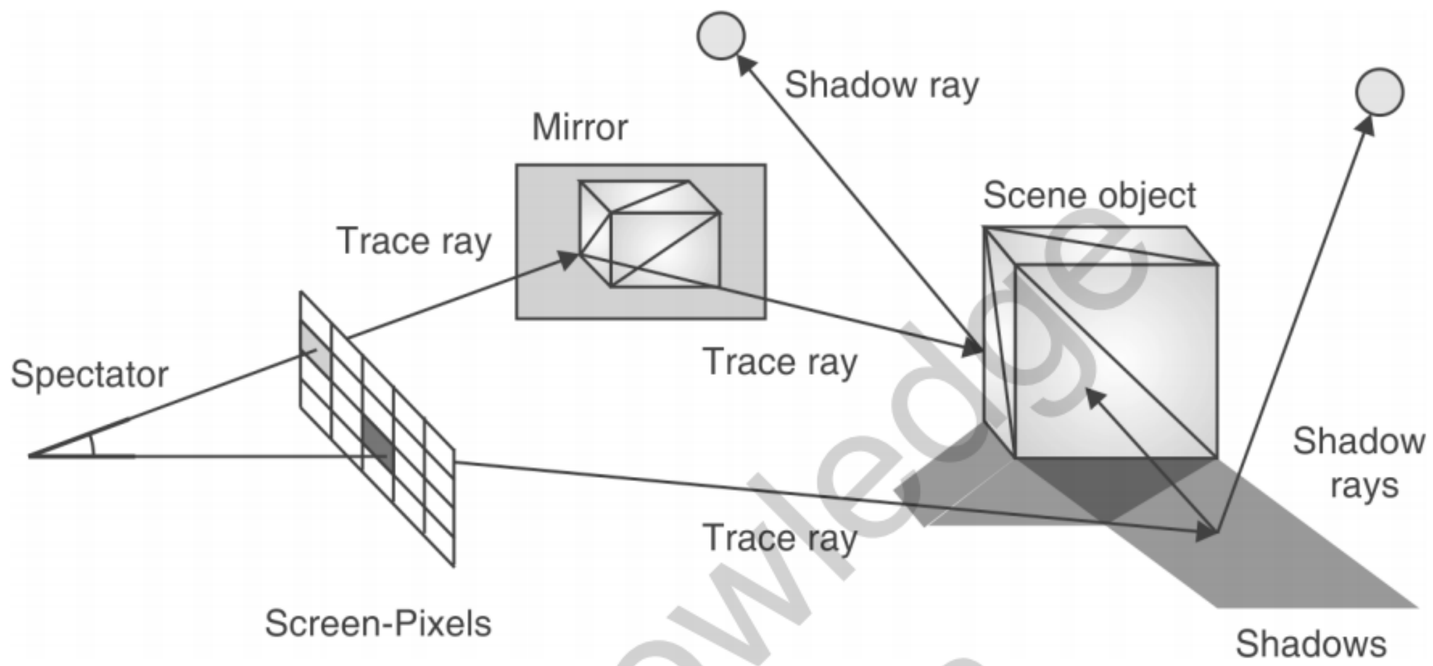


Fig. 8.8.1 : Ray tracing

- The objective is to determine the color of each light ray that strikes the view window before reaching the eye.
- If we trace rays from the light source, very few rays reach to the eye, most of the rays keep bounding in a random direction. This approach will waste our time. To save our effort, we can follow a *backward* approach, in which instead of tracing a ray from the light source, we start tracing it form eye itself.
- If the original ray came directly from the light source, then the backwards ray will go directly to the light source; if the original bounced off a table first, the backwards ray will also bounce off the table. So the backwards method does the same thing as the original forward method, except it doesn't waste any effort on rays that never reach the eye.

8.8.1 Methods of Ray Tracing

Two popular methods of ray tracing are discussed here.

8.8.1.1 Basic Ray Tracing Algorithm

- The entire scene is divided into a grid. Each grid cell corresponds to one pixel on the screen. Light rays from the surface proceed in all directions. There are infinite possible paths.
- Contribution to a particular pixel is determined by tracing back the light path from pixel to the scene. We will simulate this approach for one ray.
- For each ray, we will determine the intersection of the ray with surfaces in the scene. The nearest intersecting surface is the visible surface. Ray is then reflected in other direction with an incident angle (angle of reflection is same as the angle of the incident).

- If the surface is not opaque, ray penetrates the surface and travels in refraction direction. Reflected and refracted rays are called secondary rays. Recursively this process continues. This is well represented with the help of the concept of *ray tracing tree*. Fig. 8.8.2 shows the ray tracing tree for a given environment. Termination criteria can be set by the user by setting the depth of the tree. Typically, the left branch represents reflection path and right branch represent transmission path.

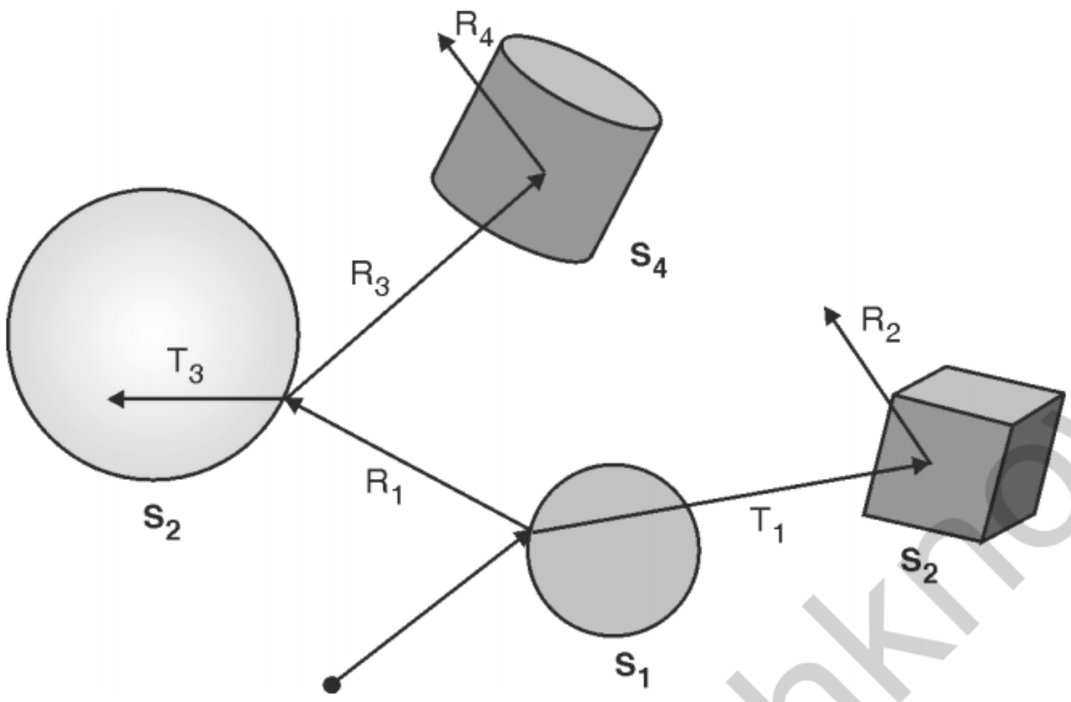


Fig. 8.8.2 : Reflection and refraction path of a ray

- The intensity of a pixel is calculated by accumulating intensity from the leaf to bottom in ray tracing tree. Attenuation of intensity from one node to the next node is proportional to the distance between the nodes. If no intersection is found, the ray-tracing tree would be null and the pixel is assigned background color.

8.8.1.2 Radiosity Lighting Model

- Radiosity is a global illumination model which overcomes the issues associated with local illumination model. It accounts light coming from the light source and light reflected from other surfaces. This approach is independent of viewpoint.
- Local illumination model considers the effect of light rays coming from nearby surfaces only once. Global illumination iteratively adds the effect of reflected rays from different surfaces. This approach keeps tracing the rays bouncing *off* and adds its cumulative effect to the scene.
- Conventional rendering methods decide the intensity at any point by considering that light is first falling on object and then it is reflected to the viewer. Radiosity model considers that light is being reflected from a chain of objects and then coming to the viewer.
- In offices, rooms, halls, and almost everywhere, interior parts are not directly exposed to the light source. In case of local illumination model, such parts of the scene appear darker. But such interior parts may be exposed to other parts in a scene which is illuminated by the direct light source.
- Radiosity model considers the effect of reflected light coming from such surfaces to interior parts of the scene and make them more realistic.
- In the radiosity method, the entire scene is divided into small patches. Consider each patch as viewpoint and center of the patch is aligned with camera or eye. The intensity of the patch is determined by the lights visible from the camera or eye. This process is repeated for each patch in the scene. And this is repeated again and again until two successive iterations have difference less than a certain threshold.
- Consider the room with a pillar and a window. Assume pillar is in the center of the room and grows from floor to ceiling and window is in the center of the wall in front of the pillar. Another assumption is that there exists only one light source and that is a small sun in the sky, visible from the window. Consider that in the absence of sun, the entire room is completely dark, there does not exist any emissive object. The entire room is divided into small patches.
- If we place our eye at the top of the pillar, the sun wants to be visible and light is not coming from any other source, so that patch would appear completely dark.

Tech Knowledge Publication

- If we slide down even further on the pillar, next patch also might not be directly visible to the sun, so that would also be completely dark. Patches which are partially or fully exposed to the sun will be illuminated by applying normal lighting models. This calculation is applied to all the patches of room. At the end of the first iteration, very few patches will be illuminated.
- In the next iteration, patches which are not directly exposed to the sun may be now exposed to the surfaces which are already illuminated in the first iteration. Light may bounce off from illuminated patches to other surfaces. Hence the intensity of many patches are updated in the second iteration and scene becomes more bright.
- This process is repeated until it satisfies the stop criteria. The number of iteration depends on the complexity of the scene and the number of light sources. The process may terminate after few iterations or it may go on up to a few thousands of iterations.
- Let's assume the rate at which surface emits the energy is E_1 and the rate at which it reflects energy back to the environment is ρE_2. Where E_2 is radiant energy incident on the given surface. If there is no leakage of flux from any object surface, then total energy is given by,

$$E_i = E_{1i} + \rho E_{2i}$$

Energy incident on any surface is the summation of all the energy coming from all other surfaces.

$$E_2 = \sum E_j \cdot f_{ij}$$

Where f_{ij} is the form factor governing the interchange of radiation between bodies i and j. This form factor depends on the temperature and shape of both bodies. So, the total energy emitted from surface i is given by,

$$E_i = E_{1i} + \rho \sum E_j \cdot f_{ij}$$

- Radiosity algorithm is used for the global illumination effect produced by objects and environment. Due to the series of iterative and recursive calculations, this process is very time and space consuming.

Advantages

- Radiosity generates very realistic view for diffuse surfaces.
- It is easy to understand and implement.
- With the use of hardware, it can be optimized for better performance.

Disadvantages

- Due to extensive computation, it is slow.
- It cannot properly handle the point light source.
- It is not suited for shiny surfaces.
- Tessellation i.e. dividing the scene in patches and processing each patch is time-consuming.

8.9 Shading Algorithms

Q. What is the surface shading algorithm?

SPPU - May 15, 2 Marks

Q. Enlist and explain shading algorithms with their disadvantages. **SPPU - May 17, 7 Marks**

Q. Enlist and explain in detail any 2 shading algorithms.

SPPU - May 18, Dec. 18, May 19, 7 Marks

- In this section, we will discuss techniques to render the polygon surfaces.
- The surface can be illuminated either using constant intensity value or intensity may be calculated at each pixel using an illumination model. Surface rendered with a single intensity appears flat and unnatural.

Tech Knowledge Publications

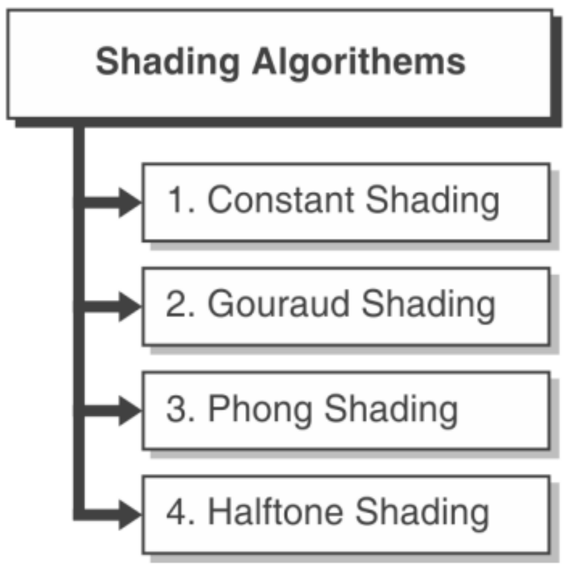


Fig. 8.9.1 : Shading algorithms

8.9.1 Constant Shading

- This is the simplest and fastest rendering technique. It is also known as **flat shading** or **faceted shading**. Illumination model is applied at single pixel only, and the entire surface is rendered with the same intensity. If two adjacent surfaces have different intensity, then this model creates a facet effect.
- This approach is valid under the following assumptions :
  - The light source is at infinite distance, so N.L is constant across the entire surface. Here, N is normal to the surface and L is the direction of light source.
  - The viewer is at infinite distance, so N.R is constant across the entire surface. R is the direction of specular reflection.
  - Polygon is not an approximation of curves surface, but it is the actual surface.
- This model is not used for realistic rendering.

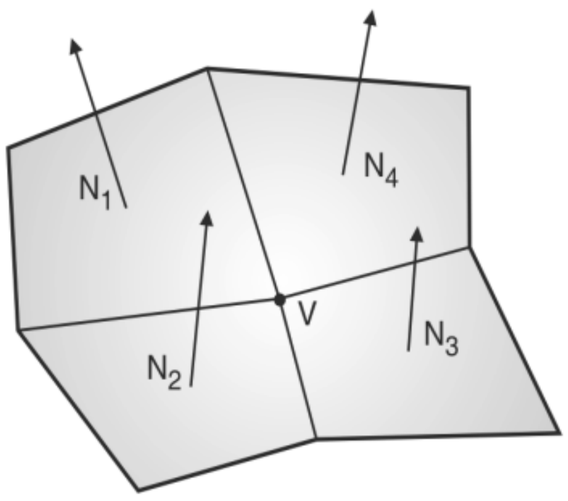


Fig. 8.9.2 : Constant shading

- Object shaded using constant shading model is shown in Fig. 8.9.7.

8.9.2 Gouraud Shading

Q. Explain Gouraud Shading algorithm.

SPPU - Dec. 17, Dec. 18, 4 Marks

- This model is also known as **intensity interpolation method** because of its characteristic working procedure.
- Polygon surface is rendered by linearly interpolating the intensity across the surface.
- The intensity value is matched with the adjacent polygon edge, so this method provides a smooth shading effect at the boundary and eliminates facet effect produced in constant shading model.
- Intensity computation of surface is done as follow :
  - The average unit normal is calculated at every vertex.
  - Illumination model is applied at every vertex to decide the intensity of vertex.
  - Perform linear interpolation between two vertices.
  - To obtain a normal vector at the shared vertex by averaging the normal vector of each surfaces sharing that vertex.
  - As shown in Fig. 8.9.3, three surfaces with normal vectors N_1, N_2 and N_3 are sharing the vertex V. So average unit normal vector at vertex V is given as,

$$N_v = \frac{N_1 + N_2 + N_3}{|N_1 + N_2 + N_3|}$$

- In general, a vertex may be shared by n surfaces, so average normal vector is computed as,

$$N_v = \frac{\sum_{i=1}^{n} N_i}{\left| \sum_{i=1}^{n} N_i \right|}$$

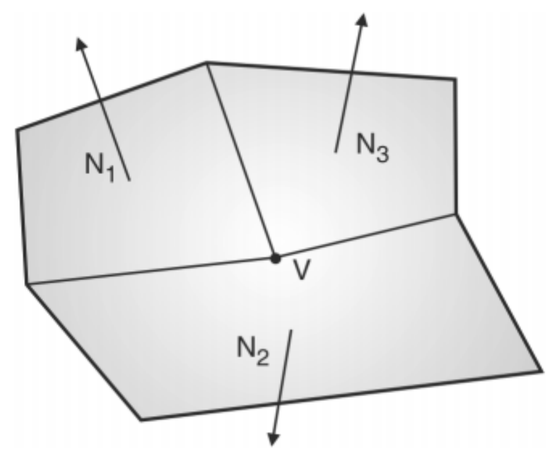


Fig. 8.9.3 : Average normal vector computation

Technknowledge Publications

- Once all vertex normals are computed for the surface, illumination model can be applied to compute intensity at the vertex.
- Once the vertex intensity is known, we can interpolate intensity on edge-sharing two vertices. And similarly, we can interpolate the intensity along the two edges for each scan line.
- As shown in Fig. 8.9.4, suppose normal vector at each vertex 1, 2 and 3 is computed and I_1, I_2 and I_3 are their respective intensity. Scan line intersects polygon edges at points a and b. Let us assume that intensity at the intersection point is I_a and I_b. We can define intensities at intersection point using linear interpolation as follow.

$$I_a = \frac{y_a - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_a}{y_1 - y_2} I_2$$

$$I_b = \frac{y_b - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_b}{y_3 - y_2} I_2$$

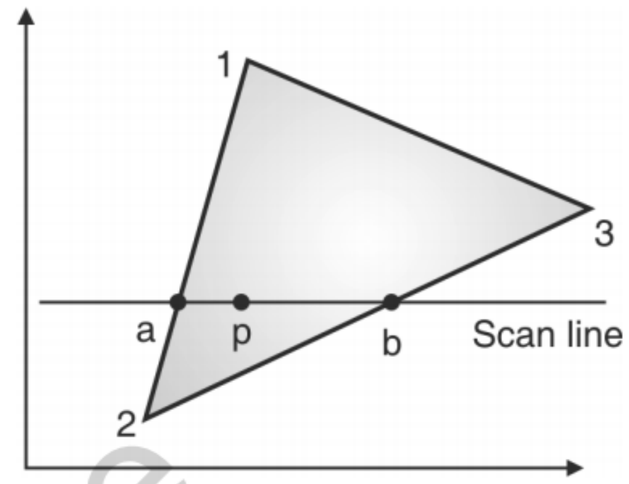


Fig. 8.9.4 : Intensity interpolation across edges and the scan line

- Once the intensity of intersection points of the scan line is known, we can compute intensity at any point p along the scan line as follow.

$$I_p = \frac{x_b - x_p}{x_b - x_a} I_a + \frac{x_p - x_a}{x_b - x_a} I_b$$

- Evaluating these formulas for each scan line and every pixel along scan line is computationally ineffective. We can take advantage of coherence property and we can derive incremental intensity calculation.
- Using coherence property, we can derive incremental formula to compute intensity between scan lines and intensity along scan line. As shown in Fig. 8.9.5(a), interpolated intensity at edge intersection I would be,

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

- Intensity along this edge for next scan line Y = y – 1 is computed as,

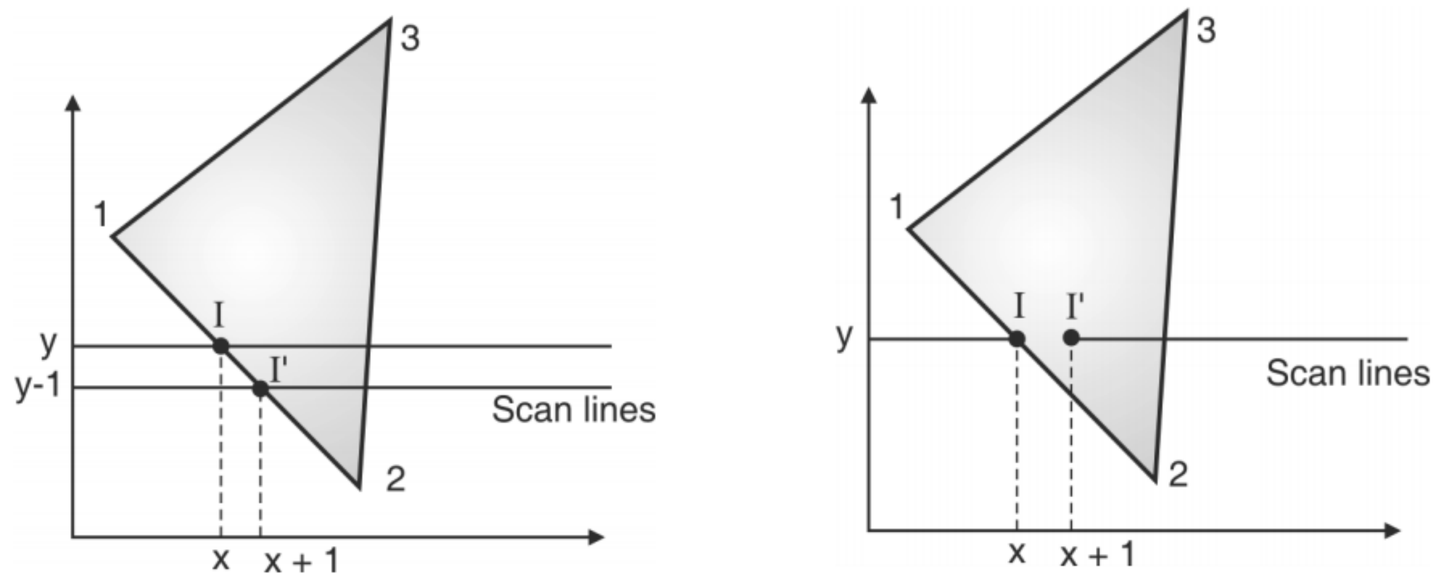
$$I' = \frac{y - 1 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y + 1}{y_1 - y_2} I_2$$

$$I' = \frac{y - y_2}{y_1 - y_2} I_1 - \frac{I_1}{y_1 - y_2} + \frac{y_1 - y}{y_1 - y_2} I_2 + \frac{I_2}{y_1 - y_2}$$

$$I' = \left(\frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2 \right) - \frac{I_1}{y_1 - y_2} + \frac{I_2}{y_1 - y_2}$$

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Techknowledge Publications



(a) Coherence between scan lines **(b) Coherence along the scan line**

Fig. 8.9.5 : Utilizing coherence property for incremental intensity calculation

– As shown in Fig. 8.9.5(b), If intensity at the location (x, y) is I, then intensity along the scan line at pixel (x + 1, y) is computed as,

$$I' = I + \frac{I_b - I_a}{x_b - x_a}$$

– Object shaded using Gouraud shading model is shown in Fig. 8.9.7.

Advantages

– Can be combined with hidden surface removal algorithm to render the visible surface of polygon along the scan line.

– Eliminates intensity discontinuity at edges.

Disadvantages

– Highlighted white spot on the surface are sometimes displayed with random shape.

– Creates match band effects.

– Surface with a sharp intensity drop cannot be rendered properly.

8.9.3 Phong Shading

Q. Write a short note on Phong Shading.

SPPU - May 15, 4 Marks

Q. Explain the Phong Shading Algorithm.

SPPU - Dec. 17, 4 Marks

Q. Write a note on Phong Reflection Model.

SPPU - Dec. 17, May 18, Dec. 18, 4 Marks

– This method is also known as the **normal vector interpolation shading**. This method is more accurate than Gouraud shading. This algorithm interpolates the normal vector rather than intensity and apply the illumination model at interpolated normal vectors at each pixel.

– Phong shading works as follow :

  - Compute averaged unit normal vector at each vertex.
  - Linearly interpolate vertex normal vector across the polygon surface.
  - Apply the illumination model at pixel along the scan line.

– Finding the average normal vector at the shared vertex is identical to Gouraud shading. As shown in Fig. 8.9.6, averaged unit normal vector at vertex 1 and 2 are N_1 and N_2 respectively. Now we will interpolate these normal along the edge.

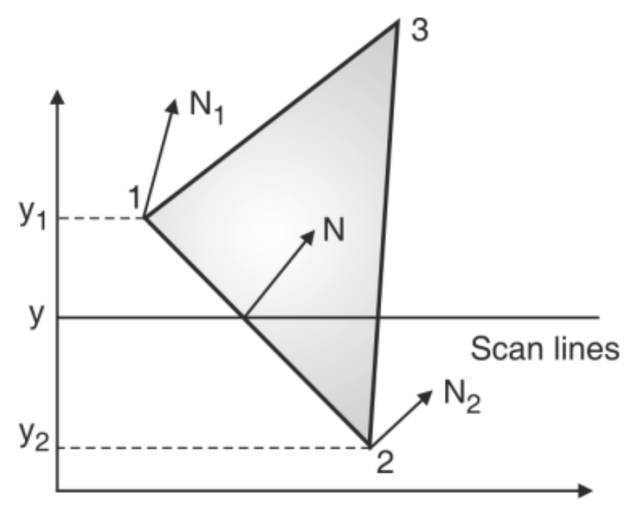


Fig. 8.9.6 : Normal vector interpolation

– Normal at the intersection of scan line y with the edge is given as,

$$N = \frac{y - y_2}{y_1 - y_2} N_1 + \frac{y_1 - y}{y_1 - y_2} N_2$$

Technknowledge Publications

– Illumination model is applied using this normal. Like Gouraud shading, we can derive the equation for incremental computation of normal vector along the scan line and between the scan lines.

Advantages

– Highlights are displayed more realistically.

– Greatly reduces the match band effect of Gouraud shading.

– Provide more accurate shading.

Disadvantages

– Need more calculation.

– Computationally expensive.

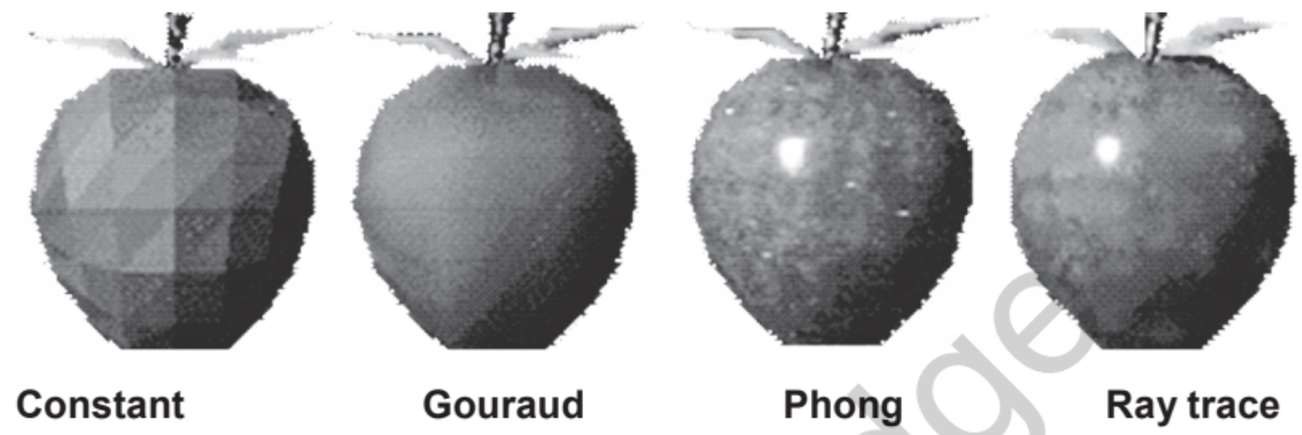


Fig. 8.9.7 : Comparison of different interpolation model

8.9.4 Halftone Shading

Q. Write short note on Halftone. **SPPU - May 17, 2 Marks**

– Halftoning is used on a device supporting a limited intensity range. Typically, antialiasing uses multiple intensity levels to reduce the aliasing effect by increasing resolution. Halftoning improves the quality of the picture by increasing visual resolution using a minimum number of intensity levels. This method is also known as **patterning**.

– Halftoning is used to reproduce the photographs for newspapers and magazines. This process is called **Halftoning** and the reproduced image is called **halftone**. It uses a 2D rectangular grid of pixels to determine intensity at a given location. Size of the rectangular grid varies according to the screen resolution. Intensity perceived by human eye is spatial integration of intensity on the visible region. Black and white monitors and old printers exploited this characteristic to produce the picture which looks like displayed with multiple intensity values.

– Halftone image looks like as if it is composed of black circles of different radius. The dark region is produced using large circles. Light regions are produced by small circles. Good quality books and magazine are printed using approximately 60-80 circles of varying radius per centimetre. The low-quality newspaper is printed using about 25-30 circles per centimetre.

– Example of halftone images are shown below :

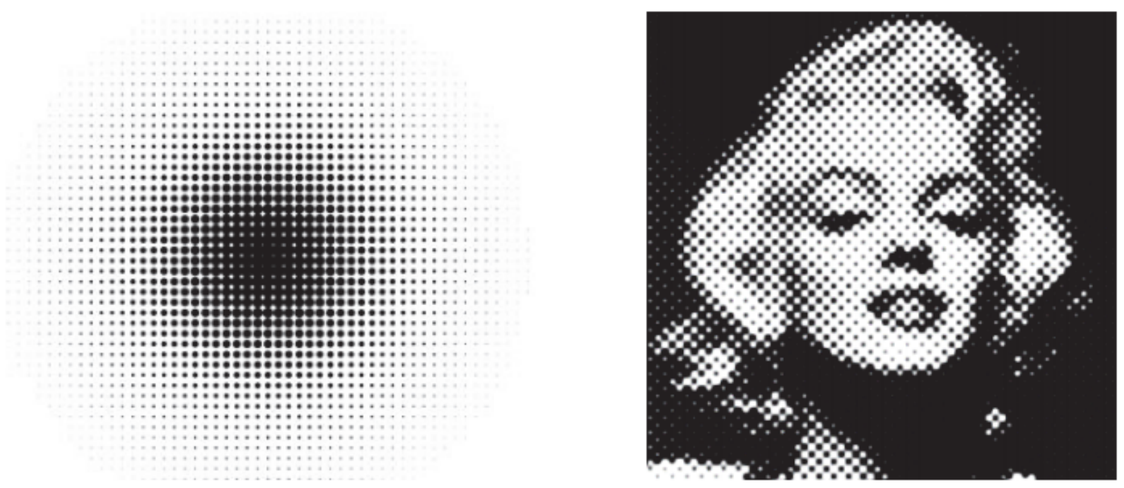


Fig. 8.9.8 : Example of halftone images

Tech Knowledge Publications

– Using n by n rectangular grid, we can display $n^2 + 1$ different intensity levels. Black and white system uses a grid of size 1 by 1 Fig. 8.9.9 shows how circles are added to the grid with increasing intensities. With 2 by 2 and 3 by 2 grid, we can display five and ten intensity levels respectively.

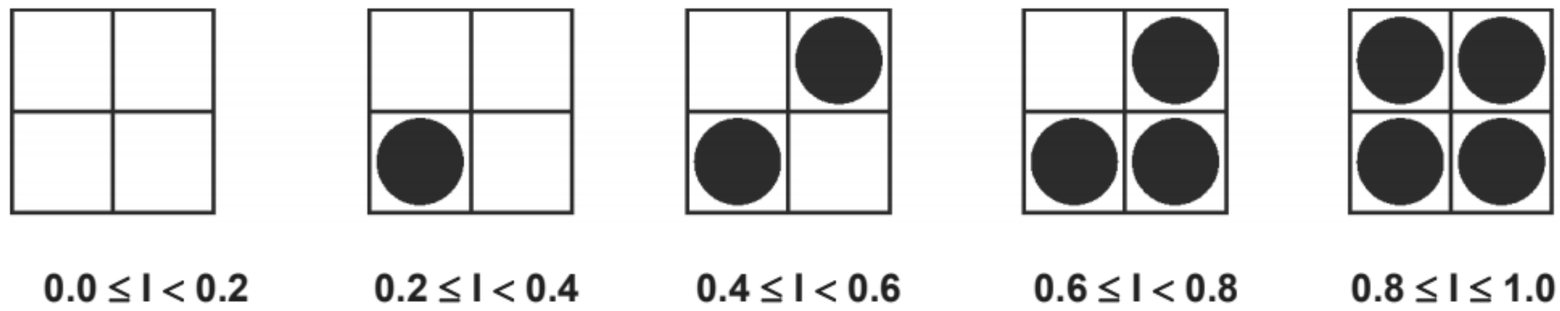


Fig. 8.9.9 : 2 × 2 - pixel grid to produce five intensity levels

Fig. 8.9.10 shows how circles are added to the grid with increasing intensities for 3 × 3 grid.

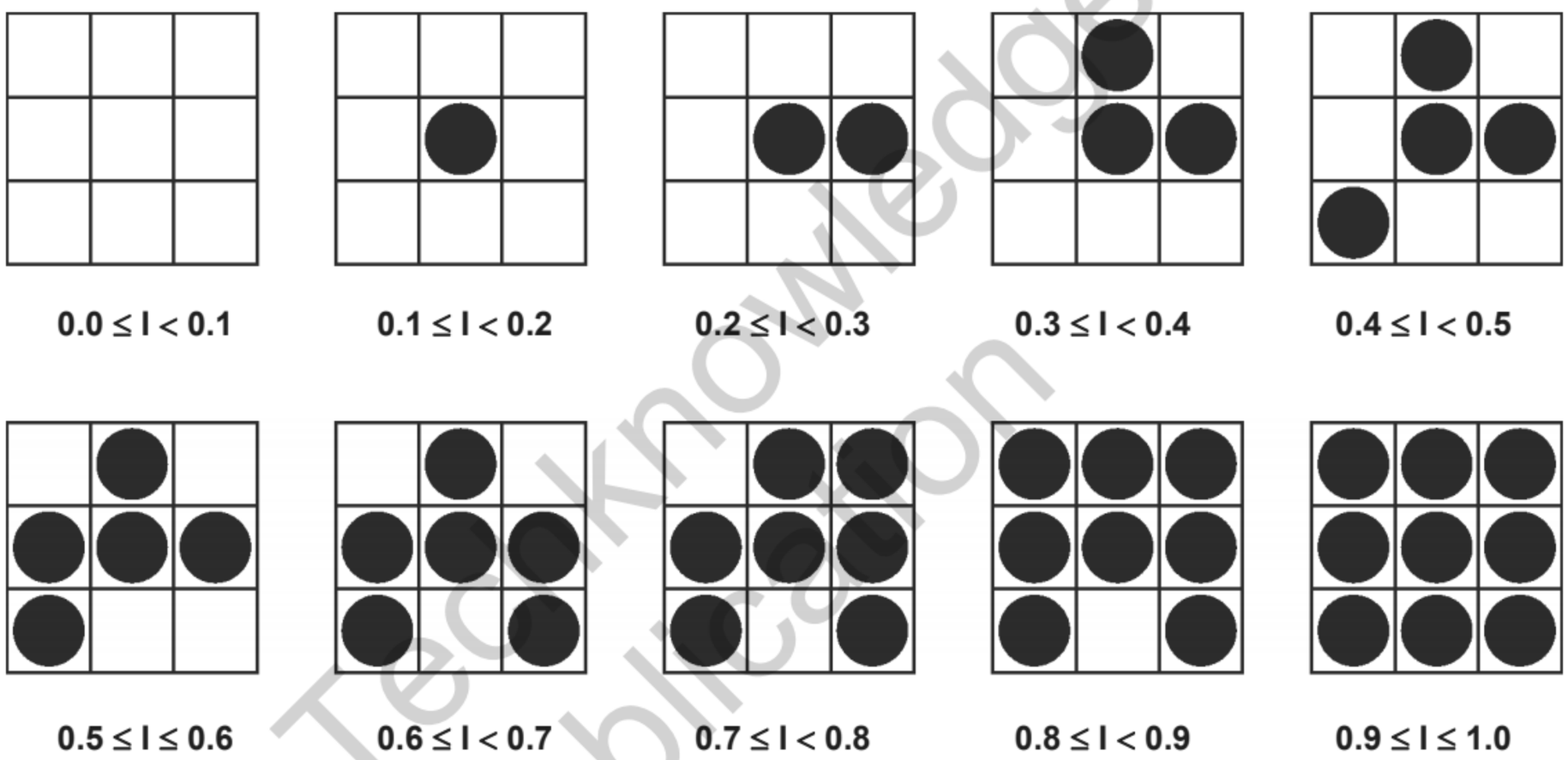


Fig. 8.9.10 : 3 × 3 - pixel grid to produce ten intensity levels

We can also use a mask to produce the pixel with the desired intensity. We shall mask the pixel with the following mask coefficients :

$$\begin{bmatrix} 8 & 3 & 7 \\ 5 & 1 & 2 \\ 4 & 9 & 6 \end{bmatrix}$$

To display intensity level i, we turn on pixels with a value less or equal to i in the above mask. n by n pixel grid increases the number of intensity levels to $n^2 + 1$, it reduces the resolution of the picture by a factor of 1/n. Screen with resolution 512 × 512 reduces to 256 × 256 intensity points with a grid of size 2 by 2. Similarly, a screen with resolution 512 × 512 reduces to 128 × 128 intensity points with a grid of size 3 by 3.

Like bilevel system, it is possible to improve the intensity range for color display systems. Display system with three-bit per pixel can display eight colors. Each electron gun is assigned one pixel to turn it on or off. With 2 × 2 pixel grid, we can have four dots in each pixel; that will leave us with possible twelve dots. Now, instead of three, we have 12 possibilities to produce color. Each RGB color has four dots, which allows five settings per pixel. This configuration provides us with total 5 × 5 × 5 i.e. 125 different shades rather than just eight.

Tech Knowledge Publications

8.9.5 Difference between Phong Shading and Gouraud Shading

Q. Explain difference between Gouraud shading and Phong shading. **SPPU - Dec. 19, 6 Mark**

Sr. No.	Phong Shading	Gouraud Shading
1.	Most sophisticated model.	Less sophisticated model
2.	It is named after Bui Tuong Phong.	It is named after Henri Gouraud.
3.	It was first published in 1973.	It was first published in 1971.
4.	Intensity is computed by interpolating normal at vertices.	Intensity is computed by interpolating color at the vertex.
5.	Requires more computation and produce smooth intensity.	Requires less computation and produce near realistic rendering.
6.	The light equation is used at each pixel.	The light equation is used at each vertex.
7.	Produces smooth and shiny surfaces.	Produces smooth surfaces.

Review Questions

Q. 1 Discuss Point and Distributed light source.

Q. 2 Write a short note on ambient light.

Q. 3 What is diffuse refection?

Q. 4 Describe point source illumination.

Q. 5 Explain Phong specular reflection model in detail.

Q. 6 Explain specular reflection.

Q. 7 Discuss combined diffuse and specular reflections with multiple light sources.

Q. 8 Write short notes on Warn model.

Q. 9 Explain Ray Tracing.

Q. 10 Discuss Basic Ray Tracing Algorithm.

Q. 11 Describe Radiosity Lighting Model.

Q. 12 State advantage and disadvantages of radiosity lighting model.

Q. 13 Write a short note on Constant shading model.

Q. 14 Explain Gouraud Shading algorithm.

Q. 15 Write a note on Phong Reflection Model.

Q. 16 Write short notes on Halftone shading.

Q. 17 Differentiate between Phong and Gouraud shading.

Technknowledge Publication

❑❑❑

CHAPTER 9

UNIT IV

Visible Surface Detection

Syllabus

Hidden Surfaces Introduction, Back face detection and removal, Algorithms: Depth buffer (z), Depth sorts (Painter), Area subdivision (Warnock)

9.1 Introduction

- In the real scene, there may be many 3D objects placed randomly, completely or partially hiding some other objects.
- At any given viewing direction, some of the surfaces may not be visible. Identifying only visible surfaces from the scene is a major concern of computer graphics.
- For displaying a realistic scene, it is often necessary to render visible surfaces only. Many algorithms have been devised to find visible surfaces efficiently.
- There is a time-space trade-off in different algorithms. Some methods are quick with more memory requirement. Some are space-efficient with higher running time.
- Visible surface detection algorithms are also known as *hidden line* or *hidden surface removal* algorithms. Aim of such algorithms is to identify the visible lines or surfaces of a given set of object from particular viewing direction.

Visible Surface Detection Algorithms

Visible surface detection algorithms may be classified either as object space algorithms or Image-space algorithms.

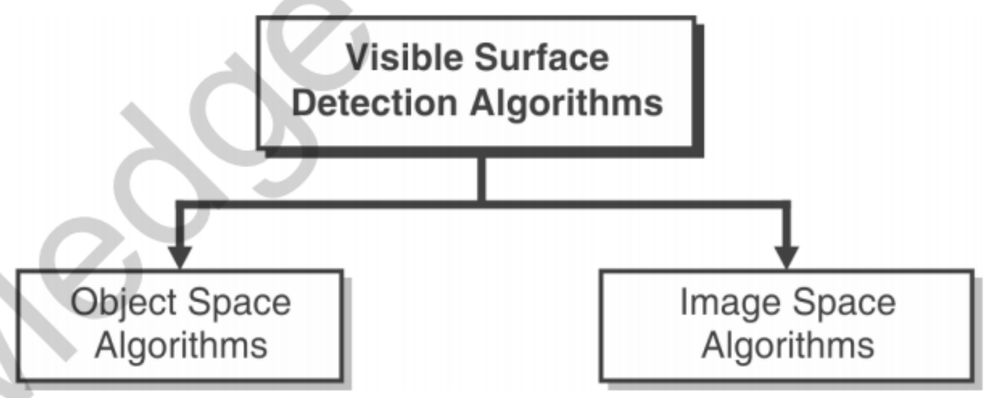


Fig. 9.1.1 : Classification of surface detection algorithms

1. Object Space Method

- Object space methods determine the visible surface of an object by comparing the entire object or part of an object with other objects within the scene. These algorithms operate in physical coordinates system.
- This class of algorithms is faster and performs less computation. Following methods belong to this category :
  - Backface detection
  - Painter's algorithm
  - Robert's algorithm

2. Image Space Methods

- In the image space method, visibility of the object is decided by pixel by pixel comparison of overlapping objects. The pixel of the nearest object to the viewer is selected for display. These algorithms operate in a screen coordinate system. Such algorithms are more precise but computationally intense. Following methods belong to this category :
  - Depth buffer method

Technical Publication

- Area subdivision method
- Octree method
- Scan line method
- Ray tracing algorithm

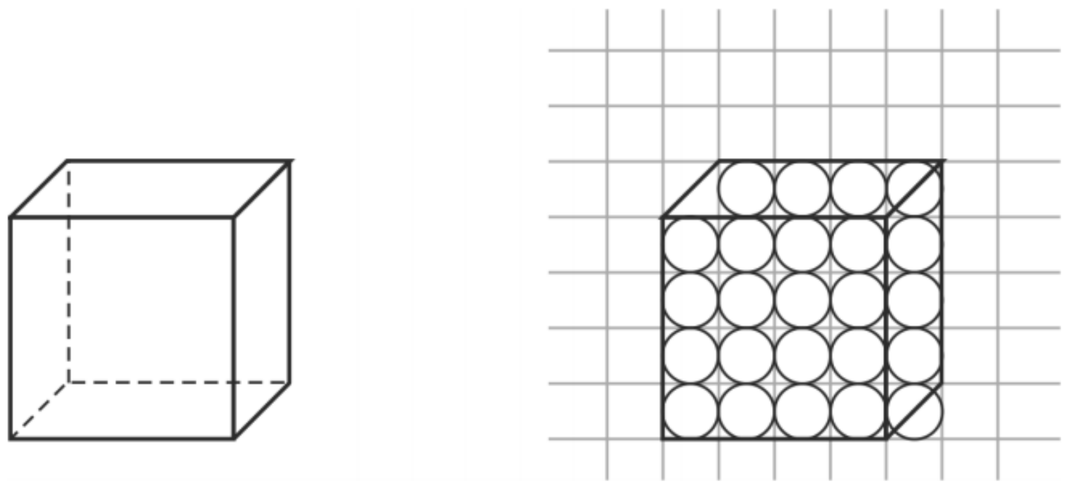


Fig. 9.1.2 : Object space v/s image-space approach

- Most of the visible surface detection algorithms are image space algorithms. However, the choice of algorithm depends on the application. Generally, object space algorithms are preferable.
- Performance of the algorithm can be improved with sorting and coherence property. All the objects in the scene are arranged from front to back in increasing order of their depth from the viewer.
- The object nearest to the viewer may hide the part of the scene, so we do not have to render a portion of hidden objects behind the visible front object.
- Thus, coherence property is used to take advantage of regularity in the scene. In the same way, the individual scan line can also be assumed to have some interval, either hidden or visible. Moreover, two consecutive scan lines also exhibit almost similar properties.

9.2 Back Face Detection and Removal

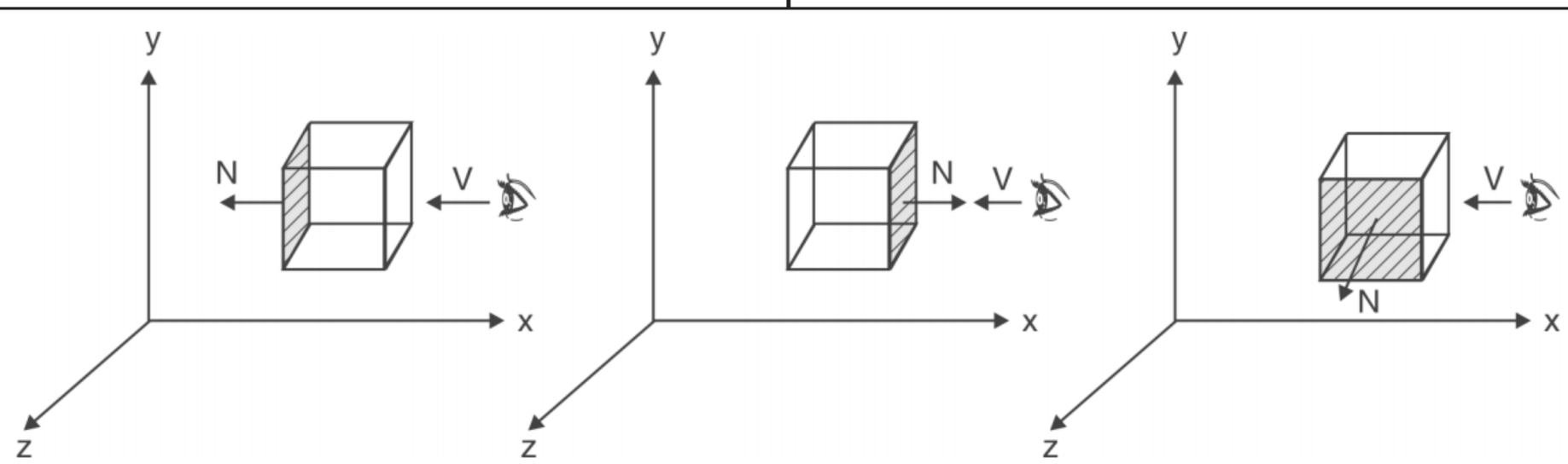
Q. Explain Back-face Detection and Removal.

SPPU - Dec. 17, May 18, 4 Marks

Q. Explain Back-face Removal.

SPPU - Dec. 19, 6 Marks

- Sometimes visible surface detection is also referred to as **hidden surface removal of back face detection**. However, there is a subtle difference in these methods. It is the object space method.
- A visible surface detection algorithm decides which part of the scene is visible and projects it on screen.
- Whereas, back-face detection algorithms consider the entire object and keep removing hidden surfaces from the scene.
- Former algorithms select visible surfaces to display, later category of algorithms removes hidden surfaces from the scene.
- Hidden surfaces can be easily detected from the relationship between viewing direction and surface.
- As shown in Fig. 9.2.1, if outward normal of surface and line of sight are same in this case surface would not be visible.



(a) Back face **(b) Front face** **(c) Parallel face**

Fig. 9.2.1 : Cube surfaces and their normal

TechKnowledge Publications

- As shown in Fig. 9.2.1(a), the shaded surface is parallel to YZ plane and it's outward normal (N) points in negative X-direction which is same as the viewing direction (V).

  So we consider,

$$N = (-1, 0, 0) \text{ and } V = (-1, 0, 0).$$

$$N \cdot V = (-1, 0, 0) \cdot (-1, 0, 0)$$

$$= 1 + 0 + 0$$

$$= 1$$

$$> 0, \text{ so surface is hidden.}$$

- For Fig. 9.2.1(b), the shaded surface is parallel to YZ plane and it's outward normal is in the positive X direction.

  So, $N = (1, 0, 0)$ and $V = (-1, 0, 0)$

$$N \cdot V = (1, 0, 0) \cdot (-1, 0, 0)$$

$$= -1 + 0 + 0$$

$$= -1$$

$$< 0, \text{ so surface is visible.}$$

- Here Normal to the plane and viewing direction are parallel to each other but they are in the opposite direction.
- In Fig. 9.2.1(c), the shaded surface is parallel to the XY plane, and its normal is in the positive Z direction. So the surface is parallel to viewing direction and normal to the surface and viewing direction are perpendicular to each other.

  So, $N = (0, 0, 1)$ and $V = (-1, 0, 0)$

$$N \cdot V = (0, 0, 1) \cdot (-1, 0, 0)$$

$$= 0 + 0 + 0$$

$$= 0,$$

  so surface is parallel to viewing direction

- If the angle between viewing direction and normal to the plane is between + 90° to – 90°, then the surface is visible, otherwise it is hidden and should be removed.
- If viewing direction is along Z-axis then let's say view direction vector is $V = (0, 0, V_z)$. Assume that the normal vector is defined by vector $N = (A, B, C)$.

$$N \cdot V = (A, B, C) \cdot (0, 0, V_z) = V_z \cdot C$$

- If we consider the right hand viewing system, with viewing direction in the negative Z-axis, then the surface is back faced if $C \leq 0$ and surface is visible if $C > 0$.
- For convex polyhedron like a cube, surfaces are either fully visible or completely hidden.
- But concave polyhedrons like shown in Fig. 9.2.2 may have some surfaces partially visible, for that we need to perform some more tests to decide which surface is visible and what portion of it is visible.

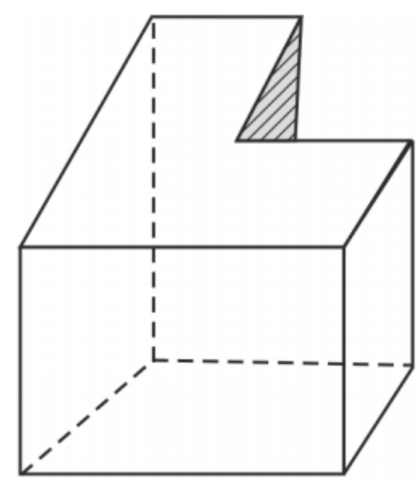


Fig. 9.2.2 : Concave polyhedron

9.3 Depth Buffer (z)

Q. Write a short note on Z-buffer.

SPPU - May 16, May 17, May 18, Dec. 18, 2 Marks

Q. Describe the Z-buffer hidden surface algorithm.

SPPU - May 19, 4 Marks

Q. Explain Z-buffer with their advantage.

SPPU - May 17, 3 Marks

- **Depth buffer method** is also known as the **z-buffer method**. This is an *image space* approach.
- It is a simple, widely used and efficient method to decide the visibility of the pixel.
- It compares the depth of the pixels of overlapping objects into the scene and renders the nearest pixel to the viewer.

Tech Knowledge Publications

- This algorithm is more suited to polygons, but it can also be extended for objects with non-planar surfaces with additional computation.
- As shown in Fig. 9.3.1, depth of corresponding pixels of overlapping objects are compared and the nearest pixels are projected.
- This algorithm is generally applied on normalized coordinates, where all values fall between 0 and 1.
- This algorithm is an extension of the idea of the frame buffer. Implementation requires two buffers of the size of screen resolution.
- One buffer is used to keep track of the depth of visible pixel called **depth buffer** (z-buffer), and another is used to store its corresponding intensity value known as **intensity buffer** (i-buffer).
- In the beginning, z-buffer is initialized with zero and i-buffer is initialized with the background color.
- The scene is scanned pixel by pixel, the depth of each object is calculated and compared. Depth of the nearest pixel to the viewer is stored in z-buffer and its intensity is stored in i-buffer at corresponding spatial location(x, y).
- If no object exists at location (x, y) in the scene then z is set to zero and intensity will be set to background color for that pixel.

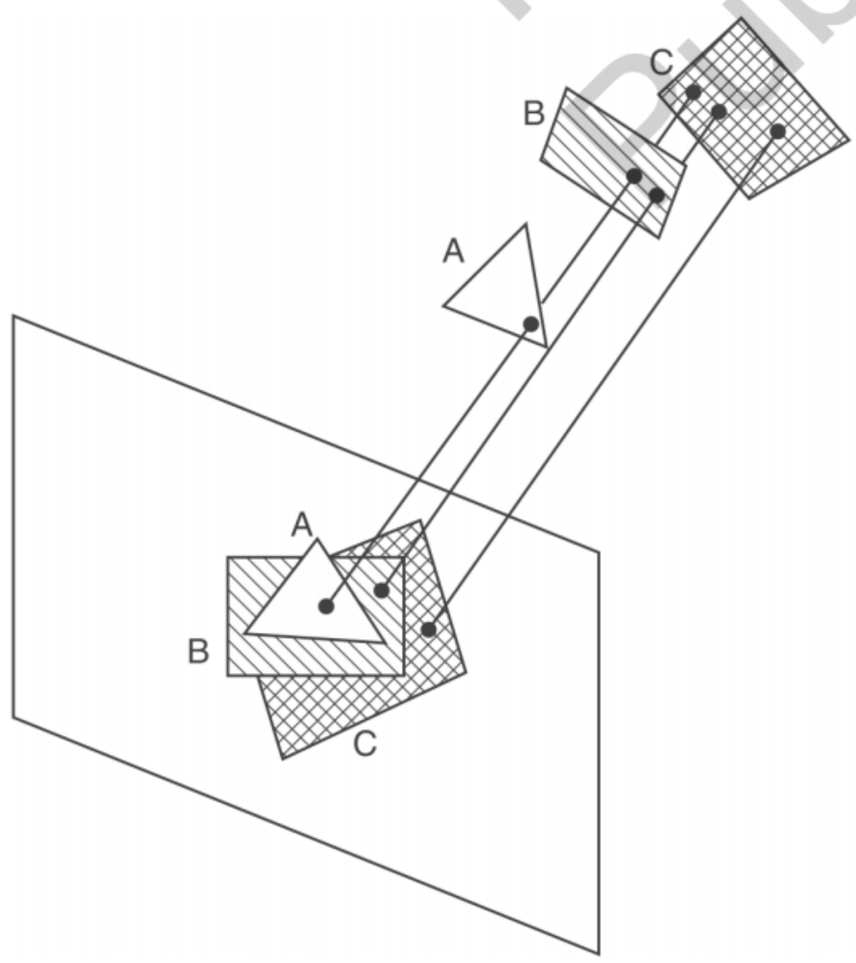


Fig. 9.3.1 : Projection of objects on plane

Algorithm for depth buffer technique is demonstrated here :

```
Algorithm DEPTH_BUFFER()

//Initialization
for each location (x, y) in buffers do
        Z_buffer (x, y) ← – ∞      // depth of farthest plane
        Ibuffer (x, y) ← Ibackground
end

// Rasterization
for  each polygon P in scene do
      for  each pixel (x, y) in polygon P do
            p_z ← Z value of polygon at (x, y)
            if   p_z > Z_buffer (x, y) then
            Z_buffer (x, y) ← P_z
            I_buffer(x, y) ← P_color
            end
      end
end
```

For any plane, z can be calculated as follows :

- Equation of plane is Ax + By + Cz + D = 0. Where (x, y, z) is pixel on plane and A, B, C, D are constants describing the plane.

$$\therefore \quad z_{old} = \frac{-Ax - By - D}{C}$$

- On scan line Y = y, next pixel would be (x + 1, y). The depth at a pixel (x + 1, y) is given as,

$$z_{new} = \frac{-A(x+1) - By - D}{C}$$

$$= \frac{-Ax - By - D - A}{C}$$

$$= \frac{-Ax - Bx - D}{C} - \frac{A}{C}$$

$$z_{new} = z_{old} - \frac{A}{C}$$

Tech Knowledge Publications

– A and C are constant so A/C ratio is also a constant, we need only one subtraction to find z at adjacent pixel (x + 1, y) on the same scan line.

– Similarly, z for the point (x, y) at next scan line Y = y + 1 would be,

$$z_{new} = \frac{-Ax - B(y+1) - D}{C}$$

$$= \frac{-Ax - Bx - D}{C} - \frac{B}{C}$$

$$= z_{old} - \frac{B}{C}$$

Advantages

1. Simple to understand.
2. Easy to implement.
3. No restriction on the number of polygons in the scene.
4. Work for dynamic scene.
5. Sorting of objects is not required.
6. Fast with hardware support.

Disadvantages

1. Require more memory.
2. Time-consuming.

   $$\text{Time} = O(n \cdot d^2)$$

   Where d^2 is the size of the scene, and n is a number of objects.
3. Prone to aliasing effect due to rasterization.
4. If we alter the order of polygon for the processing, then also algorithm ends up with the same result.

9.4 Depth Sorts (Painter)

Q. Explain the Painter's hidden face removal algorithm. **SPPU - May 17, 6 Marks**

Q. How Painter algorithm is useful in hidden surface removal? Explain with their advantages. **SPPU - May 18, 6 Marks**

Q. Write short notes on the following back face removal algorithm: Painter's algorithm. **SPPU - Dec. 18, 2 Marks**

Q. Explain the Painter's algorithm. **SPPU - May 19, Dec. 19, 3 Marks**

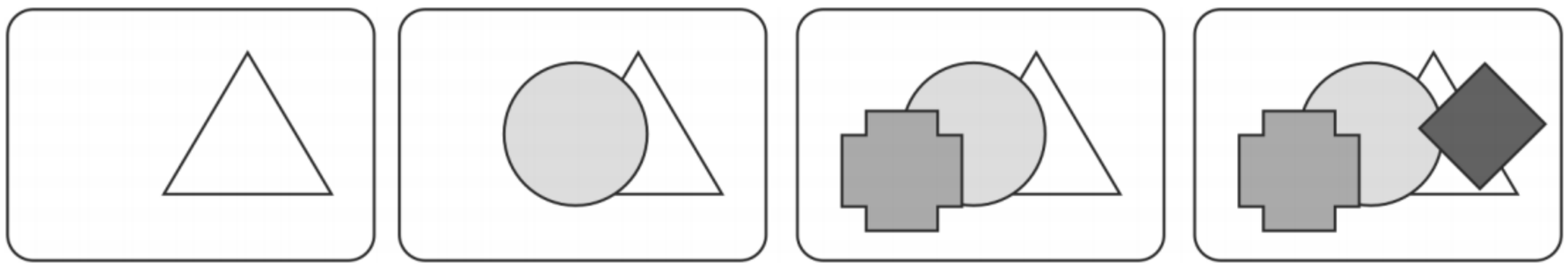
– Depth sorting method is a hybrid approach, it utilizes both, image space and object space operations. This method works as follow :

  - Sort the surfaces in decreasing depth using object space approach.
  - Scan covert surfaces starting from maximum depth, using the image space approach.

– This algorithm works the same as an artist does painting and hence it is coined as the **Painter's algorithm**.

– Artist first paints the background, then adds most distant objects in the scene, then nearest objects are added and the process continues.

– Finally, it paints foreground objects and the process stops. Each color layer overlaps the portion of the previous layer.

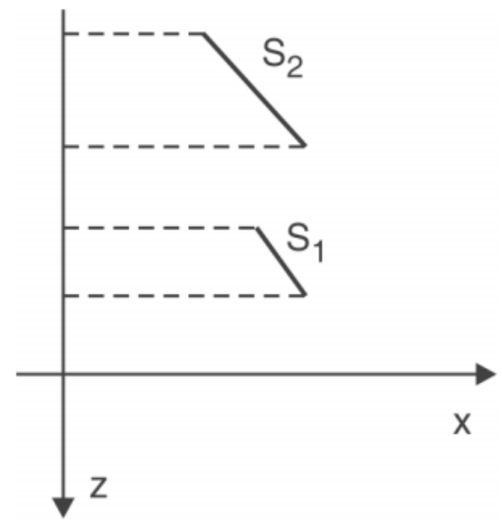


(a) (b) (c) (d)

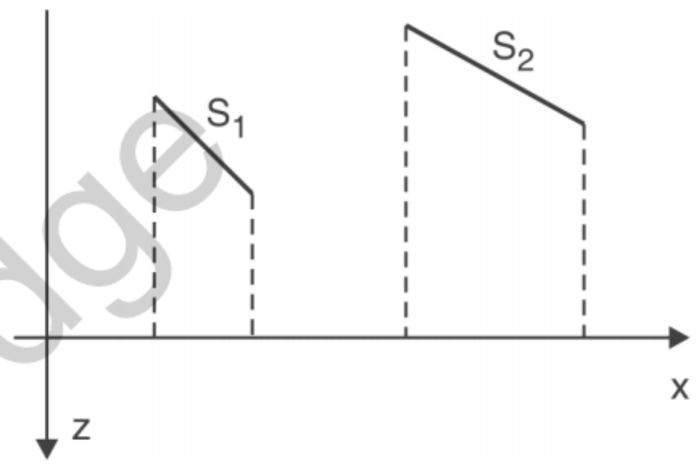
Fig. 9.4.1 : Artist's approach for drawing objects

Techknowledge Publications

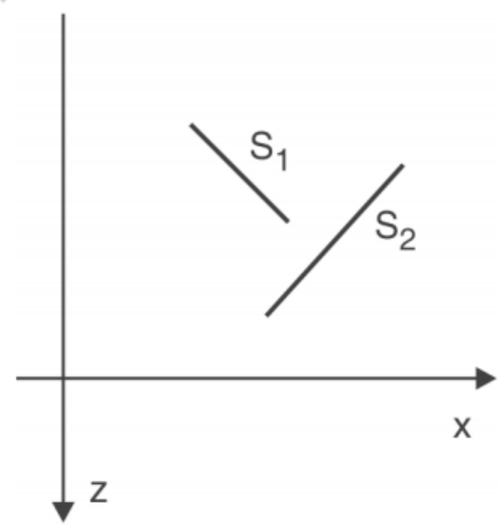
- Analogues to this, depth sort method also start filling the depth buffer with an intensity value of the farthest surface. Processing surfaces from most distant to nearest to the viewer, for each surface depth buffer, is filled up. The portion of the frame buffer is overlapped every time.
- There are many ways to process the surfaces. Let us consider that viewing direction is Z-axis. All surfaces are sorted on their Z values.
- Surface S_1 is compared with remaining all surfaces to check depth overlap. If there is no depth overlapping, S_1 is the farthest surface and is scan converted.
- Fig. 9.4.2(a) shows the two surfaces overlapping in XY plane without depth overlapping. Such surfaces are processed without any further processing.
- If there is depth overlapping, we need to do some extra processing to check if surfaces can be reordered. We perform the following four tests if depth overlap is detected for surface S_1 and S_2. If any of the tests are true, reordering is not required Tests are listed in increasing level of difficulty.
  - Extents (Bounding rectangles) of S_1 and S_2 in the XY plane do not overlap.
  - Surface S_1 is completely behind overlapping surface S_2 from viewing direction
  - Overlapping surface S_2 is completely in front of surface S_1 relative to viewing direction
  - The bounding box of projections of S_1 and S_2 on view plane do not overlap.
- We perform the above tests in the same order, if any of the tests are true, we select next surface S_i to compare with S_1. If all overlapping surfaces to S_1 pass these tests, S_1 is the farthest surface and it is scan converted without reordering.



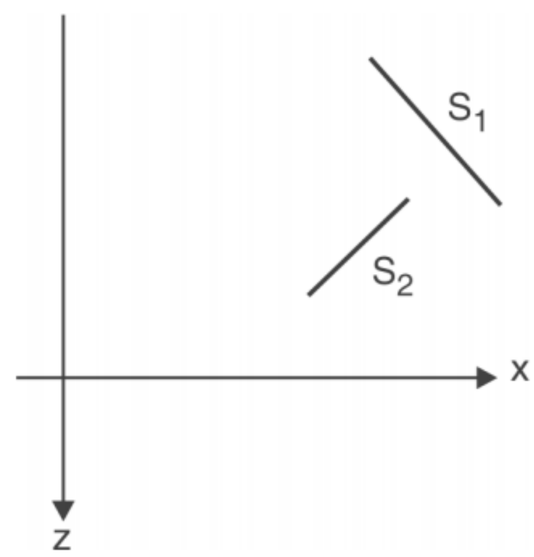
(a) Overlap in X but not depth overlap



(b) Surfaces with depth overlap but no overlap in X-direction



(c) Surface S_1 is completely behind S_2



(d) S_2 is completely in front of S_1but S_1 is not completely behind S_2

Fig. 9.4.2 : Different surface positions

Technknowledge Publications

- In test 1, we check the overlapping of two surfaces in X and Y direction. If they do not overlap in XY plane, they can be rendered without processing even though they overlap in the Z direction. Fig. 9.4.2(a) shows such two surfaces.

- To check the validity of test 2 and 3, we put all the vertices of the surface S_1 in-plane equation of S_2. If all vertices of S_1 are behind S_2, the sign of evaluated plane equation would be negative and if all points of S_1 are in front of S_2 then sign would be positive. For any of the vertex, if sign changes, implies they need reordering.

- If none of the first three conditions is true, we evaluate the last condition. Two surfaces may or may not overlap even though their extent overlap. As shown in Fig. 9.4.3(a), the extent of two surfaces are overlapping but the surfaces themselves do not overlap, whereas in Fig. 9.4.3(b), do overlap.

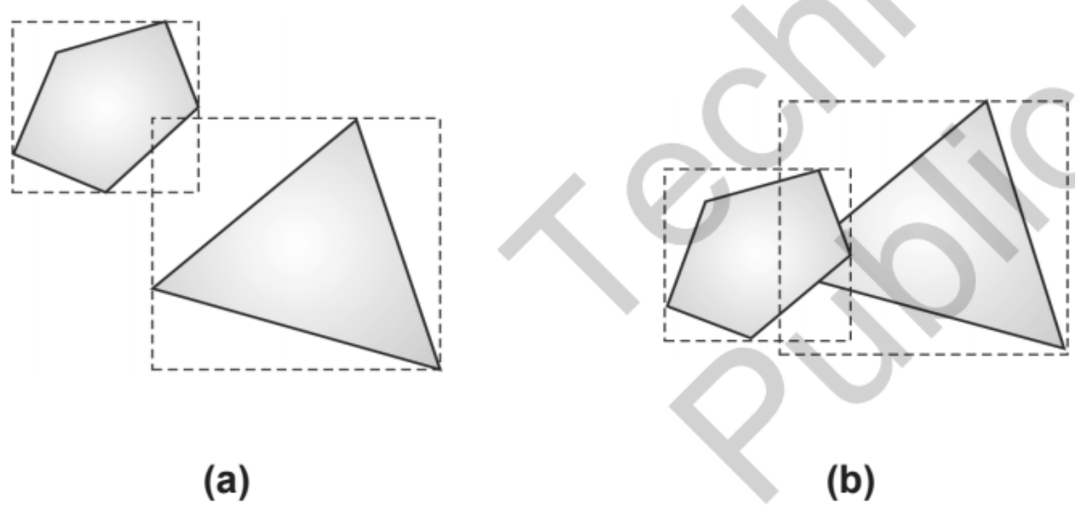


Fig. 9.4.3 : Polygons with overlapping extents

- If all four conditions fail, we interchange the role of surface S_1 and S_2 and perform all four tests again. On failure of all conditions again, we divide the surface into two parts to avoid cyclic overlap and continue the process.

9.5 Area Subdivision (Warnock)

Q. Explain Warnock's hidden face removal algorithm.

SPPU - May 17, 6 Marks

Q. How Warnock algorithm is useful in hidden surface removal? Explain with their advantages.

SPPU - May 18, 6 Marks

Q. Explain Warnock's algorithm.

SPPU - May 19, Dec. 19, 4 Marks

- This is a divide and conquer approach. It is also known as **area subdivision method**. It utilizes the tendency of human eye-brain information processing. Human brain spends less time on the area with little information, and more time and efforts are spent on an area containing more information. Warnock's algorithm takes advantage of *area coherence*.

- Warnock's algorithm evaluates the relationship between the window (area) in the image (object). If the window is not empty, a subdivision of the window is carried out until predefined level of division is achieved or window contains information simple enough to resolve.

- There are four possible relationships that a surface can have with an area of subdivided view plane. These relationships are described in Fig. 9.5.1.

- **Surrounding polygons** are completely containing the area (White square) of interest (Fig. 9.5.1 (a))

- **Intersecting polygons** intersect the area (Fig. 9.5.1 (b))

- **Contained polygons** are completely inside the area (Fig. 9.5.1 (c))

- **Disjoint polygons** are completely outside the area (Fig. 9.5.1 (d))

Techknowledge Publications

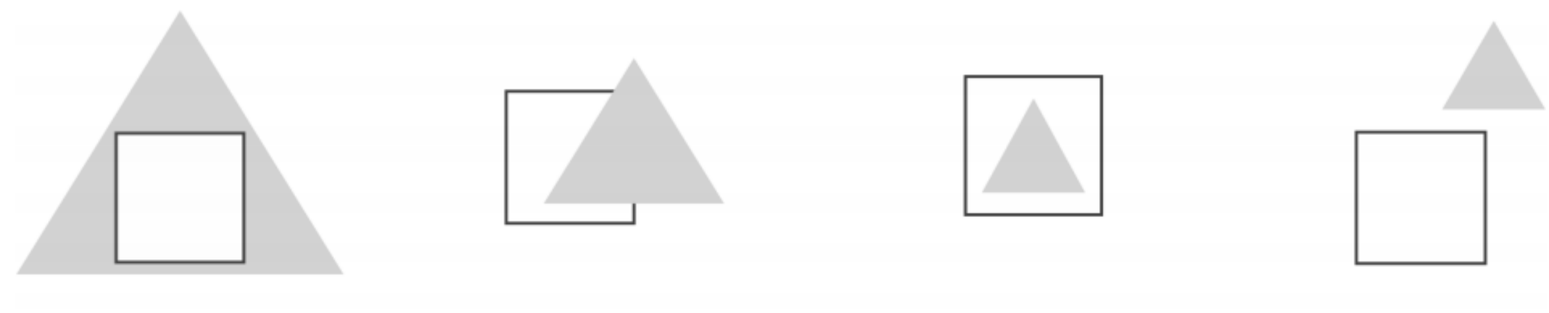


Fig. 9.5.1 : Relationship between two polygons

- If polygons are disjoint (Fig. 9.5.1 (d)), polygon has no influence on the area of interest. The part of the intersecting polygon which is outside the area is also irrelevant. The part of intersecting polygon which is inside the area can be treated same as the contained polygon.
- Surface visibility can be decided by the four relationships illustrated in Fig 9.5.1. If any of the following case is true, no further subdivision of area is needed.

  Case 1 : If all the polygons are disjoint, fill the area with background color.

  Case 2 : If there is only one contained or intersecting polygon, then area is first filled with background color followed by the part of the polygon contained in the area.

  Case 3 : If there is only one surrounding polygon then fill the area with polygon color.

  Case 4 : If there are more than one intersecting, overlapping or inside polygons, more processing is required.
- Case 4 is further described in Fig. 9.5.2.
- In Fig. 9.5.2, all intersection point of surrounding polygon is near to viewer than the rest of all intersection points of intersecting and contained polygon. So the area of interest is colored with a surrounding polygon.
- However, in Fig. 9.5.2, surrounding polygon is in front of remaining polygons but it's all intersection points are not close to the viewer.
- Depth sort algorithm accepts this case, but Warnock's algorithm performs subdivision until one of the four relations holds true or termination criteria are matched. After performing subdivision, no need to examine disjoint (case 1), intersecting (case 2) and surrounding (case 3) polygons. Only those cells are subdivided which contains multiple polygons (case 4).

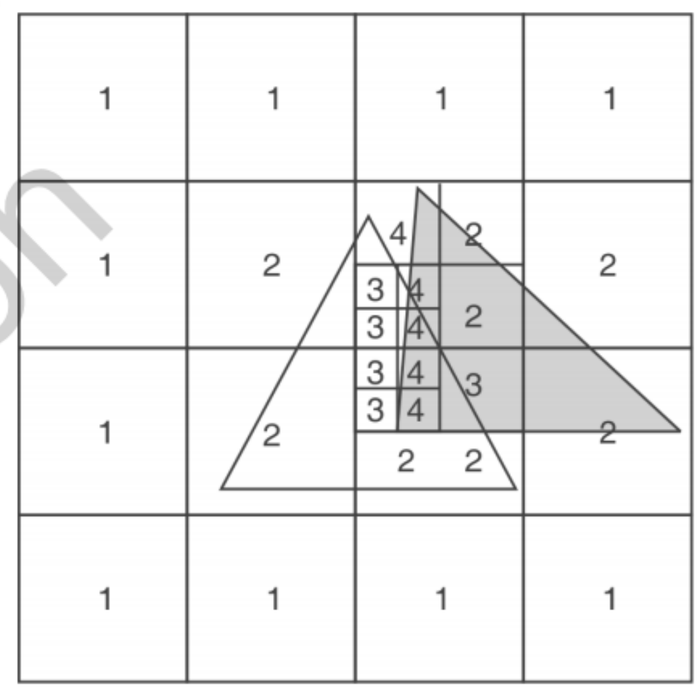


Fig. 9.5.2 : Equal area subdivision

- This algorithm operates in both, object space and image space. After a certain level of subdivision, the scene may contain many small polygons. At that level, we can apply the image-space algorithm to determine the closest point to the viewer. Typically, the subdivision is stopped when screen resolution reaches to 1024 by 1024 (i.e. 10 level of subdivision).
- After that, the depth of the individual area is computed and the area is displayed with the polygon color which is nearest to the user. Antialiasing methods are applied to further improve the display quality.

Technknowledge Publication

– Equal area subdivision is shown in Fig. 9.5.3 creates unnecessary divisions. As an alternative to that, we can use division about vertex as shown in Fig. 9.5.3. In this approach, subdivision takes place at vertex only. In the mentioned diagram, First subdivision is carried out at point P followed by second division at point Q.

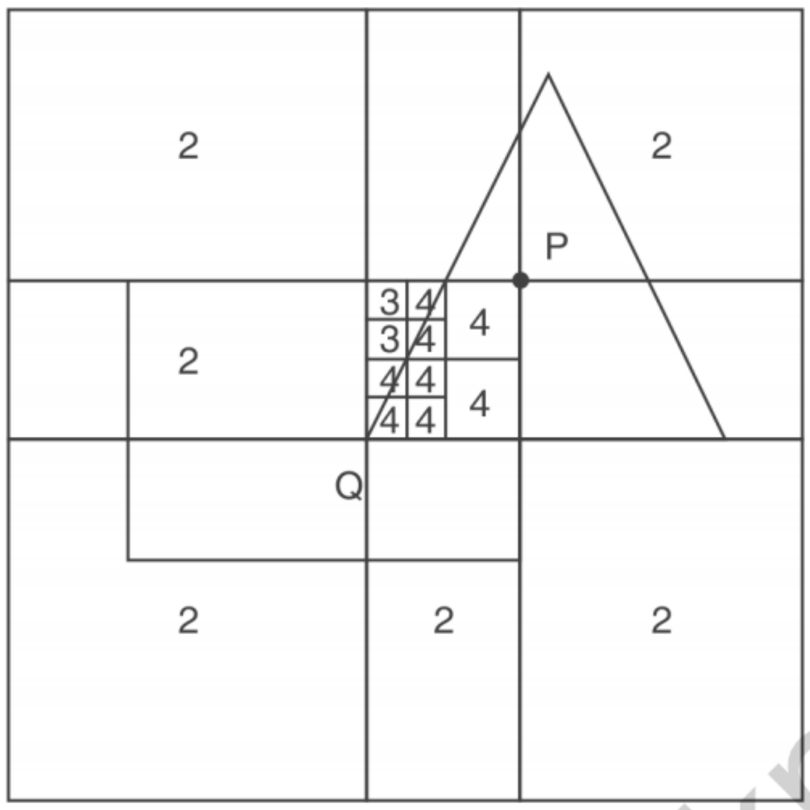


Fig. 9.5.3 : Subdivision from vertex

Applications

– Used for hidden surface removal.

– Can make rendering possible for the complicated image by subdividing it.

– Quickly process the area with little information.

Review Questions

Q. 1 Classify visible surface detection algorithms.

Q. 2 Write short note on back surface detection method.

Q. 3 Write application of Area Subdivision method.

Q. 4 Explain depth buffer method.

Q. 5 Write algorithm for projection of objects on plane with diagram.

Q. 6 How z can be calculated for any plane .

Q. 7 Write advantages and disadvantages for depth buffer method.

Q. 8 Explain scan line method.

Q. 9 Write short note on Area Subdivision method.

Q. 10 Explain depth sorting method.

❑❑❑

Technical Knowledge Publication

CHAPTER 10

UNIT V

Curves and Fractals

Syllabus

Curves : Introduction, Interpolation and Approximation, Blending function, B-Spline curve, Bezier curve,

Fractals : Introduction, Classification, Fractal generation: snowflake, Triadic curve, Hilbert curve, Applications.

10.1 Curves

10.1.1 Introduction

- Real-world objects are not always made up of basic primitives like line, circle, ellipse, square etc.
- Usually, natural objects are very irregular in shape and size. It may not be possible to describe such shapes with standard mathematical formulas.
- Through the curve generation procedure, it is possible to model any random shape. However, the process of curve generation requires detail mathematical analysis of object shape. The process is very complex.
- 3D lines, curves or surfaces are produced either using the mathematical function or by defining a set of data points, called control points. Curves or surfaces may be represented using the parametric or non-parametric form.
- When the function is used to plot the curve, graphics packages evaluate predefined equations for passed arguments and generate pixels along the desired path. The surface is often derived with the help of tessellation. It is first approximated as a polygon mesh. This is done by triangulation. Each triangle is defined using three vertices.
- Four or more vertices may not form a single plane. Quadratic and super quadratics are produced using this approach.
- When an object is defined using a set of discrete points, the shape is estimated by regression methods. Regression techniques try to generate a best-fitting curve for a given set of control points. This is useful in digitizing coordinates, to design new shapes, to define a smooth path for animation motion etc.

10.1.2 Interpolation and Approximation

Q. Write a short note on interpolation and approximation. **SPPU - Dec. 18, 4 Marks**

- As we discussed, the curve is specified by a set of control points. Position of control points controls the shape of the curve. Fitting a single curve of the higher degree to all the control points is difficult. Normally, the big curve is approximated by joining piecewise polynomials of lower degree, which approximates few control points only.
- If the curve passes through the control points, it is called **interpolation**. Interpolation curves are used in animation and specifying camera motion. It is also used in digitizing the coordinates.

Technical Publication

- If the curve does not pass through the control points and approximate the shape, it is called **approximation** or **extrapolation**. Such curves are used to estimate the shape of the object surface. Fig. 10.1.1 shows the geometry of both methods.
- Manipulating spline curves using control point representation is very easy. The designer can adjust the shape of the curve by repositioning the control point. It is highly preferable in interactive graphics design. Applying transformation on each point of the curve is equivalent to applying transformation first to control points and redrawing the curve.

(a) Interpolation **(b) Approximation**

Fig. 10.1.1 : Interpolation v/s approximation

- Many curves satisfy the convex hull property. The **convex hull** is the enclosed polygon which passes through the outermost control points. We say the curve is satisfying convex hull property if the entire curve lies within the convex hull.

(a) Curve satisfies convex hull property **(b) The curve does not satisfy convex hull property**

Fig. 10.1.2 : Convex hull

- Curve having convex hull property indicates that the curve grows smoothly within the convex hull. Curves which do not satisfy the convex hull property may oscillate arbitrarily.
- Clipping the curve satisfying convex hull property is easy. Trivial acceptance and rejection conditions are easy to evaluate for them. If bounding box of the convex hull is inside clipping region, no need to clip the curve. And if a bounding box is completely outside clipping window, clip the entire curve.

10.1.3 Blending Function

- Let us assume that the curve passes through the n control points (x_i, y_i, z_i), i = 1, 2, …, n. The curve is represented by parametric form. Lagrange polynomials are used for polynomial interpolation.
- Coordinates of each point on the curve is given by the sum of each point weighted by the blending functions.

$$f_x(u) = \sum_{i=1}^{n} x_i \cdot B_i(u)$$

Tech Knowledge Publications

$$f_y(u) = \sum_{i=1}^{n} y_i \,.\, B_i(u)$$

$$f_z(u) = \sum_{i=1}^{n} x_i \,.\, B_i(u)$$

– **Function $B_i(u)$** is called **blending function**. Blending function determines the effect of ith control point for given 'u' on the rest of the curve points. Parameter u varies from 0 to 1.

– We can force the curve to pass form given point i, by setting $B_i(u) = 1$. This value of u has complete control over the curve.

– Following blending functions will be used to interpolate the four control points :

$$B_1(u) = \frac{(u-1)(u-2)}{(-1)(-2)(-3)}$$

$$B_2(u) = \frac{(u+1)(u-1)(u-2)}{(1)(-1)(-2)}$$

$$B_3(u) = \frac{(u+1)\,u(u-2)}{(2)(1)(-1)}$$

$$B_4(u) = \frac{(u+1)\,u(u-1)}{(3)(2)(1)}$$

For $u = -1$, $B_1(u) = -1$, $B_2(u) = B_3(u) = B_4(u) = 0$

For $u = 0$, $B_2(u) = 1$, $B_1(u) = B_3(u) = B_4(u) = 0$

For $u = 1$, $B_3(u) = -1$, $B_1(u) = B_2(u) = B_4(u) = 0$

For $u = 2$, $B_4(u) = 1$, $B_1(u) = B_2(u) = B_3(u) = 0$

Individual coordinates of the points on the curve in space can be computed as,

$$x = x_1B_1(u) + x_2B_2(u) + x_3B_3(u) + x_4B_4(u)$$

$$y = y_1B_1(u) + y_2B_2(u) + y_3B_3(u) + y_4B_4(u)$$

$$z = z_1B_1(u) + z_2B_2(u) + z_3B_3(u) + z_4B_4(u)$$

A generalized formula for the bending function is given as,

$$B_i(u) = \frac{(u+1)(u)(u-1)\ldots[u-(i-3)]\,[u-(i-1)]\ldots[u-i-2]}{(i-1)(i-2)(i-3)\ldots(1)(-1)\ldots(i-n)}$$

10.1.4 Bezier Curve

Q. Explain the Bezier curve. List its properties.

SPPU - Dec. 15, May 17, May 19, 6 Marks

Q. Write a short note on the blending function of the Bezier curve. **SPPU - May 18, Dec. 18, 4 Marks**

– Bezier curve was first invented by Pierre Bezier. Due to its nice properties, Bezier curves are widely used in engineering designs and CAD packages.

– With cubic polynomials, Bezier curve can approximate any number of control points. In the Bezier curve, the degree of polynomial depends on a number of control points used to generate curve segment.

– In the Bezier curve, the degree of the polynomial is always one less than a number of control points. Fig. 10.1.3 shows Bezier curves with three, four and five control points having a degree of polynomial two, three and four, respectively.

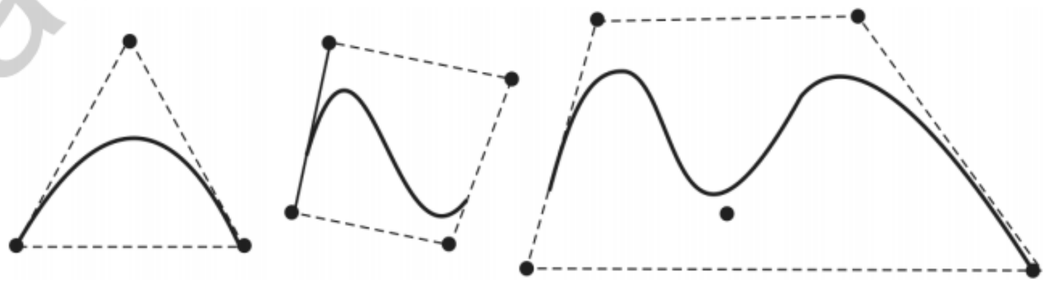


Fig. 10.1.3 : Bezier curves of degree 2, 3 and 4 with their convex hull

– With n + 1 control points, parametric equation of Bezier curve approximating points p_0 to p_n,

$$P(t) = \sum_{i=0}^{n} p_i \,.\, BEZ_{i,n}(t),\ 0 \le t \le 1$$

Blending functions of Bezier curves are directly derived from **Bernstein polynomials**.

$$BEZ_{i,n}(t) = C(n, i)t^i (1-t)^{n-i}$$

C(n, i) is the binomial coefficient :

$$C(n, i) = \frac{n!}{(n-i)!\,i!}$$

Technknowledge Publication

Bezier curve equation for each direction is written as,

$$x(t) = \sum_{i=0}^{n} x_i \cdot BEZ_{i,n}(t), 0 \le t \le 1$$

$$y(t) = \sum_{i=0}^{n} y_i \cdot BEZ_{i,n}(t), 0 \le t \le 1$$

$$z(t) = \sum_{i=0}^{n} z_i \cdot BEZ_{i,n}(t)\ 0 \le t \le 1$$

Properties of Bezier Curves

- Degree of the curve is one less than a number of control points
- Always interpolates first and last control points and approximates remaining two.
- The slope of the derivative at the beginning is along the line joining the first two points and slope of the derivative at the end is along the line joining last two points.
- Bezier curve always satisfies the convex hull property.
- At any parameter value t, sum all four Bezier blending function is always 1, i.e.,

$$\sum_{i=0}^{n} BEZ_{i,n}(t) = 1$$

- Polynomial smoothly follows the control points without much oscillation.
- Bezier curves do not have local control; repositioning one control point changes the entire curve.
- The curve is invariant under affine transformation.
- The basis functions are real.
- The curve exhibits variation diminishing property, i.e. any line intersects the Bezier curve at most as often as that line intersects the polygon interpolating control points.
- Bezier curve can fit any number of control points.
- Reversing the order of control points yield the same Bezier curve.

Applications of Bezier Curves

Bezier curve is used in many design applications. Bezier curves are used to model shapes and surfaces of automobiles, engineering, architectures, and aeronautics objects. Few of the applications are listed where the Bezier curve is being used to model the shapes.

- Computer graphics
- Computer animation
- Engineering design
- CAD packages
- Modelling
- Graphics packages like photo shop
- Designing automobile parts

Cubic Bezier Curves

- Cubic Bezier curve is important to discuss. Cubic Bezier curves are the bridge between computation cost and smoothness.
- They are very much flexible and estimate shape nicely. We should specify four control points to generate a cubicBezier curve.
- Blending functions for cubic Bezier curve is derived by putting n = 3 in the equation of Bezier curve,

$$BEZ_{0,3}(t) = (1-t)^3$$

$$BEZ_{1,3}(t) = 3t(1-t)^2$$

$$BEZ_{2,3}(t) = 3t^2(1-t)$$

$$BEZ_{3,3}(t) = t^3$$

- Fig. 10.1.4 shows the plot of blending functions over parameter range $0 \le t \le 1$. For t = 0, only first blending function is non zero and it has value 1. And at t = 1, only last blending function is non zero and its value is 1.

Technknowledge Publication

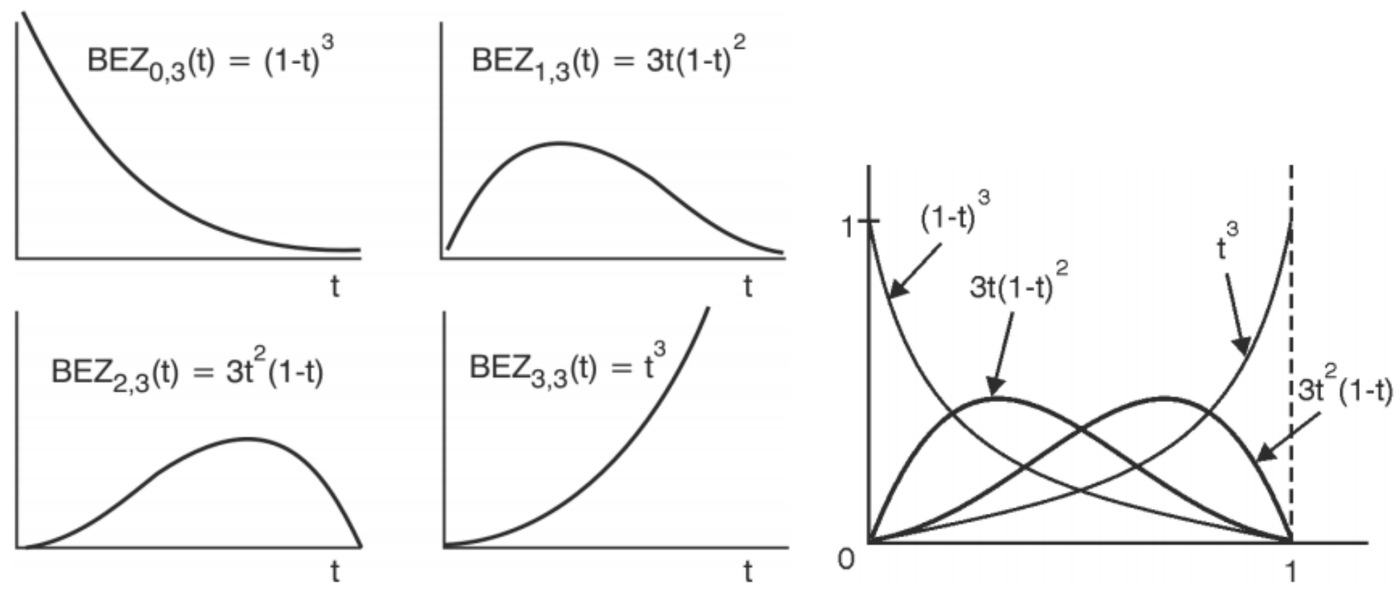


Fig. 10.1.4 : Bezier blending functions

- Thus Bezier curve always interpolates endpoints. For the rest of the values of t, all blending functions contribute to the shape of the curve.
- Blending functions $BEZ_{1,3}$ and $BEZ_{2,3}$ are maximum at t = 1/3 and t = 2/3 respectively. At any value of t, the summation of Bezier blending function is always 1.

Matrix Representation

We can derive a matrix representation of a cubic Bezier curve using blending function representation as below :

$$P(t) = \sum_{i=0}^{n} p_i \cdot BEZ_{i,n}(t),\ 0 \le t \le 1 = p_0 \cdot BEZ_{0,3}(t) + p_1 \cdot BEZ_{1,3}(t) + p_2 \cdot BEZ_{2,3}(t) + p_3 \cdot BEZ_{3,3}(t)$$

$$= p_0 (1-t)^3 + p_1 3t (1-t)^2 + p_2 3t^2 (1-t) + p_3 t^3$$

$$= (1 - 3t + 3t^2 - t^3)p_0 + (3t - 6t^2 + 3t^3) p_1 + (3t^2 - 3t^3)p_2 + t^3 p_3$$

$$P(t) = [t^3\ t^2\ t\ 1] \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$P(t) = T \cdot M_{Bez} \cdot G_{Bez}$$

Where, M_{Bez} and G_{Bez} represent Bezier basis matrix and Bezier geometric matrix respectively.

Limitations of the Bezier Curve

- Bezier curve approximates the control points so it cannot be used in the applications which require interpolating curve.
- Bezier curve does not have local control i.e. changing the position of one control point influences the entire curve.
- Degree of the curve depends on a number of the control point i.e. adding more control points increases the smoothness of the curve at the cost of processing higher degree polynomials.

TechKnowledge Publications

Subdivision Method

- Another method to generate a Bezier curve is a subdivision method. This method generates a Bezier curve by midpoint calculations. Let us derive the Bezier curve defined by control points P_1, P_2, P_3 and P_4.
- Join consecutive control points using straight lines. Suppose midpoint of lines joining points P_1P_2, P_2P_3 and P_3P_4 are L_2, H and R_3 respectively. Naming conventions are chosen to simplify the understanding of computation.

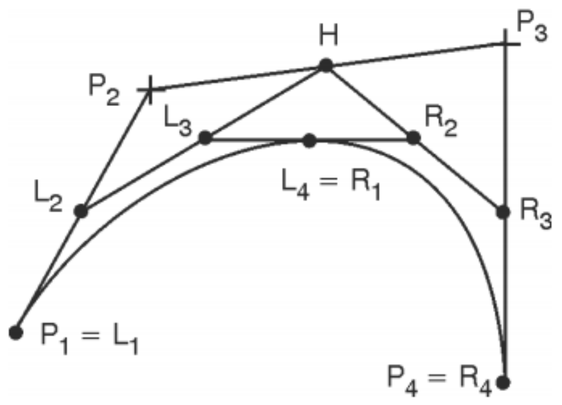


Fig. 10.1.5 : Bezier curve generation using subdivision method

- Join points L_2 and H and name its midpoint L_3. Similarly, join points H and R_3 and name its midpoint R_2. Join points L_3 and R_2 and assign label L_4 to its midpoint ($L_4 = R_1$).
- Recursively draw two Bezier curve segments between control points (L_1, L_2, L_3, L_4) and (R_1, R_2, R_3, R_4) respectively. Both these curve segments are independent Bezier curves.
- Numbers of recursive calls decide the smoothness of the curve. We can derive the matrix representation for this subdivision as follows.

From Fig. 10.1.5,

$$L_1 = P_1 = \frac{8P_1}{8}$$

$$L_2 = \frac{P_1 + P_2}{2} = \frac{4P_1 + 4P_2}{8}$$

$$H = \frac{P_2 + P_3}{2}$$

$$L_3 = \frac{L_2 + H}{2} = \frac{\frac{P_1 + P_2}{2} + \frac{P_2 + P_3}{2}}{2}$$

$$L_3 = \frac{P_1 + 2P_2 + P_3}{4}$$

$$= \frac{2P_1 + 4P_2 + 2P_3}{8}$$

$$R_4 = P_4 = \frac{8P_4}{8}$$

$$R_3 = \frac{P_3 + P_4}{2} = \frac{4P_3 + 4P_4}{8}$$

$$R_2 = \frac{H + R_3}{2} = \frac{\frac{P_2 + P_3}{2} + \frac{P_3 + P_4}{2}}{2}$$

$$= \frac{P_2 + 2P_3 + P_4}{4}$$

$$= \frac{2P_2 + 4P_3 + 2P_4}{4}$$

$$L_4 = R_1 = \frac{L_3 + R_2}{2}$$

$$= \frac{\frac{P_1 + 2P_2 + P_3}{4} + \frac{P_2 + 2P_3 + P_4}{4}}{2}$$

$$= \frac{P_1 + 3P_2 + 3P_3 + P_4}{8}$$

Bezier left subdivision matrix is given as,

$$G_B^L = D_B^L \cdot G_B$$

$$= \frac{1}{8}\begin{bmatrix} 8 & 0 & 0 & 0 \\ 4 & 4 & 0 & 0 \\ 2 & 4 & 2 & 0 \\ 1 & 3 & 3 & 1 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Where, D_B^L is Bezier left subdivision matrix

$$G_B^R = D_B^R \cdot G_B$$

$$= \begin{bmatrix} 1 & 3 & 3 & 1 \\ 0 & 2 & 4 & 2 \\ 0 & 0 & 4 & 4 \\ 0 & 0 & 0 & 8 \end{bmatrix}\begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

Where, D_B^R is Bezier right subdivision matrix.

Technknowledge Publication

Ex. 10.1.1 : Given B_0 [1, 1], B_1 [2 3], B_2 [4 3] and B_3 [3 1] the vertices of a Bezier polygon determine the points on the Bezier curve.

Soln. :

The curve is defined by control points B_0, B_1, B_2, B_3. The curve passes through the endpoints (1, 1) and (3, 1)

Matrix representation of the Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B = [t^3 \;\; t^2 \;\; t \;\; 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 3 \\ 4 & 3 \\ 3 & 1 \end{bmatrix}$$

$$= [t^3 \;\; t^2 \;\; t \;\; 1] \begin{bmatrix} -4 & 0 \\ 3 & -6 \\ 3 & 6 \\ 1 & 1 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [-4t^3 + 3t^2 + 3t + 1 \quad -6t^2 + 6t + 1]$$

Let us find few points on curve for given parameter value,

$$t = 0 \Rightarrow Q(0) = [-4(0)^3 + 3(0)^2 + 3(0) + 1 \quad -6(0)^2 + 6(0) + 1]$$

$$= [1 \;\; 1]$$

$$t = 0.2 \Rightarrow Q(0.2) = [-4(0.2)^3 + 3(0.2)^2 + 3(0.2) + 1 \quad -6(0.2)^2 + 6(0.2) + 1]$$

$$= [-0.032 + 0.12 + 0.6 + 1 \quad -0.24 + 1.2 + 1]$$

$$= [\,1.688 \quad 1.96\,]$$

$$t = 0.5 \Rightarrow Q(0.5) = [-4(0.5)^3 + 3(0.5)^2 + 3(0.5) + 1 \quad -6(0.5)^2 + 6(0.5) + 1]$$

$$= [-0.5 + 0.75 + 1.5 + 1 \quad -1.5 + 3 + 1]$$

$$= [\,2.75 \quad 2.5]$$

$$t = 0.8 \Rightarrow Q(0.8) = [-4(0.8)^3 + 3(0.8)^2 + 3(0.8) + 1 \quad -6(0.8)^2 + 6(0.8) + 1]$$

$$= [-2.048 + 1.92 + 2.4 + 1 \quad -3.84 + 4.8 + 1]$$

$$= [3.272 \quad 1.96]$$

$$t = 1.0 \Rightarrow Q(1.0) = [-4(1.0)^3 + 3(1.0)^2 + 3(1.0) + 1 \quad -6(1.0)^2 + 6(1.0) + 1]$$

$$= [-4 + 3 + 3 + 1 \quad -6 + 6 + 1]$$

$$= [3 \quad 1]$$

Technknowledge Publications

Value of t	Point on curve
t = 0.0	[1 1]
t = 0.2	[1.688 1.96]
t = 0.5	[2.75 2.5]
t = 0.8	[3.272 1.96]
t = 1.0	[3 1]

Ex. 10.1.2 : Find the equation for the Bezier curve, which passes through control points (0, 0) and (– 3, 2) and controlled by (6, 4) and (3, 1). Also find points on curve for t = 0, 0.4, 0.8, 1.

Soln. :

Assume that curve is defined by control points p_0, p_1, p_2 and p_3. The curve passes through the endpoints (0, 0) and (– 3, 2).

So, p_0 = (0, 0) and p_3 (– 3, 2). Let us consider p_1 = (6, 4) and p_2 = (3, 1)

Matrix representation of Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \;\; t^2 \;\; t \;\; 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 6 & 4 \\ 3 & 1 \\ -3 & 2 \end{bmatrix}$$

$$= [t^3 \;\; t^2 \;\; t \;\; 1] \begin{bmatrix} 6 & 11 \\ -27 & -21 \\ 18 & 12 \\ 0 & 0 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [6t^3 - 27t^2 + 18t \qquad 11t^3 - 21t^2 + 12t]$$

Let us find point on curve for given parameter value,

$$t = 0 \Rightarrow Q(0)$$

$$= [6(0) - 27(0) + 18(0) \qquad 11(0) - 21(0) + 12(0)]$$

$$= [0 \quad 0]$$

$$t = 0.4 \Rightarrow Q(0.4)$$

$$= [6(0.4)^3 - 27(0.4)^2 + 18(0.4) \qquad 11(0.4)^3 - 21(0.4)^2 + 12(0.4)]$$

$$= [0.384 - 4.32 + 7.2 \qquad 0.704 - 3.36 + 4.8]$$

$$= [3.264 \quad 2.144]$$

Techknowledge Publication

$$t = 0.8 \Rightarrow Q(0.8)$$

$$= [6(0.8)^3 - 27(0.8)^2 + 18(0.8) \quad 11\cdot(0.8)^3 - 21(0.8)^2 + 12(0.8)]$$

$$= [5.632 - 17.28 + 14.4 \quad 5.632 - 13.44 + 9.6]$$

$$= [2.752 \quad 1.792]$$

$$t = 1 \Rightarrow Q(1) = [6(1)^3 - 27(1)^2 + 18(1) \quad 11(1)^3 - 21(1)^2 + 12(1)]$$

$$= [6 - 27 + 18 \quad 11 - 21 + 12] = [-3 \quad 2]$$

Value of t	Point on curve
t = 0.0	[0 0]
t = 0.4	[3.264 2.144]
t = 0.8	[2.752 1.792]
t = 1.0	[– 3 2]

Ex. 10.1.3 : Construct the Bezier curve of order three with control points $P_1(0, 0)$, $P_2(1, 3)$, $P_3(4, 2)$ and $P_4(2, 1)$. Generate at least five points on the curve.

Soln. :

The curve passes through the endpoints (0, 0) and (2, 1)

Matrix representation of the Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \ t^2 \ t \ 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 3 \\ 4 & 2 \\ 2 & 1 \end{bmatrix}$$

$$= [t^3 \ t^2 \ t \ 1] \begin{bmatrix} -7 & 4 \\ 6 & -12 \\ 3 & 9 \\ 0 & 0 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [-7t^3 + 6t^2 + 3t \quad 4t^3 - 12t^2 + 9t]$$

We have to find at least five points on the curve. Let us find point on curve for parameter step value 0.2.

$$t = 0.0 \Rightarrow Q(0.0)$$

$$= [-7(0.0)^3 + 6(0.0)^2 + 3(0.0) \quad 4(0.0)^3 - 12(0.0)^2 + 9(0.0)]$$

$$= [0 + 0 + 0 \quad 0 - 0 + 0]$$

$$= [0 \quad 0]$$

$$t = 0.2 \Rightarrow Q(0.2)$$

$$= [-7(0.2)^3 + 6(0.2)^2 + 3(0.2) \quad 4(0.2)^3 - 12(0.2)^2 + 9(0.2)]$$

$$= [-0.056 + 0.24 + 0.6 \quad 0.032 - 0.48 + 1.8]$$

$$= [0.784 \quad 1.352]$$

$$t = 0.4 \Rightarrow Q(0.4)$$

$$= [-7(0.4)^3 + 6(0.4)^2 + 3(0.4) \quad 4(0.4)^3 - 12(0.4)^2 + 9(0.4)]$$

$$= [-0.448 + 0.96 + 1.2 \quad 0.256 - 1.92 + 3.6]$$

$$= [1.712 \quad 1.424]$$

$$t = 0.6 \Rightarrow Q(0.6)$$

$$= [-7(0.6)^3 + 6(0.6)^2 + 3(0.6) \quad 4(0.6)^3 - 12(0.6)^2 + 9(0.6)]$$

$$= [1.512 + 2.16 + 1.8 \quad 0.864 - 4.32 + 5.4]$$

$$= [5.472 \quad 1.944]$$

$$t = 0.8 \Rightarrow Q(0.8)$$

$$= [-7(0.8)^3 + 6(0.8)^2 + 3(0.8) \quad 4(0.8)^3 - 12(0.8)^2 + 9(0.8)]$$

$$= [-3.584 + 3.84 + 2.4 \quad 2.048 - 7.68 + 7.2]$$

$$= [2.656 \quad 1.568]$$

$$t = 1.0 \Rightarrow Q(1.0)$$

$$= [-7(1.0)^3 + 6(1.0)^2 + 3(1.0) \quad 4(1.0)^3 - 12(1.0)^2 + 9(1.0)]$$

$$= [-7 + 6 + 3 \quad 4 - 12 + 9]$$

$$= [2 \quad 1]$$

Value of t	Point on curve
t = 0.0	[0 0]
t = 0.2	[0.784 1.352]
t = 0.4	[1.712 1.424]
t = 0.6	[5.472 1.944]
t = 0.8	[2.656 1.568]
t = 1.0	[2 1]

Techknowledge Publication

Ex. 10.1.4 : Given four control points (10, 10), (15, 15), (20, 15) and (30, 10), find the points to plot Bezier curve by using step size as 0.2.

Soln :

Assume that curve is defined by control points p_0, p_1, p_2 and p_3. The curve passes through the endpoints (10, 10) and (30, 10),

So, p_0 = (10, 10) and p_3 (30, 10). Let us consider p_1 = (15, 15) and p_2 = (20, 15)

Matrix representation of the Bezier curve is,

$$Q(t) = T \cdot M_B \cdot G_B$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 10 & 10 \\ 15 & 15 \\ 20 & 15 \\ 30 & 10 \end{bmatrix}$$

$$= [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} 5 & 0 \\ 0 & -15 \\ 15 & 15 \\ 10 & 10 \end{bmatrix}$$

∴ Equation of Bezier curve would be,

$$Q(t) = [5t^3 + 15t + 10 \quad -15t^2 + 15t + 10]$$

Let us find point on curve for parameter step value 0.2.

$$t = 0.0 \Rightarrow Q(0.0)$$

$$= [5(0.0)^3 + 15(0.0) + 10 \quad -15(0.0)^2 + 15(0.0) + 10]$$

$$= [0 + 0 + 10 \quad -0 + 0 + 10]$$

$$= [10 \quad 10]$$

$$t = 0.2 \Rightarrow Q(0.2)$$

$$= [5(0.2)^3 + 15(0.2) + 10 \quad -15(0.2)^2 + 15(0.2) + 10]$$

$$= [0.04 + 3 + 10 \quad -0.6 + 3 + 10]$$

$$= [13.04 \quad 12.4]$$

$$t = 0.4 \Rightarrow Q(0.4)$$

$$= [5(0.4)^3 + 15(0.4) + 10 \quad -15(0.4)^2 + 15(0.4) + 10]$$

$$= [0.32 + 6 + 10 \quad -2.4 + 6 + 10]$$

$$= [16.32 \quad 13.6]$$

Techknowledge Publications

$$t = 0.6 \Rightarrow Q(0.6)$$

$$= [5(0.6)^3 + 15(0.6) + 10 \quad -15(0.6)^2 + 15(0.6) + 10]$$

$$= [1.08 + 9 + 10 \quad -5.4 + 9 + 10]$$

$$= [20.08 \quad 13.6]$$

$$t = 0.8 \Rightarrow Q(0.8)$$

$$= [5(0.8)^3 + 15(0.8) + 10 \quad -15(0.8)^2 + 15(0.8) + 10]$$

$$= [2.56 + 12 + 10 \quad -9.6 + 12 + 10]$$

$$= [24.56 \quad 9.4]$$

$$t = 1.0 \Rightarrow Q(1.0)$$

$$= [5(1)^3 + 15(1) + 10 \quad -15(1)^2 + 15(1) + 10]$$

$$= [5 + 15 + 10 \quad -15 + 15 + 10]$$

$$= [30 \quad 10]$$

Value of t	Point on curve
t = 0.0	[10 10]
t = 0.2	[13.04 12.40]
t = 0.4	[16.32 13.60]
t = 0.6	[20.08 13.60]
t = 0.8	[24.56 9.40]
t = 1.0	[30 10]

Techknowledge Publications

10.1.5 B-Spline Curve

Q. Explain B-splines for curve generation.

SPPU - May 15, May 16, 2 Marks

Q. Write a short note on the B-spline curve.

SPPU - May 17, May 18, 5 Marks

Q. Explain blending function for B-spline curve.

SPPU - Dec. 17, May 19, 4 Marks

- In the Bezier curve, the degree of the polynomial is always one less than the number of control points. So we cannot have more than four points for cubic Bezier curves.
- Furthermore, change in the single control point has a global effect on the Bezier curve. B-Spline curves are the most widely used class of curves for approximating the shape due to its following properties :
  - Degree of a polynomial is independent of a number of control points (which is not true in case of Bezier curve).

- They have local control over the curve and surfaces.
- However, derivation and generation of B-spline are more complex than Bezier curves.
- The B-spline curve with (n + 1) control points is expressed in the form of blending function as,

$$P(t) = \sum_{i=0}^{n} p_k \cdot B_{i,d}(t),\ t_{min} \le t \le t_{max}$$

$$\text{and } 2 \le d \le n+1$$

– For Bezier curve, the range of t was always between 0 and 1. In the B-spline curve, t depends on types of B-spline.

– B-Spline bending functions $B_{i,d}$ are polynomials with degree d – 1. Degree d can take any integer value between 2 and number of control points.

– We can achieve local control by defining the blending function on subinterval of the total range of parameter t.

– Blending functions of Bezier curve are described by Bernstein polynomials, whereas blending functions for B-spline curve are defined using Cox-deBoor recursive formula as shown below :

$$B_{i,1}(t) = \begin{cases} 1, & \text{if } t_i \le t < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,d}(t) = \frac{t - t_i}{t_{i+d-1} - t_i} B_{i,d-1}(t) + \frac{t_{i+d} - t}{t_{i+d} - t_{i+1}} B_{i+1,d-1}(t)$$

– Blending functions are defined over d sub-intervals of the total range of t. So change in one control point can affect maximum d piecewise small curves, it does not have a global effect on the curve.

– The selected set of subinterval t_i is called **know vector**. We can choose any value for t_i in monotonically increasing order, i.e. $t_i \le t_{i+1}$. However, the value of t_{min} and t_{max} is controlled by a number of control points, parameter d and how we set up a knot vector.

Properties of B-Spline Curve

– Degree of polynomial does not depend on the number of control points.

– B-Spline curves do not interpolate any of the control points.

– The generated polynomial has degree d – 1 with inherent C^{d-2}parametric continuity.

– The curve has n + 1 blending function if it is described with n + 1 control points

– Size of knot vector is n + d + 1.

– The entire range of parameter t is divided into (n + d) subintervals.

– If the knot vector is defined as $\{t_0, t_1, t_2, ..., t_{n+d}\}$, B-spline curve spans only in the interval t_{d-1} to t_{n-1}.

Techknowledge Publications

- Each blending function $B_{i,d}$ spans over d subinterval of the entire range of t, starting from knot value t_i.
- B-Spline allows local control.
- C^2 continuity is inherent in B-Spline.
- Each B-spline curve segment is affected by d control points.
- Change in one control point influence shape of at most d curve sections.
- B-Spline curve satisfies the property of convex hull, so they are tightly bound within the control point convex hull boundary.
- For any value of t in the range t_{d-1} to t_{n+1}, the sum of all blending function is 1, i.e.,

$$\sum_{i=0}^{n} B_{i,d}(t) = 1$$

According to the specification of knot vector, B-spline curves are classified into three categories :

1. Uniform B-Splines
2. Open uniform B-splines
3. Non-uniform B-Splines

1. Uniform Periodic B-Spline

- In uniform B-Splines, knot values in knot vector are equispaced, i.e. the difference between two consecutive knot values is constant. For example {– 3, – 2, – 1, 0, 1, 2, 3, 4} is uniform knot vector. Knot values are often normalized such that they fall in range 0 to 1, for example {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6}.
- A more convenient way is to represent the knot vector with integer knot values, starting from zero, i.e. {0, 1, 2, 3, 4, 5, 6, 7}.
- Blending functions of uniform B-spline are periodic and successive blending function is translated/shifted version of the previous one.

$$B_{i,d}(t) = B_{i+1,d}(t + \Delta t)$$
$$= B_{i+2d}, (t + 2\Delta t) = \ldots\ldots$$

- Where Δt is the difference between two successive knot values. Blending functions for uniform B-spline are shown in Fig. 10.1.6.

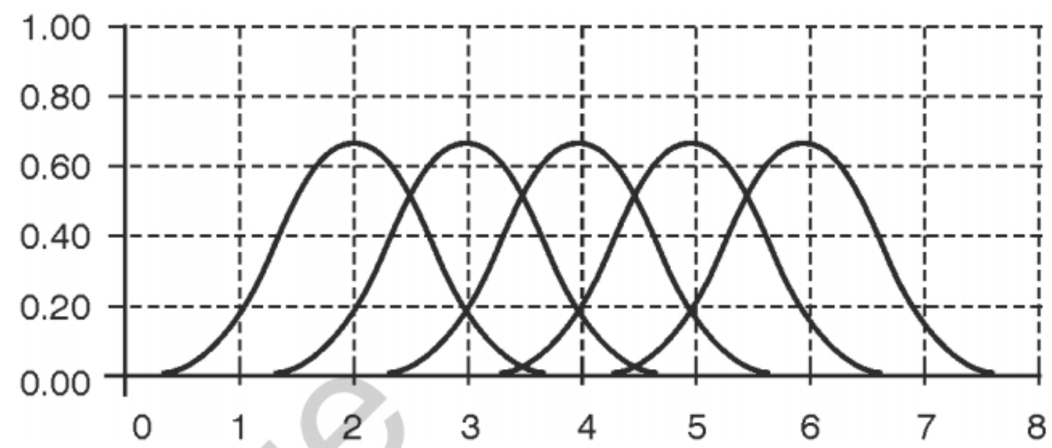


Fig. 10.1.6 : Blending functions for uniform B Spline

2. Open Uniform B-Spline

- Open uniform B-spline is the bridge between uniform and non-uniform splines. Sometimes it is considered as a special case of uniform B-spline; sometimes it is considered as a special case of non-uniform B-Spline.
- It differs from uniform spline in the specification of a knot vector. Like uniform splines, knot vector in open uniform also has uniform spacing between successive knot values except first and last. End knot values are repeated d times.
  - For d = 2 and n = 4, size of knot vector is n + 1 + d = 7. First and last knot value is repeated d = 2 times. So, starting with 0, knot vector is specified as, {0, 0, 1, 2, 3, 4, 4}
  - For d = 4 and n = 5, size of knot vector is n + 1 + d = 10. First and last knot value is repeated d = 4 times. So, starting with 0, knot vector is specified as,{0, 0, 0, 0, 1, 2, 3, 3, 3, 3}
- We can normalize the range of knot value between 0 and 1.
  - For d = 2 and n = 4, {0, 0, 0.25, 0.5, 0.75, 1, 1}
  - For d = 4 and n = 5, {0, 0, 0, 0, 0.33, 0.66, 1, 1, 1, 1}

Technknowledge Publications

- We can generate integer valued knot vector for given n and d using following formula :

$$t_i = \begin{cases} 0, & \text{for } 0 \le i < d \\ i - d + 2, & \text{for } d \le i \le n \\ n - d + 2, & \text{for } i > n \end{cases}$$

- The first knot has value 0 and last has value n – d + 2.
- Characteristics of the open B-spline curve are very much similar to the Bezier curve. Open B-spline reduces to Bezier curve when the degree of the polynomial is one less than the control point, i.e. d = n + 1. For such cubic B-spline, d = 4, n + d = 4, n + 1 = 4, so size of knot vector is 8. In this case, knot vector would be {0, 0, 0, 0, 1, 1, 1, 1}.
- Blending functions for open uniform B-spline with n = 4 and d = 3 are shown in Fig. 10.1.7.

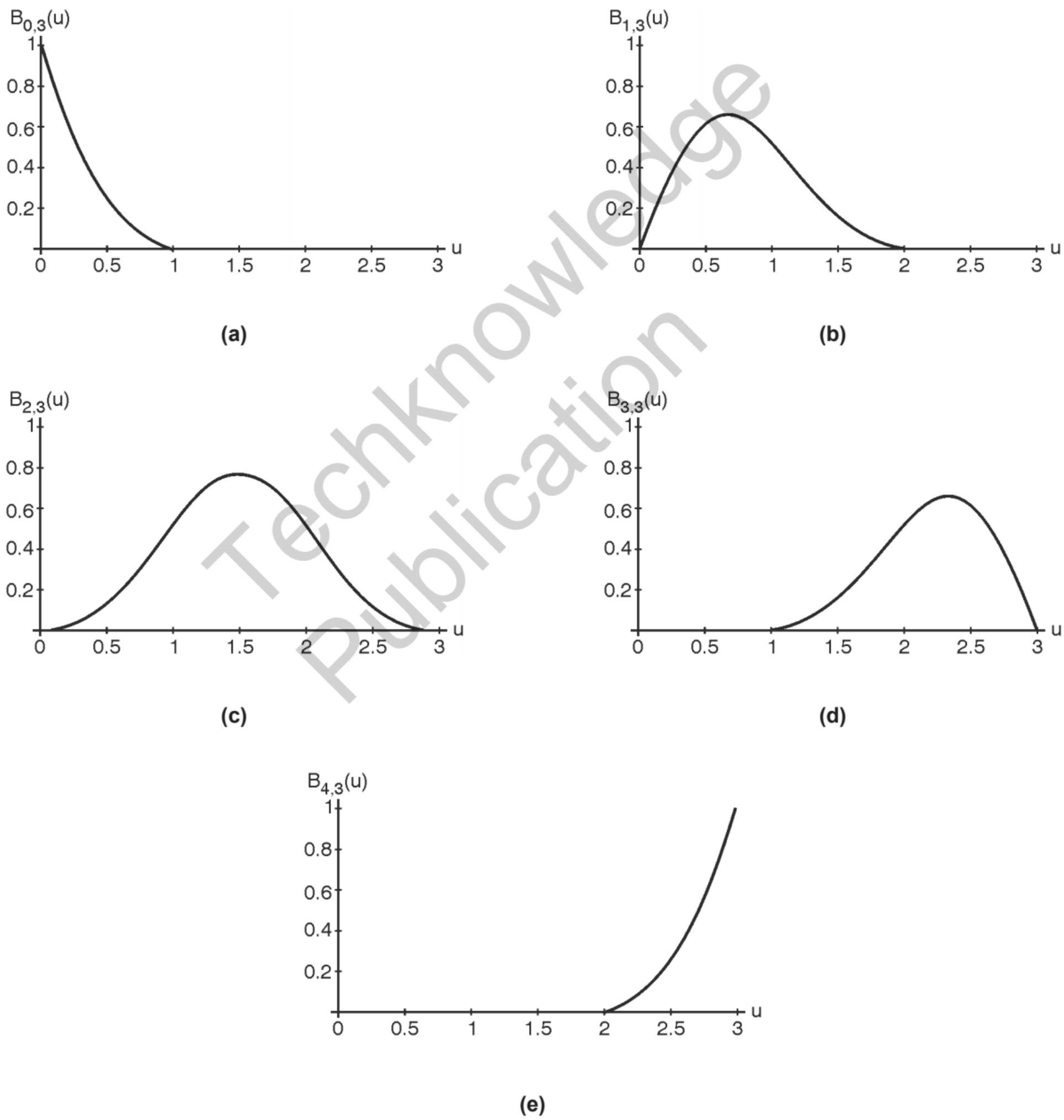


Fig. 10.1.7 : Blending functions for open uniform curve

TechKnowledge Publications

3. Non-uniform B-spline

- Knot vector for this class of polynomial is least restrictive. Knot vector can repeat the knot values; spacing between successive knot values can be different. The only constraint is that knot value must be in monotonically increasing order.
- Examples are,

  {0, 1, 2, 3, 4, 5}

  {0, 1, 1, 2, 3, 4, 4, 5}

  {0, 0.1, 0.4, 0.4, 0.7, 0.9, 1, 1}
- Such curves are most flexible in controlling the shape. Due to the unequal spacing of knot value, we get blending functions with different shape at different intervals. By repeating the knot values, we can disturb the shape of the curve and even we can introduce a discontinuity in the curve. For each repeat, discontinuity of curve reduces by one. Derivation of blending functions of non-uniform B-spline is the same as uniform and open uniform curves.
- With the help of n + 1 control points, we can use Cox-deBoor recursive formula to find blending function with specified degree and knot vector. Matrix representation can be used to avoid recursive computation of Cox-deBoor equation.
- Blending function for non-uniform B-Spline is shown in Fig. 10.1.8.

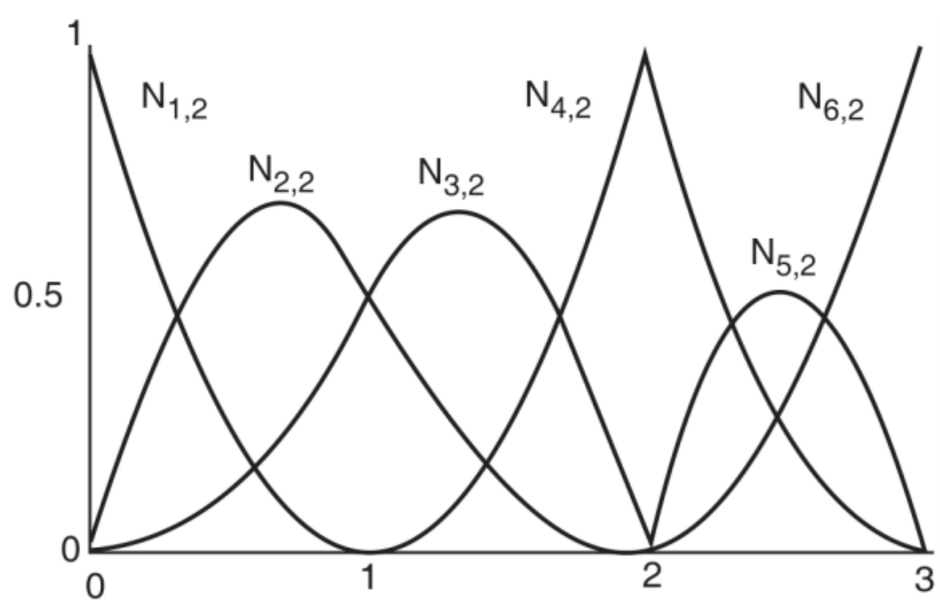


Fig. 10.1.8 : Blending functions for non-uniform curve

10.1.6 Bezier Vs. B-Spline Curve

Q. Differentiate between Bezier curve and B-spline curve. **SPPU - Dec. 19, 6 Marks**

Sr. No.	Bezier Curve Generation	B-Spline Curve Generation
1.	Passes from the first and last control point.	Does not interpolate any control point.
2.	Does not have local control on a curve.	It has local control.
3.	Changing one control point affects the position of all points on the curve.	Changing one control point affects only a few points on the curve.
4.	Degree of Bezier curve is determined by the number of control points.	Degree of the B-Spline curve is independent of the number of control points.
5.	Need special treatment to join two Bezier curves with c_2 continuity.	Continuity is inherent in the B-Spline curve.
6.	Less flexible.	More flexible.
7.	During piecewise approximation, adds 3 control point to generate a new curve segment.	During piecewise approximation, adds only 1 control point to generate a new curve segment.

10.2 Fractals

10.2.1 Introduction

Q. Explain the fractal line with suitable example. **SPPU - Dec. 15, May 16, May 17, May 18, Dec. 18, 3 Marks**

Q. What is fractal? **SPPU - Dec. 19, 3 Marks**

Tech Knowledge Publications

Fractals are infinitely complex patterns that are self similar across different scales. Natural objects are often fractals, for example tree, mountain, clouds, hurricanes etc.

Fractal objects possess the following two properties

- Infinite detail at every point.
- Self-similarity between object parts.
- Natural objects have infinite details; however, we should design a process which produces finite detail because on the computer we cannot display infinite detail.
- After certain levels of zoom in, Euclidean shapes lead to smooth drawing. While in the fractal object, if we keep zoom in, we continue to see the same detail as it appears in the original object. Fig. 10.2.1 describes the discussed concept.

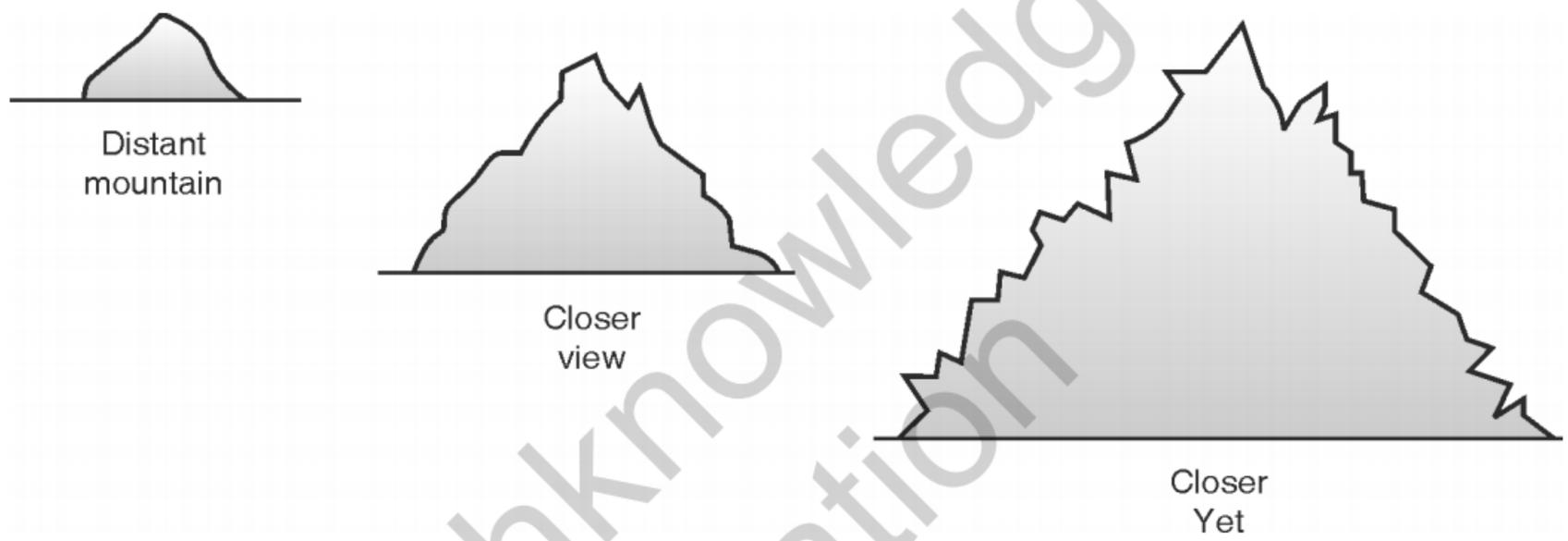


Fig. 10.2.1 : View of the mountain at different zoom level

- Zoom in effect on the monitor is achieved by selecting a smaller and smaller window with the same viewport. Less and less detail is mapped to the same size as the viewport.

10.2.2 Classification

Q. Explain characteristics and classification of fractals.

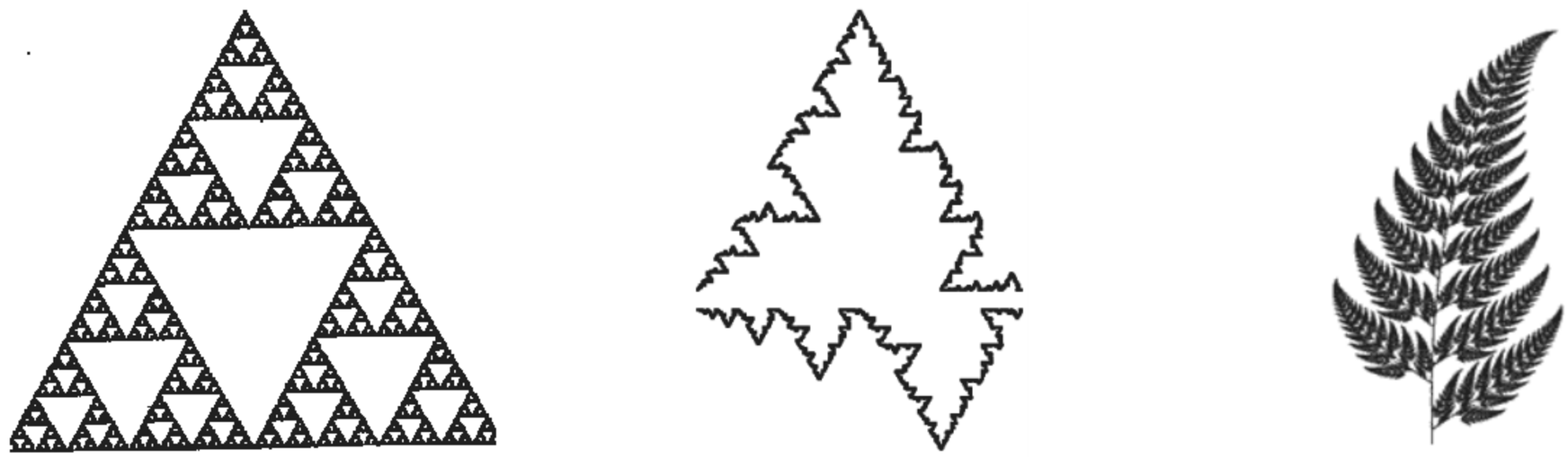
SPPU - Dec. 19, 3 Marks

Fractals can be classified as follow :

1. **Self-similar fractals :** Such fractal exhibits the self-similarity in shape. They have components which are a scaled-down version of the object itself. We can construct the object by applying different scaling factors to all parts or by using the same scaling factor for all. If we use a random scaling factor to generate the subparts, a fractal is called **statistically self-similar**. Statistically self-similar fractals are used to construct shapes like tree, shrubs, plants etc.
2. **Self-affine fractals :** Such fractals use different scaling factors for all three directions. We can add random variation to get statistically self-affine fractals. This class of fractals is best suited to model terrain, water, clouds etc.
3. **Invariant fractals :** Nonlinear transformation is applied to construct such fractals. A self-squaring function in complex space produces self-squaring fractals like Mandelbrot. Self-inverse fractals are generated using the inversion procedure.

Tech Knowledge Publications

Example of each class is depicted in following Fig. 10.2.2 :

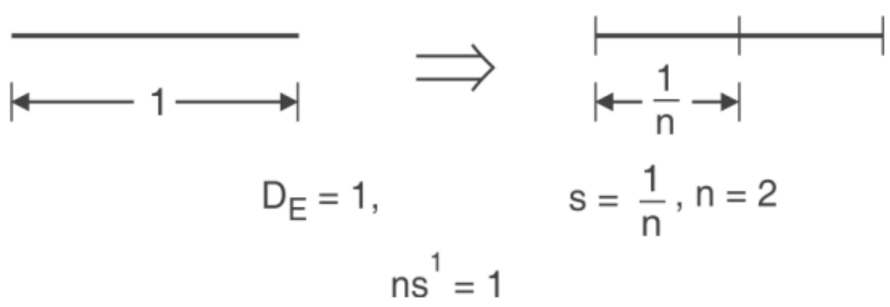


(a) Self-similar fractals (b) Self-affine fractals (c) Invariant fractal

Fig. 10.2.2 : Types of fractals

10.2.3 Fractal Dimension

- Amount of variation in object detail is described by a number called **fractal dimension**. Fractal dimension can be any real number.
- Detail variation in a fractal is controlled by the fractal dimension number D. We can consider D as a measure of the roughness of the object.
- Objects with jaggy boundary have larger D value. We can write iterative procedure controlling detail by specifying D. However, it is difficult to estimate the value of D for real objects.
- Self-similar fractals are obtained by recursively applying the same scaling factor to Euclidean shapes.
- Fig. 10.2.3 shows the relationship between scaling factor and a number of parts for unit length line, unit square and unit cube with s = ½. With s = ½, the line is divided into two equal segments, the square is divided into four equal parts of equal area and cube is divided into eight subcubes of equal volume.



(a)

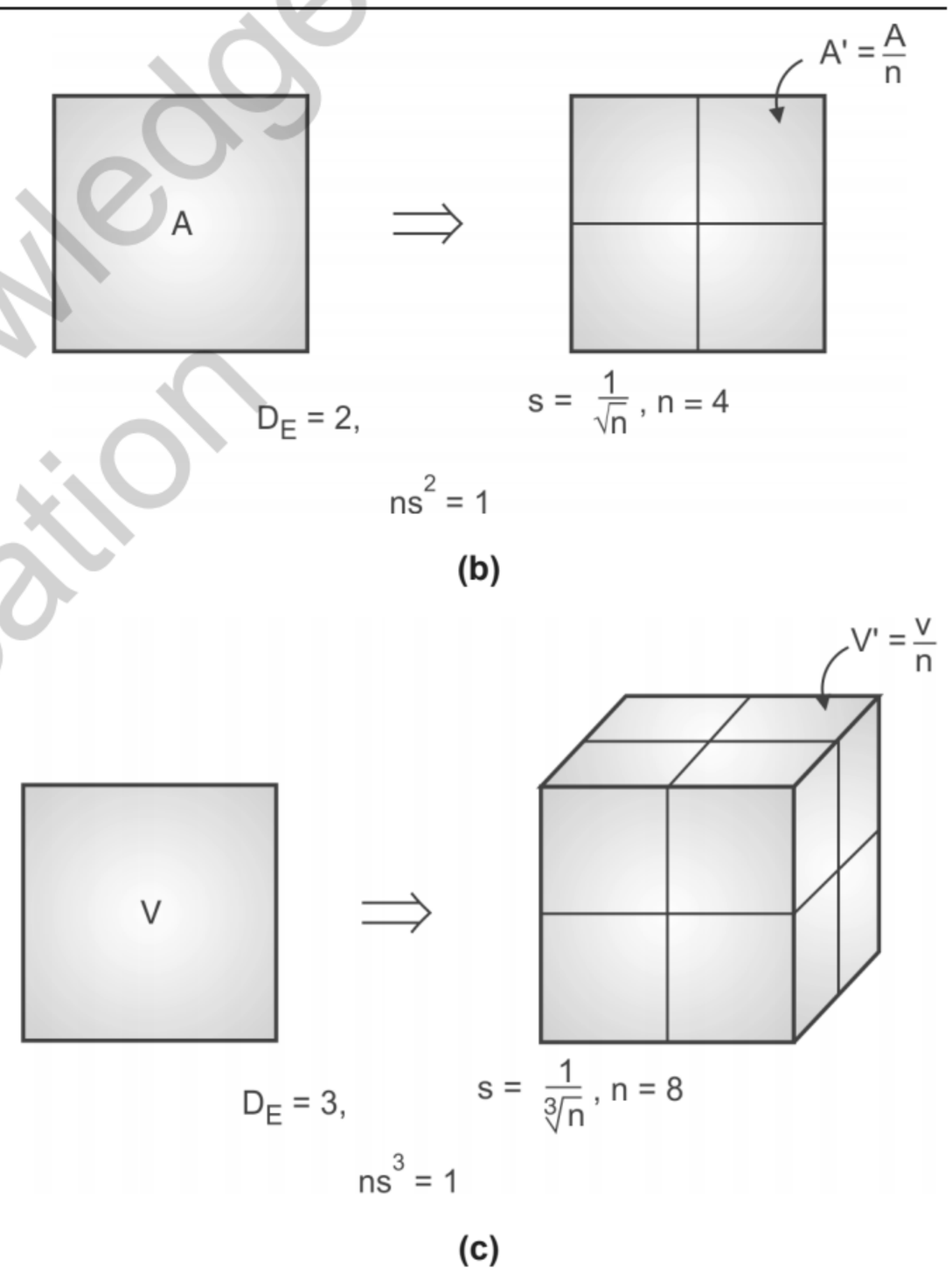


Fig. 10.2.3 : Subdivision of object with scaling factor s = ½ and Euclidean dimensions ((a) D_E = 1, (b) D_E = 2 and (c) D_E = 3)

For these objects, the relationship between a number of subparts and the scaling factor is $n \cdot s^{DE} = 1$. For self-similar objects,

$$n \cdot s^D = 1$$

Technical Knowledge Publications

Solving the equation,

$$D = \frac{\ln n}{\ln (1/s)}$$

– D is called the fractal similarity dimension. For a self-similar object with different scaling factor for a different part, self-similarity dimension is obtained as,

$$\sum_{i=1}^{n} S_i^D = 1$$

Where s_i is the scaling factor for i^{th} subpart of the object.

– For complex shapes like curves and surfaces, it is difficult to estimate the characteristics of subparts of the shape. The large shape is approximated by subdivides the shape into smaller simpler shapes, that is called **topological covering**. For example, smooth curves can be approximated using a very small line segment. The surface can be approximated using small polygons like triangles.

– Covering methods are used to determine object properties like length, area, the volume of an object by summing these properties smaller objects – forming the parent object. Covering methods also help in determining fractal dimensions (D).

10.2.4 Fractal Generation

– Fractals are generated by applying transformation function repeatedly to the points in the region of interest. If P(x, y, z) is the initial point, each iteration of transformation function generates detail as,

$$P_1 = F(P), P_2 = F(P_1), P_3 = F(P_2), \ldots$$

– Transformation function can be applied to a set of points of selected base primitive. This primitive can be anything like line, arc, curve, surface etc. We can specify a deterministic or non-deterministic (random) process in each iteration. The transformation function can be either simple geometry transformation operations like translation, rotation, scaling, reflection etc, or it can be specified using non-linear transformation functions.

– The objects we create and display on the screen is generated using finite transformation operation. So they have finite dimensions. We cannot display the details which are smaller than the pixel. Details appearing on view depend on a number of transformations we applied. If we increase the number of transformation, more and more subtle details are added to the scene.

10.2.4.1 Snowflake

– The snowflake is generated from the Koch curve. It was described by H. V. Koch in 1904.

– Koch curve begins with the straight line. It divides the line into three segments. The middle segment is replaced by two sides of an equilateral triangle with the apex pointing outside. Now we have four segments. This same process is repeated on these four segments. General procedure for generating the Koch curve can be stated as :

1. Divide a straight line into three equal parts.
2. Draw an equilateral triangle on the middle segment, pointing in an outward direction with a base of the middle segment.
3. Remove the middle line segment.

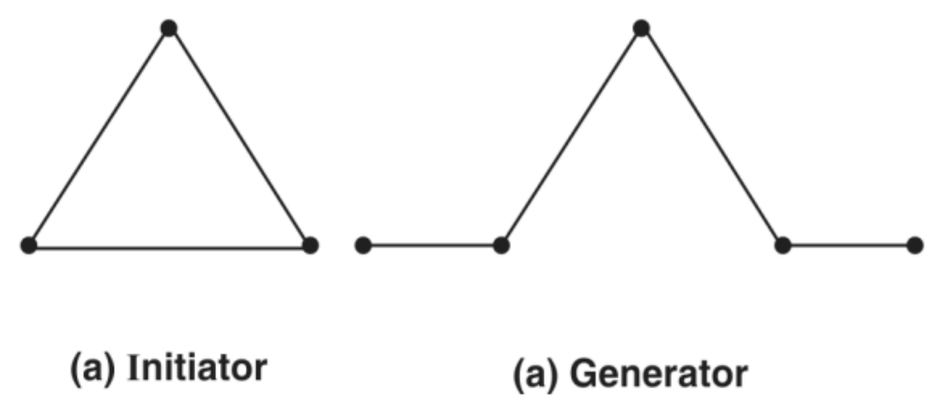


Fig. 10.2.4 : Initiator and generator for koch curve

– This procedure is repeated to achieve more and more smoothness on the curve.

TechKnowledge Publications

- To construct, deterministic, self-similar fractals, we start with geometric shapes called the initiator.
- A subpart of the initiator is then repeatedly replaced by generator pattern.
- Fig. 10.2.4 shows the initiator and generator pattern for the Koch curve.

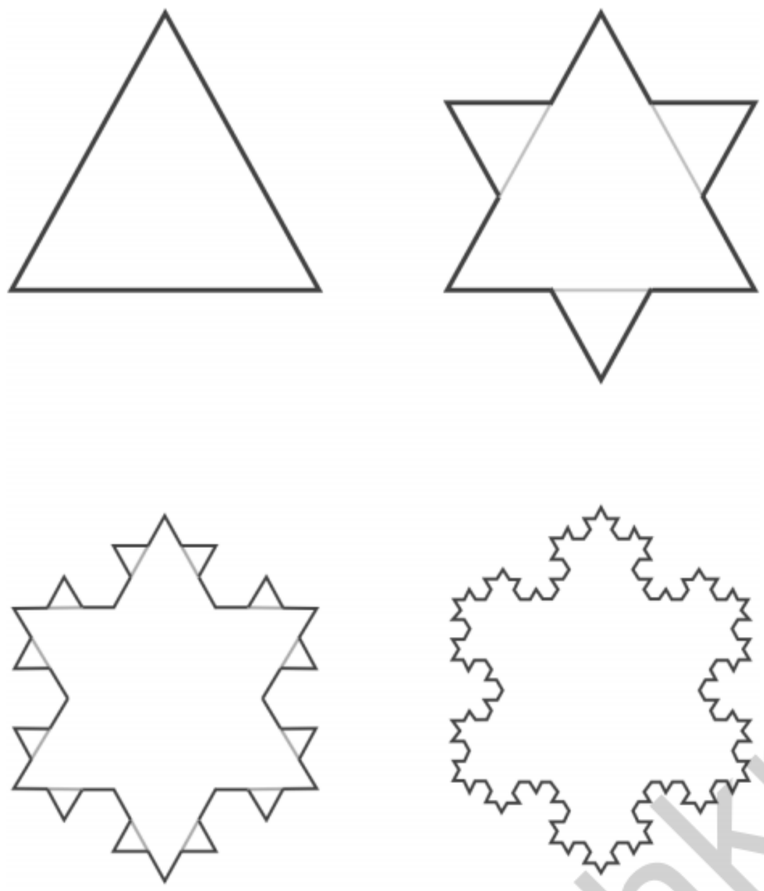


Fig. 10.2.5 : Snowflakes

10.2.4.2 Triadic Curve

Q. Explain the Koch curve with example.

SPPU - May 17, Dec. 17, May 18, May 19, 4 Marks

Q. Explain the Koch curve and its application in detail.

SPPU - Dec. 18, 4 Marks

Q. Write short note on Koch curve.

SPPU - Dec. 19, 2 Marks

The triadic curve is an extension of simple Koch curve.

Steps to generate triadic curve are stated here :

Step 1 : Divide a straight line into three equal parts.

Step 2 : Place the equilateral triangle in the middle part and remove the baseline of the middle part. Now the curve has four segments.

Step 3 : Divide each segment into three equal parts and repeat step 2.

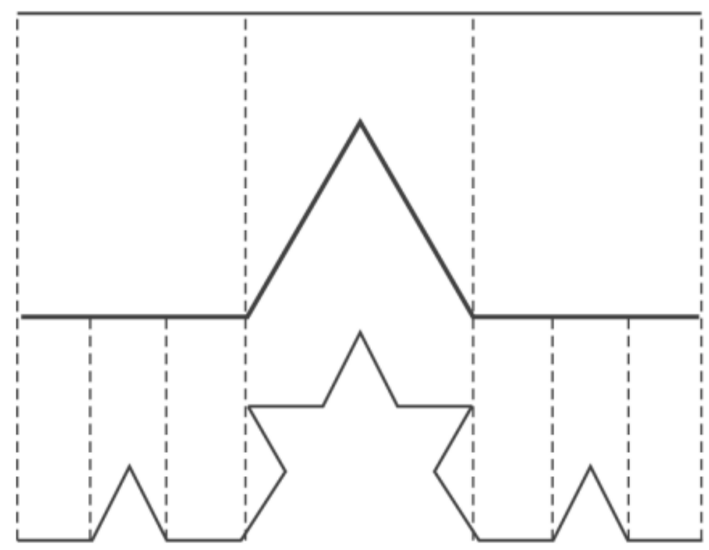


Fig. 10.2.6 : Generation of triadic curve : step 1 (top), step 2 (middle), step 3(bottom)

- This repetition goes for infinite times. If we zoom any section of the curve, the basic triangle structure can be seen.
- Length of each segment on successive division is shown in Fig. 10.2.7.

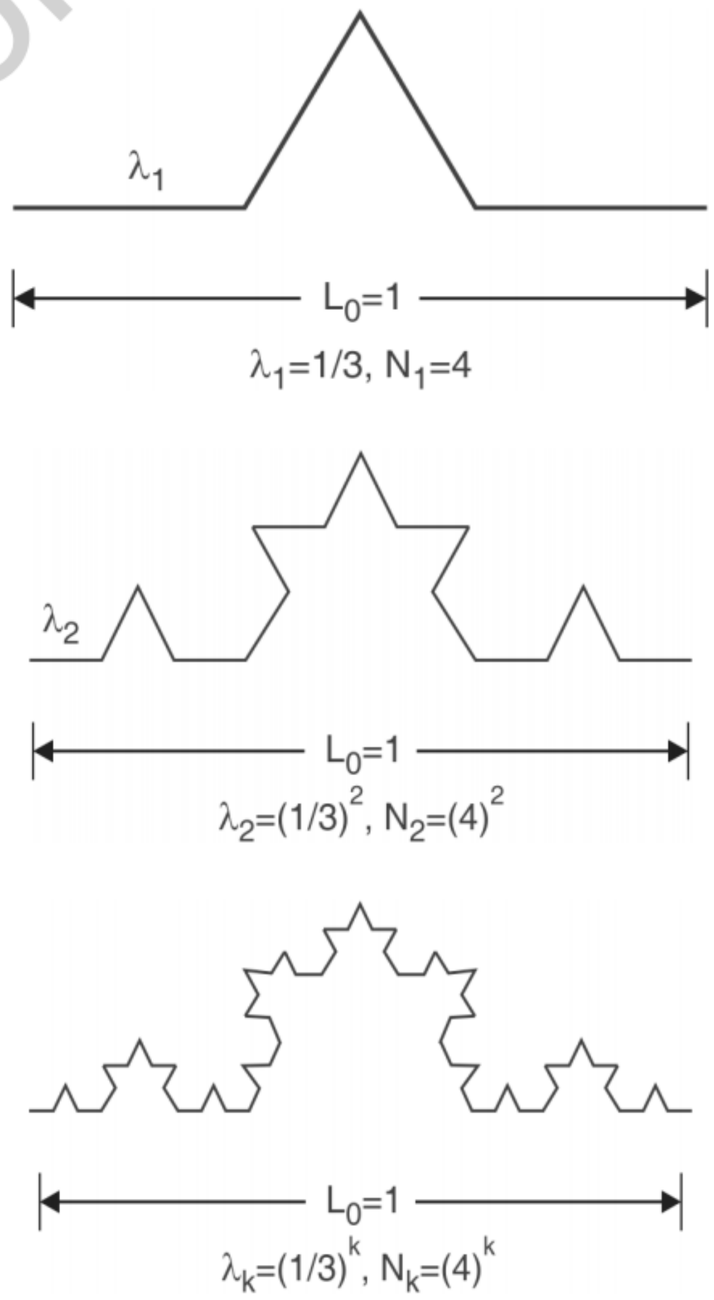


Fig. 10.2.7 : Koch curve generation

TechKnowledge Publications

10.2.4.3 Hilbert Curve

Q.	Explain Hilbert's curve with an example.	**SPPU - May 15, May 17, Dec. 17, Dec. 18, May 19, 6 Marks**
Q.	Explain the Hilbert curve and its application in detail.	**SPPU - May 18, 5 Marks**

It was introduced by the German mathematician David Hilbert. This curve is also known as the **Hilbert space-filling curve**. The basic entity of this curve is U-shape as shown in Fig. 10.2.8(a).

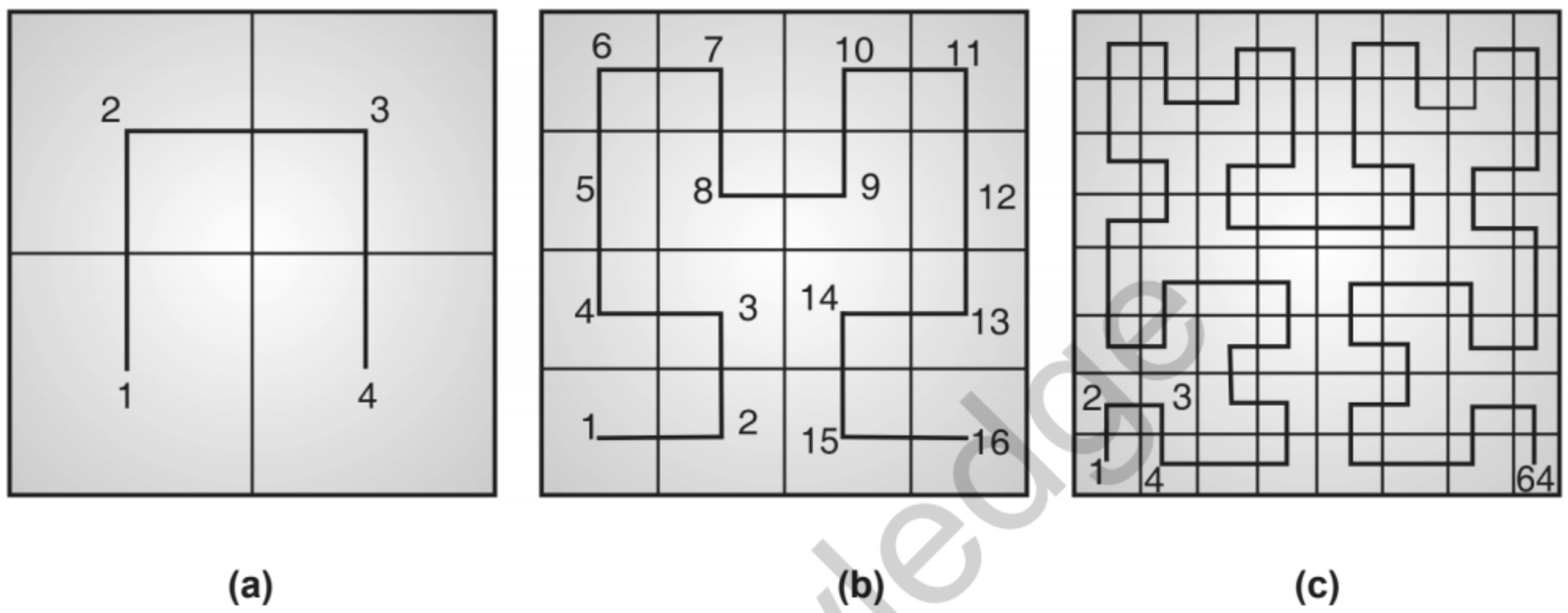


Fig. 10.2.8 : (a) First approximation, (b) Second approximation and (c) Third approximation

- Generation of Hilbert curve is simple. It visits the center of every square grid having size 2×2, 4×4, 8×8, 16×16, or any other size of $2^n \times 2^n$.
- The curve generation starts with a square. In a first approximation, the square is divided into 4 quadrants and the curve joins the centre of all quadrants by a straight line.
- In the second approximation, each quadrant is further subdivided into 2×2 grids, ending in 16 squares. Curve visits the centre of each small square in each quadrant before moving to the next quadrant (Refer Fig. 10.2.8(b))
- In third approximation, each of the 16 squares is further subdivided into 2×2 grids, ending in 64 squares. Curve visits the center of each small square before moving to higher-level quadrant (Refer Fig. 10.2.8(c))
- On applying this process continuously,
  - The curve never crosses itself
  - Curve gets closer to a square containing it
  - With each subdivision, the length of the curve increases four times.
- There is no limit on depth and hence there is no limit on curve length.
- At each subdivision, the scale of the square is reduced by factor 2, and length is increased by factor 4.
- Ultimately, the Hilbert curve is a line with topological dimension $D_t = 1$, but as the curve gets scaled and twisted at every next approximation, it fills the square.
- The fractal dimension of the curve is given as,

$$N = 2^D$$

$$4 = 2^D$$

$$\text{So, } D = 2.$$

Thus, this curve has topological dimension 1 and fraction dimension 2.

TechKnowledge Publications

10.2.5 Applications

Fractals are used to model the natural shapes. Some of the prominent application areas of fractals are listed here :

- Modelling natural structures like Geographic terrain, mountain, plant structure, clouds, vegetables etc.
- Study of the chaotic phenomenon
- Fractal art
- Space research
- Study of convergence of iterative processes
- Engineering and architecture
- Medical science
- Chemical processes
- Medical diagnostic images
- Fluid mechanics
- Image compression
- Telecommunication

Review Questions

Q. 1 What is the Bezier curve? Give the blending functions for Bezier curve.

Q. 2 Enlist the properties of Bezier curve.

Q. 3 State a few applications of Bezier curve.

Q. 4 Write a short note on the Cubic Bezier curve.

Q. 5 Give the matrix representation of the Bezier curve.

Q. 6 What are the limitations of the Bezier curve?

Q. 7 How to generate a Bezier curve using subdivision method.

Q. 8 Write a note on the B-spline curve.

Q. 9 What are the properties of the B-Spline curve?

Q. 10 Write a short note on uniform periodic B-spline curve.

Q. 11 Write a short note on open uniform B-spline curve.

Q. 12 Write a short note on non-uniform B-spline curve.

Q. 13 Differentiate the Bezier curve vs. the B-Spline curve.

Q. 14 What do you mean by fractals?

Q. 15 Write a short note on fractal dimensions.

Q. 16 Write a short note on Hilbert's curve.

❑❑❑

Technknowledge Publication

CHAPTER 11

UNIT VI

Segment, Animation and Gaming

Syllabus

Segment : Introduction, Segment table, Segment creation, closing, deleting and renaming, Visibility.

Animation : Introduction, Conventional and computer based animation, Design of animation sequences, Animation languages, Key- frame, Morphing, Motion specification.

Gaming : Introduction, Gaming platform (NVIDIA, i8060), Advances in Gaming.

11.1 Segment

- For a human, images are not just a collection of pixels, but it is a meaningful arrangement of the objects in the environment.
- A human can interpret images very easily. Images may be natural images, synthetic images, paintings etc.
- In a computer system, image information is stored in the display file.
- However, the existing structure of the display file is not suitable for viewing the image. So the image is divided into small substructures called segments. These segments are managed using a segment table.

11.1.1 Introduction

Q. What is a segment?

SPPU - Dec. 15, May 16, Dec. 17, 2 Marks

- Part of the display file is called a segment. Typically, each segment corresponds to one concept in the scene to be displayed.
- Attributes like *visibility* and *image transformation* are set for each segment.
- Set *visibility* attribute to ON, if we want to display the object, set it OFF otherwise.
- To perform the geometric transformations on the object, enable *image transformation* attribute and set appropriate parameters.
- Various geometric transformations like translation, rotation, scaling, reflection etc. can be applied to the objects. Visibility of an object may change after applying image transformations.
- There are two types of segments :
  1. **Posted segment :** When visibility attribute of the segment is set to 1, it is called a *posted segment*. Such a segment is added to the active segment list.
  2. **Unposted segment :** When visibility attribute of the segment is set to 0, it is called an *unposted segment*. Such a segment is not included in the active segment list.
- Segment table can be implemented using an array or linked list.
- In the case of an array, size is the upper bound for the number of segments that can be included in the segment table. Linked list implementation has no such restriction as it is a dynamic data structure.
- Deletion of the item in the array requires shifting of all preceding cells up by one position.
- In case of a linked list, deletion is performed by updating the links only.
- The downside of linked list implementation is the storage requirements.

Technknowledge Publication

Advantages

- Display file is divided into segments to represent the different objects.
- Managing and manipulating objects with segments is much easier than managing and manipulating the entire scene at once.
- Visibility of each object can be controlled.
- The flow of animation can be controlled using various image transformation operations on the individual segment.

Disadvantages

- Extra overhead is created to maintain the segment table.
- Display file and segment table work together to paint the scene.
- An optimal number of segments needs to be created for efficiency.
- Content of the display file is no more independent now.

11.1.2 Segment Table

Q. Explain the Segment Table with an example.
SPPU - Dec. 15, 4 Marks

Q. What is Segment Table?
SPPU - May 16, 4 Marks

Q. Explain the term with an example - Segment table.
SPPU - Dec. 18, 2 Marks

Q. Draw a sample segment table.
SPPU - May 19, 2 Marks

- As discussed earlier, attributes like visibility and image transformations are set for each segment.
- To identify each segment unambiguously, each segment is assigned a unique *segment name*. Using the segment name, we can select a particular segment and can set the appropriate attribute.
- We must know the *segment start* and *segment size* to know the boundary of the segment. When a segment is selected to display, the instructions are read out from the segment start address.
- Segment table and display file need to be work in corresponds. To do so, a table called *segment table* is designed. Structure of the segment table is shown in Fig. 11.1.1.

Segment Name	Segment Start	Segment Size	Visibility	Scale x	Scale y	–
1	–	–	ON	–	–	–
2	–	–	OFF	–	–	–
3	–	–				–
4						–
–						–

Fig. 11.1.1 : Structure of the segment table

- Array or linked list is used to hold the information of the table. Choice of data structure is programmer dependent. Both have their merits and demerits.
- Segment names are mapped to the segment number, which makes very convenient to access the entries in array representation.
- For example, we can set the visibility of any segment 'k' by executing the command,

  Visibility [segment name] = ON / OFF

Tech Knowledge Publications

- Segment name acts as an index to the segment table.
- Display file stores the picture information as a series of commands. These commands are interpreted by display file interpreter and appropriate actions are taken.
- The process is depicted in Fig. 11.1.2.

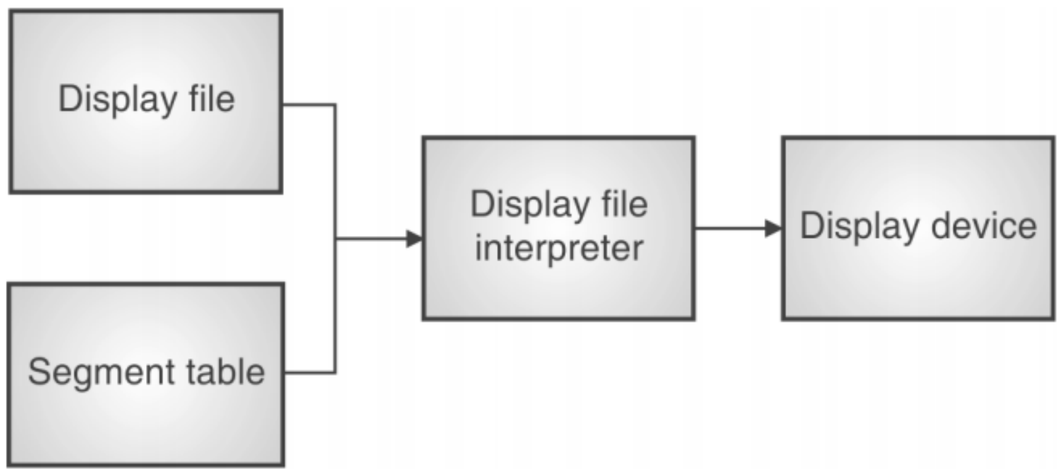


Fig. 11.1.2 : Correspondence between display file and segment table

- If the visibility of the current segment is set ON, display file interpreter interprets the commands based on the respective *segment start* and *segment size* attributes in the segment table.
- If the visibility of the segment is set OFF, display file interpreter won't interpret the segment.

11.1.3 Operations on Segment

Q. Describe various operations carried out on the segment. **SPPU - May 15, Dec. 17, 6 Marks**

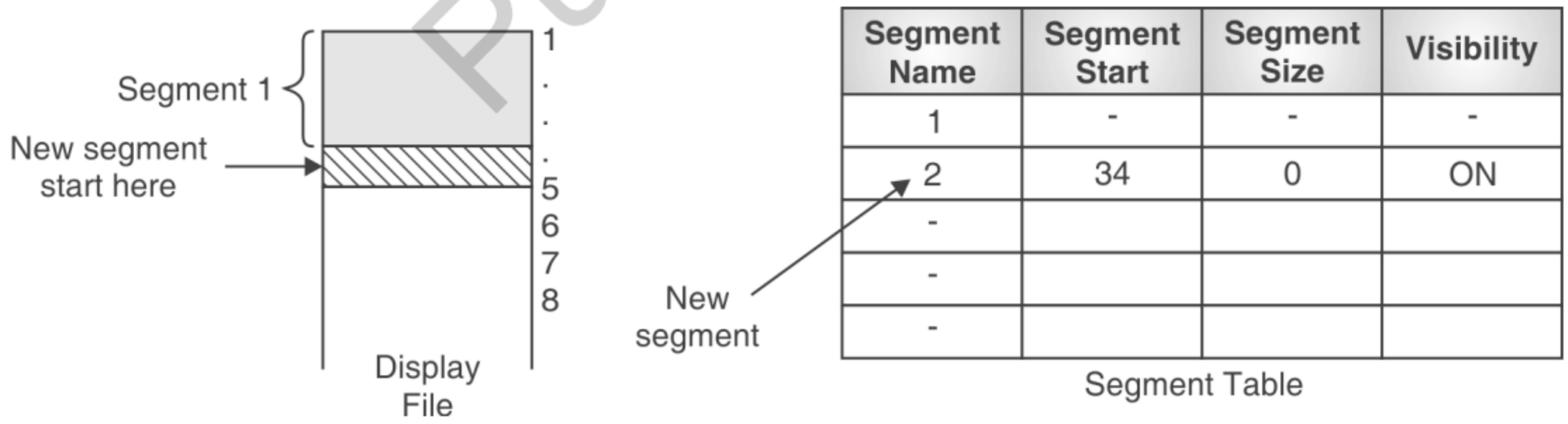
Q. Explain transformation operation on the segment. **SPPU - May 16, 2 Marks**

Creation, closing, deletion and renaming are the most frequent operations performed on the segment table.

11.1.3.1 Segment Creation

Q. Write algorithm to create a segment. **SPPU - Dec. 19, 3 Marks**

- Before we create any segment, we need to check if any other segment is open or not.
- If two segments are opened simultaneously, it is not possible to determine which segment should be drawn.
- If no segment is open, then open the segment with a valid name. Newly written commands now belong to this segment. Respective entry is made into the segment table.
- The first instruction of the segment is placed at the end of the display file.
- Size of the segment is initialized to zero and appropriate attributes are set. For example, visibility is set to ON, etc.



Segment Name	Segment Start	Segment Size	Visibility
1	-	-	-
2	34	0	ON
-			
-			
-			

Fig. 11.1.3 : Segment creation

Steps for segment creation are stated below :

Algorithm

Step 1 : If any segment is open, then give the error "segment is still open" and go to step 8.

Step 2 : Read the name of the new segment.

Tech Knowledge Publications

Step 3 : If the name of the segment is not valid then show an error message: "segment name is not valid" and go to step 8.

Step 4 : If the name of the segment is already present in the segment table, then show an error message : "Segment name already exist in name list" and go to step 8.

Step 5 : Initialize the next free space in display file as the start of a new segment.

Step 6 : Set the size of the new segment to zero and set all attributes to its default values.

Step 7 : Inform that new segment is now open.

Step 8 : Stop.

11.1.3.2 Segment Closing

- Once all the instructions of the segment are entered into the display file, the segment should be closed.
- To do this, the current open segment is renamed to segment 0. Now unnamed segment is open, which needs to be deleted.
- Steps to close the segment are stated below :

Algorithm

Step 1 : If no segment is open, then give an error message: "No segment is open" and go to step 6.

Step 2 : Rename the current open segment to segment 0.

Step 3 : Delete any other unnamed segment instruction which may have already been saved. Initialize the current unnamed segment with no instruction.

Step 4 : Assign the beginning of unnamed segment in display file as first free storage area.

Step 5 : Set the size of the unnamed segment to 0.

Step 6 : Stop.

11.1.3.3 Segment Deleting

Q. Write algorithm to delete a segment.

SPPU - Dec. 19, 3 Marks

- To delete a particular segment from the display file, we should delete that segment and recover the space occupied by that segment so that it can be used for some other segment.
- Space management depends on the type of data structure used to create the display file. If the array is used, then all subsequent entries of the deleted segment are shifted up by one position.
- This will ensure that all the free space will be at the other end of the file. The process is depicted in Fig. 11.1.4.

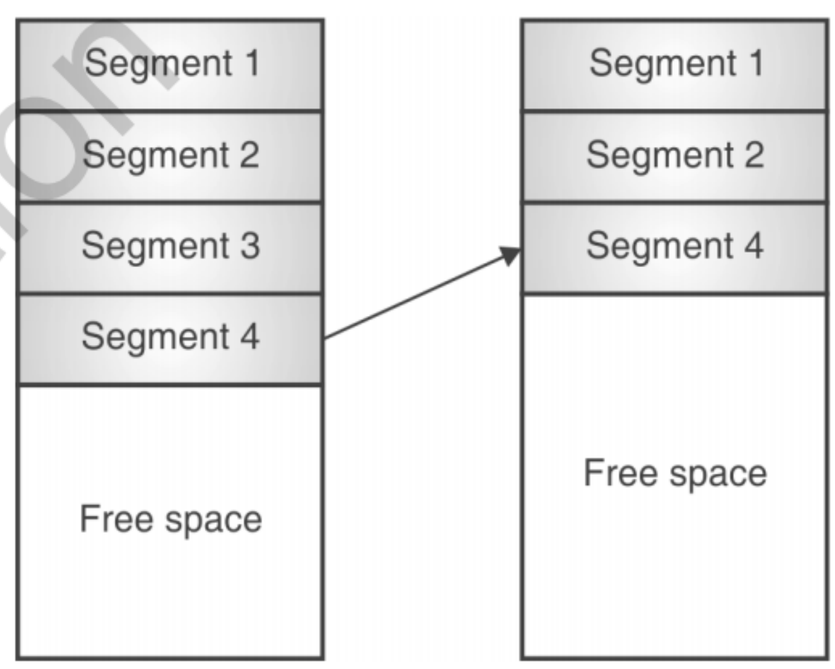


Fig. 11.1.4 : Display file- before deletion (left) and after deletion (right)

- Size of the segment to be deleted must be non-zero. The non-zero size ensures that there are some commands present in the segment.
- Zero segment size implies that there is no command present in the segment and hence there is nothing to delete.
- Next segments are shifted up by the size of the segment to be deleted. Segment 2 is deleted and the process is described in Fig. 11.1.5.

Technknowledge Publication

Segment Name	Segment Start	Segment Size	Visibility	–
1	1	7	ON	–
2	8	3	ON	–
3	11	6	ON	–
4	17	5	ON	–

(a) Before deleting segment 2

Segment Name	Segment Start	Segment Size	Visibility	–
1	1	7	ON	–
2	8	0	ON/OFF	–
3	11 – 3 = 8	6	ON	–
4	17 – 3 = 14	5	ON	–

(b) After deleting segment 2

Fig. 11.1.5 : Segment deletion

- To delete all the segments at a time, set the size of all the segment to zero and initialize the starting index of all segments to zero.
- Steps to delete the segment are described below :

Algorithm

Step 1 : Get the name of the segment to be deleted.

Step 2 : If the name is not valid then show an error message : "Segment name is not valid" and go to step 7.

Step 3 : If the segment is open, then show an error message : "Cannot delete an open segment" and go to step 7.

Step 4 : If the size of the segment is less than 0, no processing is needed, go to step 7.

Step 5 : All the segments following the segment to be deleted are shifted up by one position.

Step 6 : Recover free space by resetting the index to free space.

Step 7 : Stop.

11.1.3.4 Segment Renaming

Q. Write an algorithm to rename a segment. **SPPU - May 19, 2 Marks**

- The animation is achieved by changing the position of objects or by applying transformation operations on objects in the next frame.
- To move the position of the object in animation, we need to delete current buffer content, compute positions of objects for the next frame, create a new segment and then display it.
- This is a big amount of work. If only one buffer is used to generate the animation, picture on-screen flickers due to computational overhead.
- To overcome the problem, double buffers are used. Content of one buffer will not be deleted until another buffer is processed and filled up. This will make the scene persistent on the screen.
- Segment renaming is required in case of double buffering. During the animation, while one buffer is being displayed, the second buffer is used to alter and fill the animated scene.

Tech Knowledge Publications

– Invisible image with a temporary name is created. When the first image is deleted, replace the temporary image with the original one.

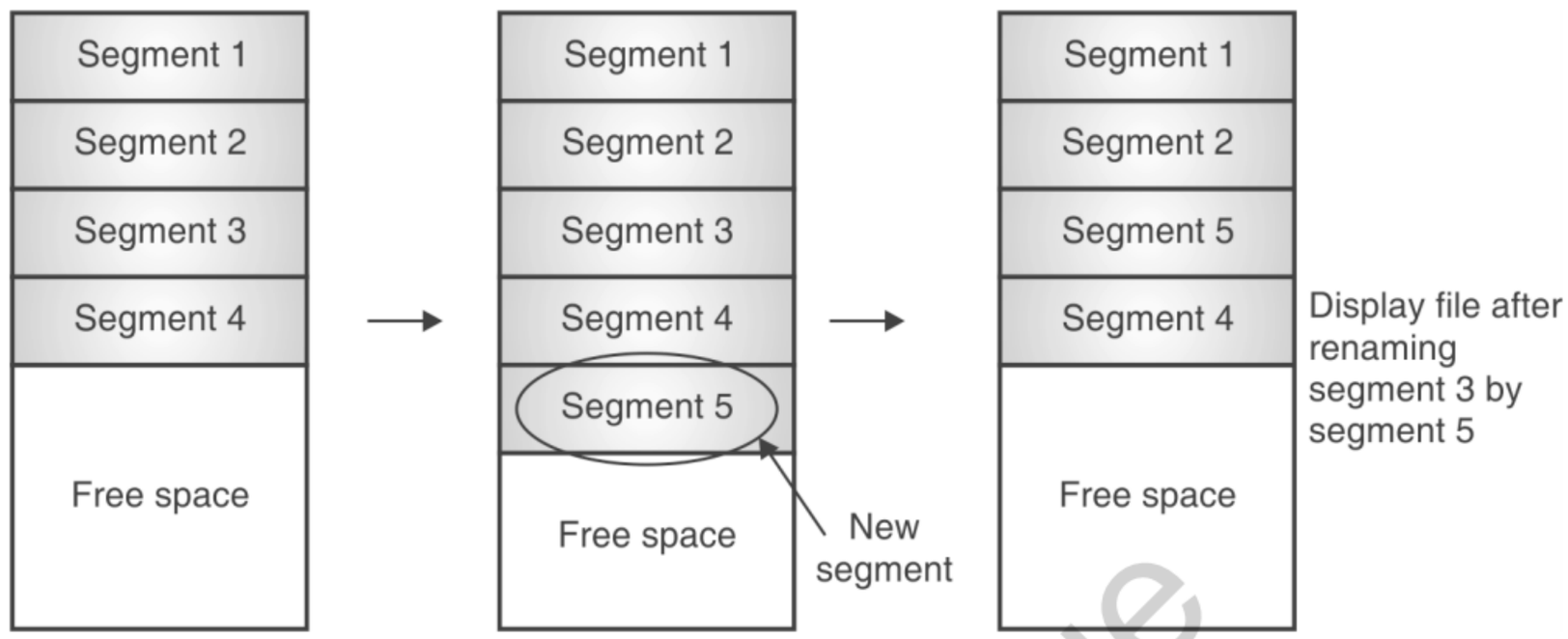


Fig. 11.1.6 : Renaming the segment

– We have to update the entries in segment table too. Segment table before the creation of a new segment, after creating a segment and after renaming, segments are shown in Fig. 11.1.7, 11.1.8 and 11.1.9.

Segment Name	Segment Start	Segment Size	Visibility	–
1	1	4	ON	–
2	5	5	ON	–
3	10	6	ON	–
4	16	7	ON	–

Fig. 11.1.7 : Segment table before creating a new segment

Segment Name	Segment Start	Segment Size	Visibility	–
1	1	4	ON	–
2	5	5	ON	–
3	10	6	ON	–
4	16	7	ON	–
5	–	–	OFF	

Fig. 11.1.8 : Segment table with both - segment 3 and segment 5 (after creating segment)

Segment Name	Segment Start	Segment Size	Visibility	–
1	1	4	ON	–
2	5	5	ON	–
3	10	**0**	**ON/OFF**	–
4	16	7	ON	–
5	**10**	**6**	**ON**	

Fig. 11.1.9 : Segment table after renaming segment 3 as segment 5

– Segment name validation and existence of the same name are checked and appropriate messages are thrown on error.

– Algorithm for segment renaming is described below.

Algorithm

Step 1 : If segment names (before and after renaming) are not valid, then give an error message: "Segment names are not valid" and go to step 6.

Step 2 : If any of the two segments are open, then given an error message: "Segments are still open" and go to step 6.

Tech Knowledge Publications

Step 3 : If the new name exists, then give an error message : "Segment name already exists" and go to step 6.

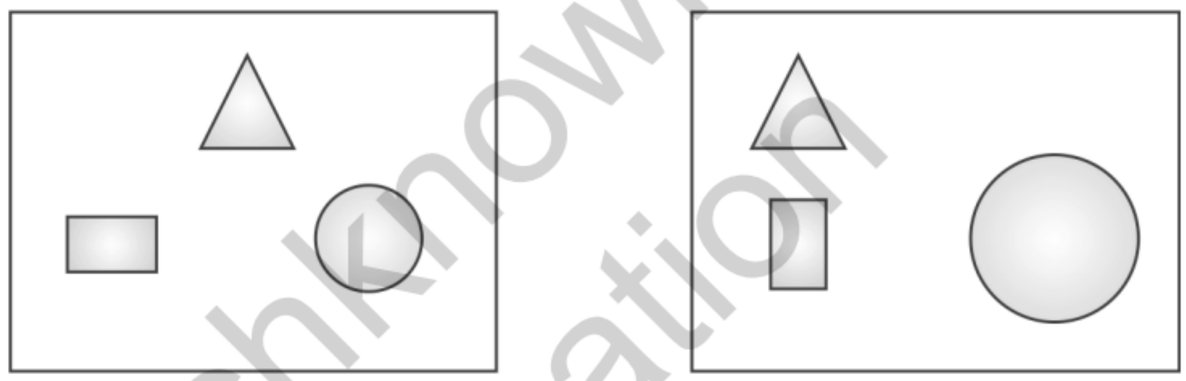
Step 4 : The old segment table entry is copied to a new position.

Step 5 : Delete the old segment.

Step 6 : Stop.

11.1.4 Image Transformation

- Transformation operations modify the object by changing its position, scale or orientation. Translation, scaling and rotation are the basic transformation operations.
- Image to be displayed on the screen is stored in the display file. So to create an animation, image transformation operations are applied to the content of the display file.
- To translate the object, rotate and scale the object, we shall specify translation vector, rotation angle and scaling parameters, respectively.
- We may be interested in transformation entire image or part of the image. Any transformation operation can be applied to an individual or set of segments.
- In Fig. 11.1.10, circle, rectangle and triangle are scaled, rotated and translated, respectively. So we need to specify scaling, rotation and translation parameters for the respective object.



(a) Scene before transformation **(b) Scene after transformation**

Fig. 11.1.10 : Scene transformation

- For image transformation, we have to add transformation parameters for each segment. Now, the structure of the segment table would be,

Segment Name	Segment Start	Segment Size	Visibility	Translate x	Translate y	Scale x	Scale y	Rotate Theta
–	–	–	–	–	–	–	–	–

Fig. 11.1.11 : Segment table for image transformation

- For each frame of animation video, the segment table is updated and appropriate transformation parameters are set.
- For static objects in the scene, transformation parameters will remain the same in subsequent frames.

TechKnowledge Publications

11.2 Animation

11.2.1 Introduction

Q. Explain in brief the basic guidelines of animation.
SPPU - Dec. 15, 6 Marks

Q. Write the basic guidelines of Animation technology.
SPPU - May 16, 4 Marks

Q. What is animation? **SPPU - May 17, 2 Marks**

Q. Explain the term with an example - Animation.
SPPU - Dec. 17, 1 Mark

- **Computer animation** refers to the time sequence of visual changes. The animation is a key concept in entertainment, education, simulation, games, scientific and engineering studies etc. Today, the scope of animation is not only limited to changing the position of an object with time, but the concept is extended to various transformation operations like scaling, rotation, variations in color, transparency, surface property, shape etc.
- The animation is achieved by displaying successive frames with minor differences. Near identical frames are displayed at a certain rate, which creates the illusion of animation in the scene.
- Entertainment and advertisement animation often transform one shape to others, like a man is converted to animal or car. Computer animation can be generated by changing camera position, orientation, movement speed, focal length, light, etc.
- Accuracy is a concern when it is to be used in study and simulation. Entertainment and advertisement applications are more concern about visual and lighting effects.

11.2.2 Conventional and Computer-based Animation

- The animation is widely used in movies, advertisement, games etc. Animation gives life to the objects present in the scene through various transformation operations. Animation can be computer-assisted or computer-generated.
- The idea behind the animation is to fool the human eye by displaying images at a certain speed so that they are interpreted as video of moving objects.
- Animation techniques can be classified as conventional animation and computer-based animation.

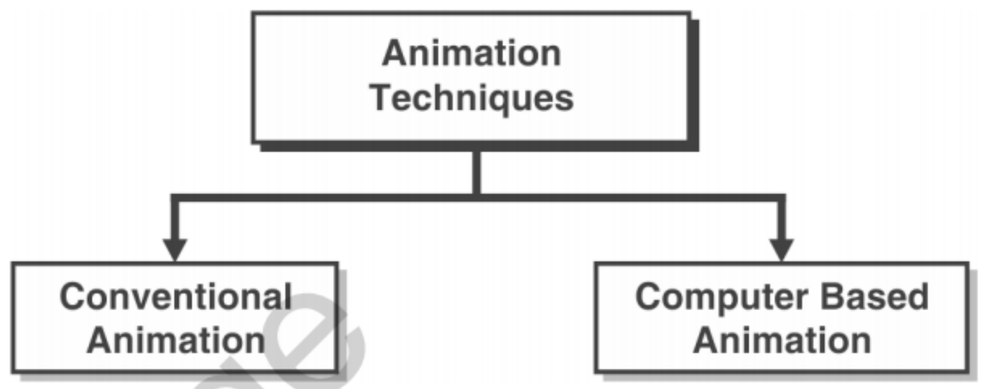


Fig. 11.2.1 : Types of animation techniques

- In the following subsections, we will discuss both the techniques

11.2.2.1 Conventional Animation

- In the conventional approach, all the frames of video are designed by hand. Frames are displayed at the rate of least at 24 frames per second. The conventional method takes a tremendous amount of time and effort to create a video.
- Conventional animation is carried out in the following steps:
- **Storyboard :** The sequence of animation is first drafted on paper in the form of sketches, which is called a storyboard.
- **Keyframes :** Keyframes are the first frame of every shot in the sequence. Keyframes are used to interpolate the animation sequence. As the animation is achieved using keyframes, such type of animation is also known as **key-frame animation**.
- **Inbetween :** Inbetweens are the set of frames between successive keyframes. Objects in motion gradually move in subsequent frames and reach to the destination in next keyframes.

Tech Knowledge Publications

- **Pencil Test :** Before the finalizing the complete animation sequence, trial video is created from keyframes and inbetweens, which is called a pencil test.
- **Cels :** Instead of applying animation to all the objects together, the animation scene is decomposed into small parts called cels. These cels may be independent or partially dependent. Each celis animated independently. This cel representation adds more flexibility to the animation.
- **Route Sheet :** Route sheet is one of the best methods to know exactly how each step of the project has progressed towards completion of the project.
- **Exposure sheet :** This is also known as camera instruction sheet or x-sheet. This tool allows the animator to organize his thoughts and give instruction to the camera operator how to shoot the animation. This sheet contains the minute description of each frame, including the position of the object, the position of the camera, dialogue, background etc.
- Conventional animation method is tedious but it provides great control over the animation to be created. Conventional animation is not limited by available computing technology. For high-quality animation, it is still faster than computer-based animation. The animation is only limited by the ability of the artist.

11.2.2.2 Computer-Based Animation

Computer-based animation is the process of creating animated images using a computer. Computer-based animations are typically 2D. Computer artist first creates objects either by drawing or using a computer. He designs the keyframes and mathematical algorithms are applied on keyframes to generate in-between frames. This process is called **tweening**.

Computer-generated animation consists of the following steps :

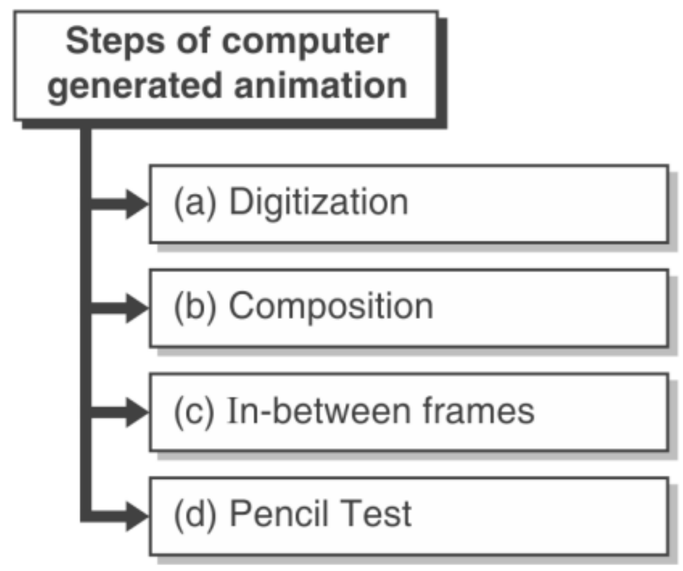


Fig. 11.2.2 : Computer-generated animation

- **Digitization :** Objects which could be part of intended animation are first digitized by one of the following methods :
  - Synthesize the object in graphics applications
  - Scan the document consisting of scene objects
  - Track the object
- Some pre-processing techniques may need to be applied to improve the pictorial information of such digitized objects.
- **Composition** : In this step, the keyframes are designed with the help of cels. Background and foreground of the scene are determined.
- **In-between frames :** In this step, successive keyframes are interpolated to generate inbetween frames. Depending upon the frame rate of the video, a number of frames are generated. Normal frame rate in the video is 24 fps or 30 fps. With 24 fps, we need to produce 240 frames to generate a video of 10 seconds.
- **Pencil Test :** Pencil test meant to check the correctess of the animation. Artist does clean up only after pencil test approved by higher authority like director of lead animator.

Technknowledge Publication

- Once artist clean up the drawings and paintings, if change is needed in animation then he has to start it from all over again. So pencil test is essential to check the flows in animation.
- Multiple small low resolution parts of the animation sequence is loaded in frame buffer, zoomed in to fit in monitor screen and pan to cover all different part of the frame buffer. The task is done fast enough to avoid the flicker.

11.2.2.3 Differentiate Conventional Animation vs. Computer-based Animation

Sr. No.	Conventional Animation	Computer-Based Animation
1	Does not use digital tools	Digital tools are used for such animation
2	Uses physical materials and activities	Uses virtual material
3	The animation is carried out in a physical environment	The animation is carried out in virtual/digital space
4	It is more time consuming	It is faster
5	Less accurate due to manual process	More accurate due to machine-oriented process
6	It has a limitation of artist's ability	It has a limitation of computing resource

11.2.3 Design of Animation Sequences

Q. Write a short note on the Design of Animation.

SPPU - May 18, 2 Marks

- In a frame by frame animation, each frame is generated using interpolation or some other methods. Frames are stored on film or they are consecutively displayed in playback mode.
- The animation sequence is defined by the following steps :
  - Storyboard layout
  - Object definition
  - Path specification
  - Keyframe specification
  - Generation of in-between frames

Even though this is a standard approach, some applications do not follow this sequence.

1. **Storyboard layout :** Storyboard layout is the center of the action. It defines the set of motion sequences taking place in the action. Storyboard defines the main actions of events. It consists of sketches of the important ideas of animation. In short, the storyboard is a set of basic events in animation.

2. **Object definition :** Scene is a collection of objects, which in turn made up of basic primitives like line, circle, ellipse, curve, surfaces etc. *Object definition* describes the geometric properties of each object present in the scene. Along with that, it stores associated movements for each object. For example, flying reply logo, walking man etc.

The object model can be described using one of the following technique :

- **Rigid :** Such objects are rigid and they do not have any moving part.

Tech Knowledge Publications

- **Articulated :** Such objects are formed using the parent-child hierarchy. A subpart of the object may be rigid, but the movement is allowed between two different subparts. For example, human hand, upper and lower part of the hand is a static object, they form motion through the elbow. Elbow is the articulation point between these two objects. Fig. 11.2.3 shows the articulated human body model. Circles are articulation points, subparts can move from articulation points. Part enclosed between two articulation points is rigid.

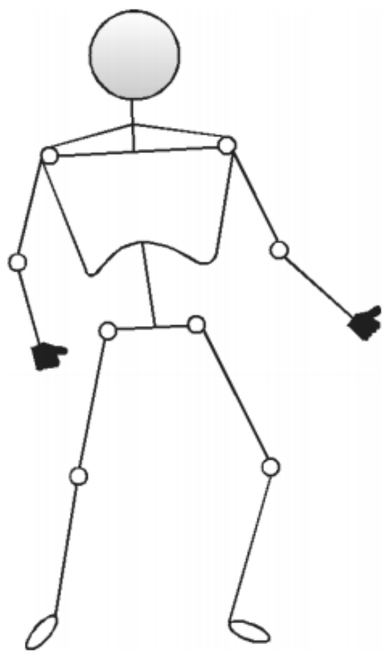


Fig. 11.2.3 : Articulated human model

- **Dynamic :** Such model employs the law of physics for the simulation of the object body.
- **Particle-based :** Flock of birds, smoke, group of aunts etc. follow the rule of nature. Set of motion determines the movement and collision of particles in space.
- **Behaviour-based :** Such types of animation mimic the behaviour of real animals of the planet.

3. **Path specification :** This specification tells about how the movement is added to the object. There are two types of specification.
   - **The object is stationary, camera in motion :** In this specification, the object remains stationary and the camera moves around the object. Camera records the different sides of the object. Camera path is specified in spatial coordinates. Along with camera path, speed of camera and orientation of the camera is also specified in software. Fig. 11.2.4 simulates the discussed concept.

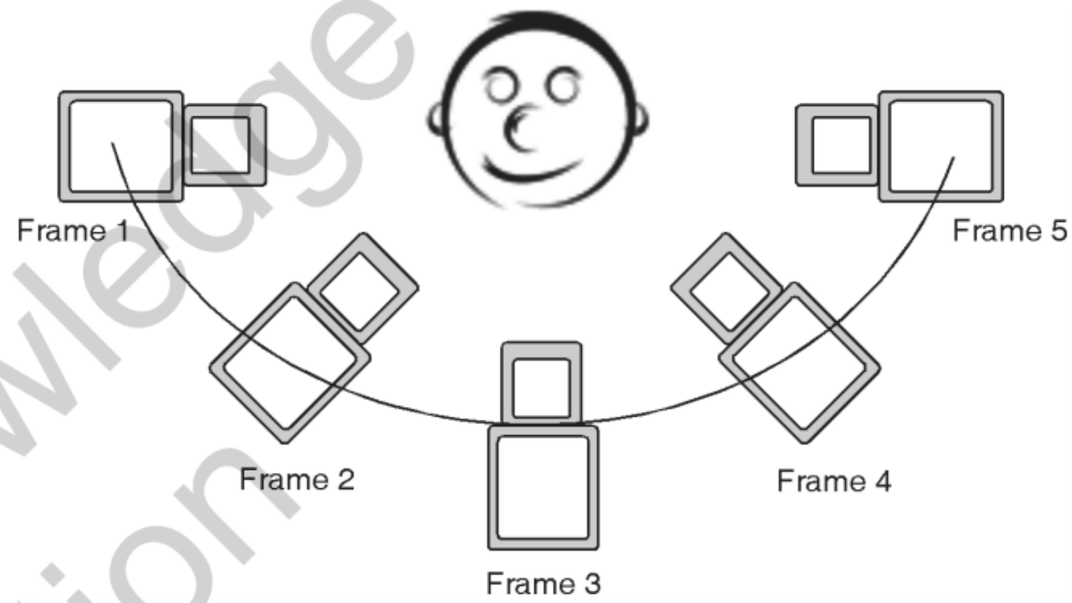


Fig. 11.2.4 : Object is static, camera in motion

   - **The camera is stationary, the object is in motion :** In this specification, the camera position is fixed during an animation, movement of the object is specified. If the object is rigid, the transformation for only one point needs to be specified. If object shape changes with time, shape interpolation are also required along with shape movement. Fig. 11.2.5 describes the rigid body object animation with a static camera.

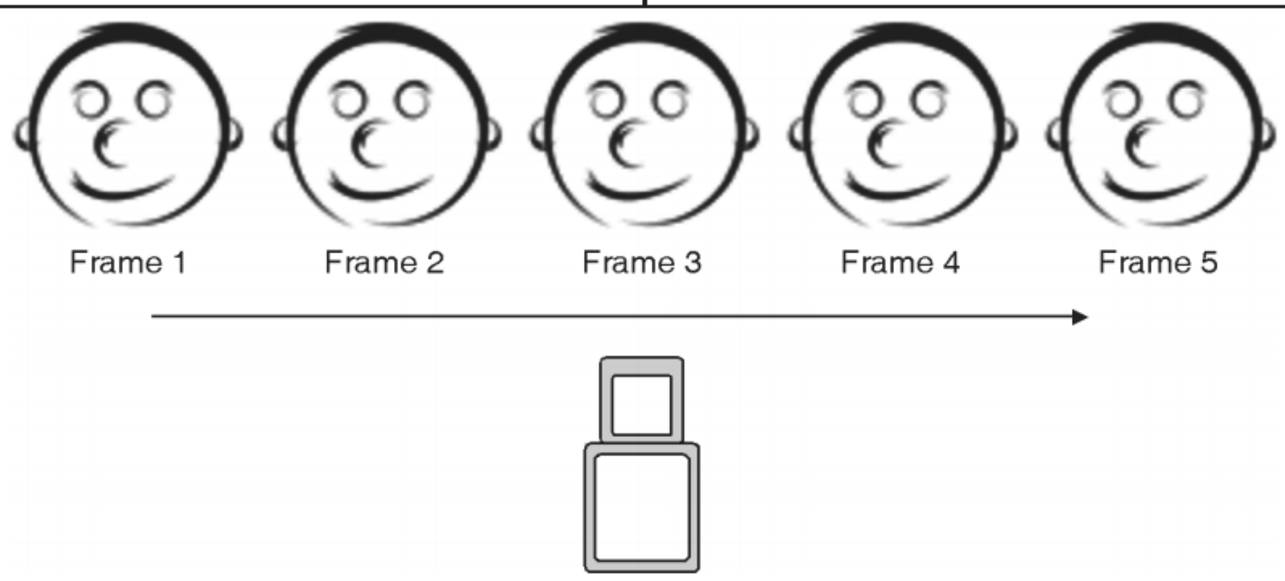


Fig. 11.2.5 : Camera is static, object in motion

TechKnowledge Publications

4. **Keyframe :** *Keyframe* is the actual representation of the scene at a certain time. Keyframe defines the position, scale and orientation of the object with its surface property for that particular time. To model abrupt changes over a short time, more keyframes need to be specified.

5. **In between frames :** *In between frames* are the intermediate frames between a pair of keyframes. Choices of numbers of in-between frames depend on the type of media on which animation is to be displayed. 24 frames are required for film, 60 frames are required for non-interlaced graphics monitor, and 30 frames are required for interlaced display. Typically, three to five in-between frames are set up between a pair of keyframes. For 1 minute of animation with 30 fps, we need 1800 frames. If we set up 4 *in-between* frames between successive keyframes, we need 450 keyframes. Fig. 11.2.6 explains the concept of in-between frames.

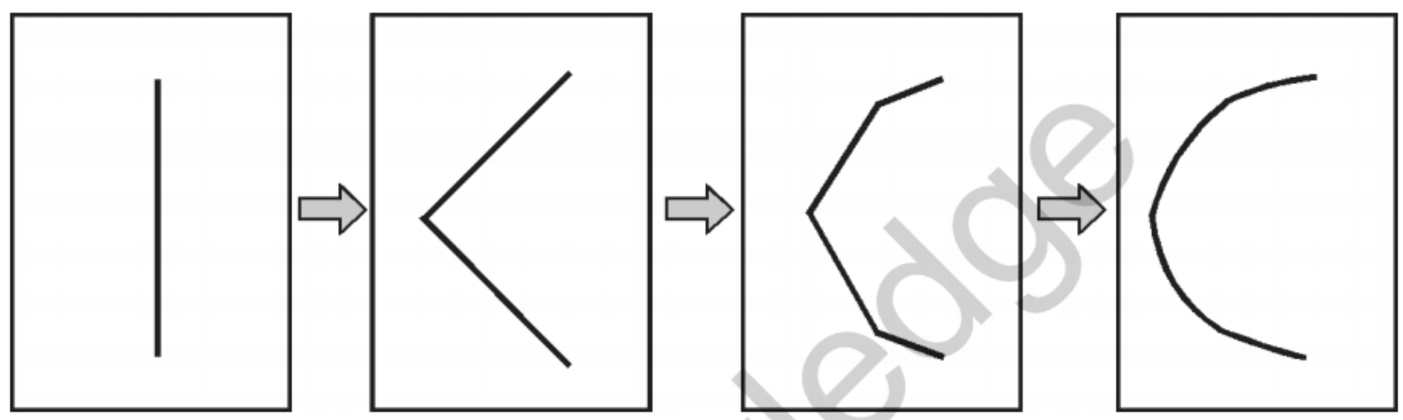


Fig. 11.2.6 : Shape transformation

11.2.4 Animation Languages

- Like other programming languages, the design and control of animation sequences are also handled with a set of animation routines.
- There are general-purpose languages such as C, Lisp, Pascal, FORTRAN can be used to program animation functions.
- Such general-purpose languages have limited functionalities and have many limitations in the context of animation. Many specialized programming languages have been developed.
- Animation functions typically include graphics editor, key frame generator, in-between generator and other graphics routines.
- With the help of graphics editor, we can design and modify the shapes using spline curves, surfaces, CGS and other representations.
- Some of the most crucial aspects of animation specification are positioning objects and light sources, specifying properties of surfaces and light sources, setting camera parameters etc. These all together form scene description.
- Another standard function is action specification, which involves motion path specification for objects and camera.
- We also need routines for viewing and geometric transformation, visible surface detection and surface rendering operations.
- **Key frame systems** are specialized language designed to generate in-between from the user-specified keyframes. Typically, objects are defined by rigid bodies connected at joints with a limited number of degrees of freedom.
- In **parameterized systems**, object motion characteristics are specified as a part of object definition itself. These parameters are tunable, which control the degree of freedom, motion limitations and allowable shape changes
- **Scripting systems** defines the object specification and animation sequences using input script. Scripting languages contain a rich set of libraries, which allows the construction of various objects and motions.

Technknowledge Publication

11.2.5 Keyframe

Q. Explain the term with an example - Keyframe.

SPPU - Dec. 17, May 18, May 19, 2 Mark

- Keyframe animation does interpolation to generate intermediate frames from a given pair of frames. The animator can control a number of intermediate frames to be generated. More keyframe generates smooth animation.
- Only a few seconds of animation of a bouncing ball produces many frames. To generate smooth and realistic animation, we should be careful in specifying position and number of keyframes in real sequence. Just by specifying the first and last frame as keyframes, we may end up with generating only liner motion of the ball.
- Following a sequence of frames are keyframes of the bouncing ball motion. These six frames are sufficient to explain and generate an animation of bouncing ball effect.

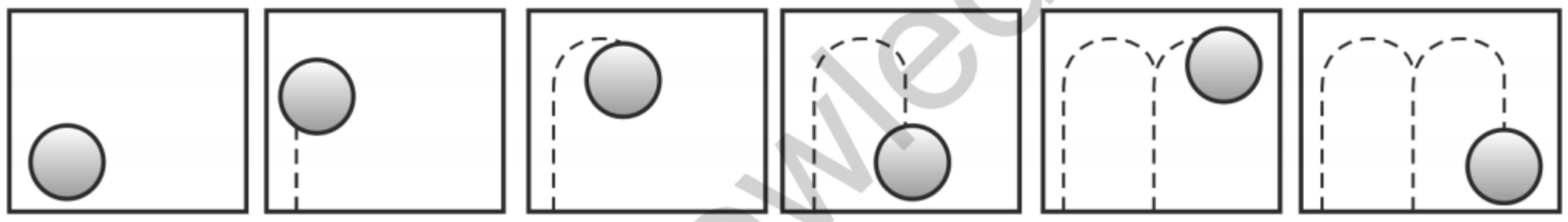


Fig. 11.2.7 : Keyframes for bouncing ball event

- Keyframe animation can be applied to motion, shape, color etc. Motion animation generates the intermediate frames defining the motion of an object at any given moment. In shape transformation, intermediate frame describes the transformation of one shape to another shape. In color transformation, in between frames determines the color change from source to the target frame.

Approaches for Keyframe Animation

There are two approaches to keyframe animation.

1. **Shape interpolation :** This approach derives in-between frames by interpolating the shape information. This has a major role in the film industry and advertisement. It performs geometric transformation to change the shape of the object. Motion can be distorted because of interpolation. Fig. 11.2.8 describes the shape interpolation.

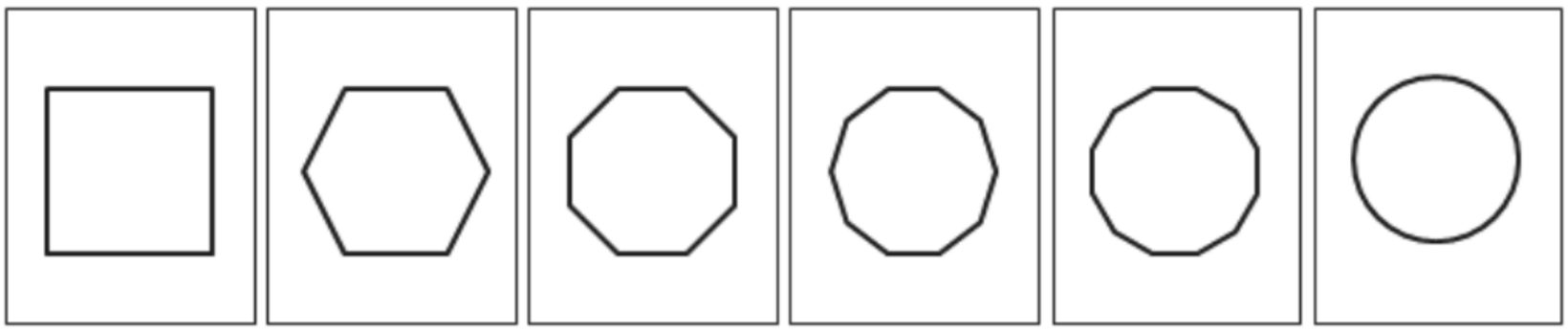


Fig. 11.2.8 : Shape interpolation

The problem arises when the numbers of vertices are not same in both objects. Typically, the pre-processing step is performed to equalize the number of vertices in both objects.

2. **Parameter interpolation :** Shape interpolation is faster but less accurate. Another way is to interpolate the parameters of both objects rather than object shape itself. This is also known as *parametric keyframe animation* or *key transformation animation*. Parameter interpolation considers various types of parameters like visual parameters, physical parameters, spatial parameters etc.

   In Fig. 11.2.9, points on keyframe k and k + 1 indicates parameters of both frames, intermediate frames gradually map parameters of keyframe 1 to parameters of keyframe k + 1.

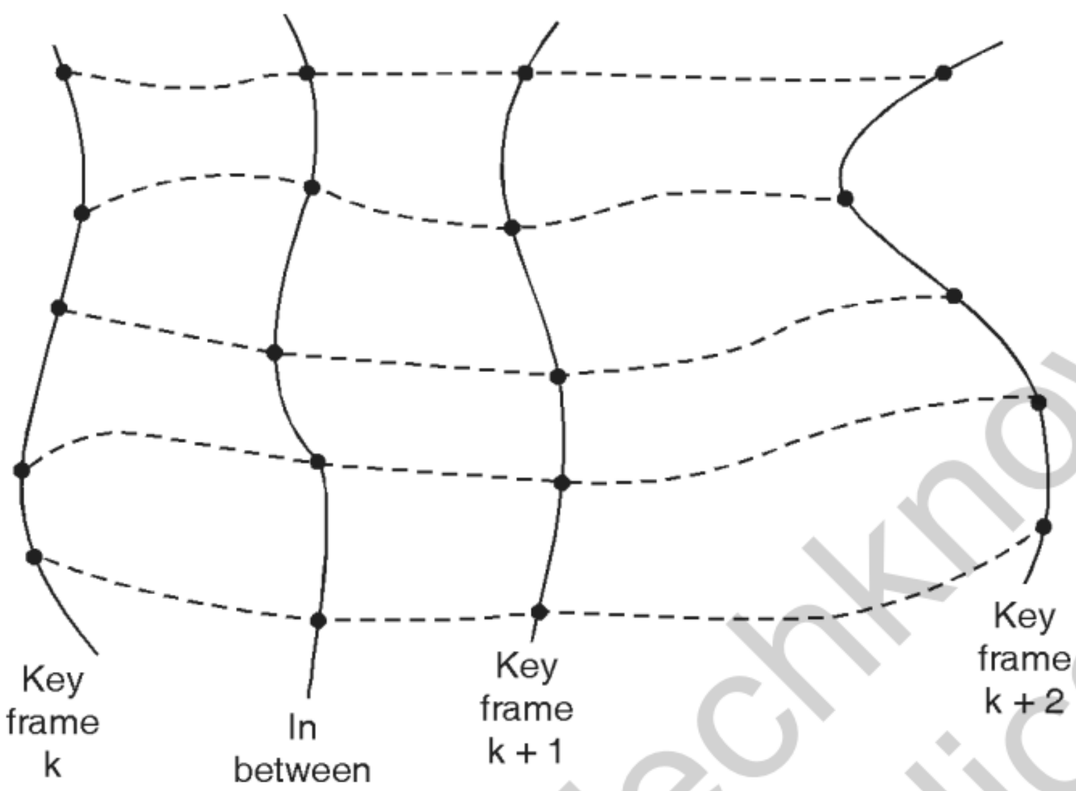


Fig. 11.2.9 : Parameter interpolation

   In both approaches, linear interpolation does not give acceptable results. Linear interpolation creates undesired effects like motion discontinuity, lack of smoothness in motion, distortion in rotation etc. Spline interpolation like quadratic or cubic interpolation can get rid of all these unwanted effects of linear interpolation.

11.2.6 Morphing

Q. What are the applications of morphing?

SPPU - May 15, 3 Marks

Q. What is morphing? **SPPU - May 17, 2 Marks**

Q. Explain the term with an example - Morphing.

SPPU - Dec. 17, May 18, May 19, 2 Marks

– *Morphing* is a way of smoothly transforming one shape to another shape. Animator has to provide first and last images, called source and destination images respectively. Intermediate images are computed from source and destination images using interpolation or some other mathematical formulas. Main techniques for computing intermediate frames are mesh morphing and field morphing.

– Morphing is carried out it two steps :

  i. **Cross dissolving :** Cross dissolving changes the color of the image at each pixel. In any in-between image, color at the location (x, y) depends on color at the same location in source and destination images. A simple morphing between two images of the same size can be achieved using cross dissolving alone.

     For example, color of a filled polygon can easily be changed using this method. If one shape is to be converted in some different shape, one needs to change the shape in prior or during cross dissolving.

  ii. **Warping :** Warping is generally used to change the shape object. Warping changes the shape features by shifting the pixels around in the image. Warping deforms the two images such that features of source image gradually map to features of the destination image. Deformed features are cross dissolved to carry out a smooth transition.

– In any technique, the intermediate frame is always represented as a weighted sum of the source and destination images. Beginning images have more influence of source image, intermediate images are average of source and destination images, and end images are similar to destination images. Somewhere near to middle, the source image is distorted to halfway forward, and the destination image is distorted halfway back. The blending of weighted source and destination frame forms an intermediate frame.

Technknowledge Publication

- For example, in a sequence of 100 frames, the 1st frame contains 100% weight age of source frame and 0% weight age of destination frame. As we move forward in a sequence of 100 frames, distance from source frame increase and its weight decreases, and distance from destination frame decreases and its weight increases.
- For 80th frame, the source frame contributes 20% and destination frame contributes 80%.
- The blending ratio of source and destination frame is given as,

$$S_{w_i} = \left(\frac{i}{\text{Total Frames}}\right) \% \text{ Source}$$

$$S_{D_i} = \left(\frac{\text{Total Frames} - i}{\text{Total Frames}}\right) \% \text{ Destination}$$

Where S_{w_i} and S_{D_i} is the weight of source and destination frame respectively.

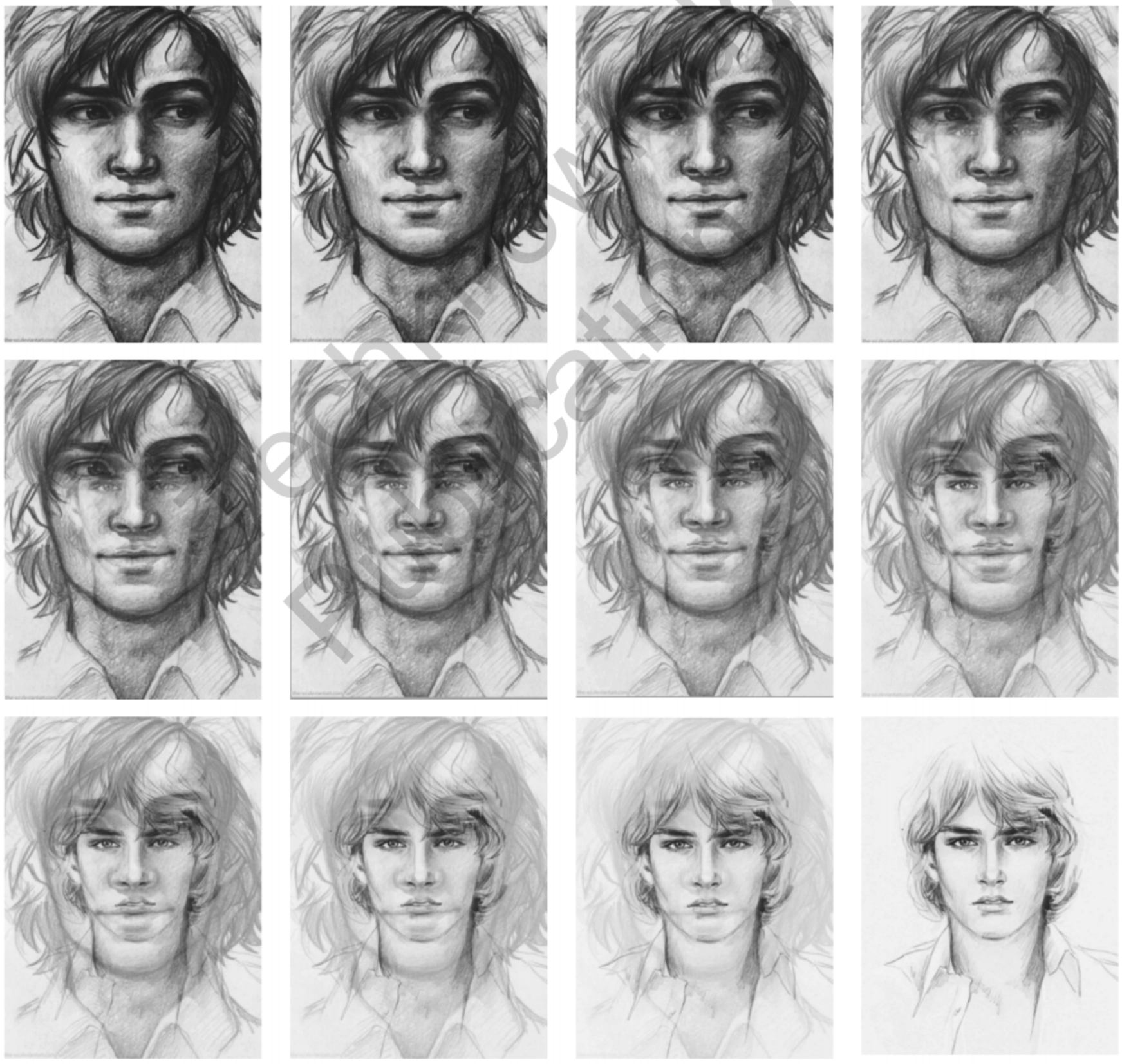


Fig. 11.2.10 : Morphing sequence

Tech Knowledge Publications

Applications of Morphing

- It is used in motion picture and animation to create special effects.
- Used to create face and object morphing in advertisement and marketing.
- Used to know how particular organ will look like after surgery.

11.2.7 Motion Specification

Q. Write short note on Motion specification.

SPPU - Dec. 19, 2 Marks

- The main issue with computer animation is describing the motion of an object in the scene. These methods are known as *Motion Control Methods* (MCMs). These methods specify how the object is animated.
- Methods may be classified according to the nature of information which is directly manipulated. This information may be geometric, physical or behavioural. Choice of a particular technique depends on automation and control.
- Some methods provide good automation with less control, while some method provides poor automation with better control. There is always a trade-off between these two parameters in the selection of motion control method.

11.2.7.1 Methods based on Geometric and Kinematics Information

- These methods are heavily animator dependent. Motion is a local entity, which is described in terms of velocity, acceleration, angles, coordinate positions etc. Performance animation, keyframe animation and image morphing belong to this class of motion control method.
- Performance animation uses magnetic or optical measurement in consistence with the recording of the direct action of real actor for an immediate or delayed playback. This technique is typically used in character animation.
- Rotation and transformation matrices are explicitly applied to associate motion with the object. An alternative way is to apply an equation that simulates the motion of an object. The motion of a bouncing ball can be approximated using the sine curve using Equation 11.2.1.

$$X(t) = A\,|\sin(\omega t + \theta)|\,e^{-mt} \qquad \ldots(11.2.1)$$

Where, t = Time,

A = Initial amplitude of the ball,

ω = Angular frequency,

θ = Phase angle,

m = Damping constant,

- This technique is fast compare to other. In key frame animation technique, animator specifies the kinematics by providing key frames. In between frames are produced by interpolation. Image morphing is also heavily used in film industry. Morphing interpolates parameters of to objects to create in between frames.
- Another way to provide motion is to specify the kinematics of the object. Kinematics includes motion parameters like position, velocity, acceleration, direction, etc.
- Velocity specifies the speed and direction of the object. Acceleration specifies the speed up or speeds down. Path of the object is typically described by splines.
- Inverse kinematics takes first and the last position of the object as an input and the motion parameters are computed automatically.

11.2.7.2 Methods Based on Physical Information

- Movement of an object under external force forms the physical modelling of the motion. Dynamics are physical modelling.
- In this method, physical data is provided by animator and motion computation is done by solving the dynamics equation. Here, motion is globally controlled. These classes of methods are further classified as *parameter adjustment* and *constraint-based methods*.

Techknowledge Publications

- According to Newton's second law, force applied on an object is proportional to the product of the mass of the object and velocity of the object.

$$F = m \cdot V$$

- Space-time constraints can be used to form character animation by solving constrained optimization. Lagrange equation may be used with an elastic model for realistic simulation of shape deformation. Wrinkles, animated clothes, curtains etc. are an active area of research under physics-based animation.

11.2.7.3 Methods Based on Behavioral Information

- This method uses the behavioural characteristics of an individual object by explicitly supplying high-level directives that indicate the behaviour without any stimulus (external force).
- The behavioural system is based on the interaction between actors. Actors in the system pass their position to surrounding actors, which in turn affects the neighbour's behaviour.
- For example, sequences of ants, flocks of birds or groups of fishes. To simulate such a scenario, the virtual object should be equipped with virtual sensors used as a basis for implementing the behavior of that object.
- The synthetic vision was the first concept to be used as the main information channel between the environment and the virtual object. This method also takes into consideration the relationship between different objects in the animation world.

11.3 Gaming

11.3.1 Introduction

- The video game is an electronic game which provides interaction between user and the video display device. The video display device may be 2D or 3D such as to monitor screen, touch screen, VR headset etc.
- Electronic systems used to play games are called platforms. Mostly, video games are designed for a specific platform, and may not work on another platform. A game designed for mobile may not work on a personal computer (PC), and vice-a-versa.
- Gradually, CRT display devices have been replaced by a flat screen, which ultimately replaced by touch screens. Most of the market of the game today is a mobile device.
- Games are being played on various platforms like a personal computer, console, handheld, Arcade, web browser, mobile, VR headset etc.
- **Personal computers** are general-purpose machine, so the performance of the game on another platform may vary vastly compared to a personal computer. PCs are easily upgradable and economical. It is a good option for multiplayer games. However, there is a small library of games for PCs and the performance of high graphics game is very average on it.
- **A console game** is played on a specific electronic device, designed for gaming purpose. It is a dedicated video gaming platform, and usually runs the games designed for that console only. Major console platforms are Xbox, PlayStation and Nintendo. Console games are relatively cheaper. It supports a huge amount of games and performance of sports games is excellent on such a specialized device. The downside of the console platform is that it is not upgradable, it has limited functionality other than playing the game and limited support to multiplayer games
- **Handheld** devices are small and portable. User can hold the device to play the game on it. It features a small screen, volume controller, joystick, speaker and other buttons. Handheld platforms are less powerful compared to PC or console
- **"Arcade game"** refers to the game played on a very specialized electronic device. Such devices are designed to play only one game. Arcade games are mostly housed in a vertical cabinet. Due to its limitation to support one game only, very few arcades are available in the country.

Tech Knowledge Publications

- **Web browsers** are a cross-platform environment. Games developed for web browsers can be played across a wide spectrum of devices. It makes accessibility of game anywhere, anytime with the internet. Such games are often designed in java or flash.
- With the rise of iOS and android operating systems and better computation capacity of mobiles, **mobile gaming** has become a significant platform for gamers. Mobile gaming covers a big revenue market for the gaming industries due to its availability to the user.
- **Virtual reality games** give a real feel to the user being a part of the game, but it requires additional VR headset. VR games are complex. Some VR systems provide various electronic support to interact within the game. VR games typically require a separate computer, console, VR headset and other equipment, which makes it costly.

11.3.2 Gaming Platform (NVIDIA, i8060)

Q. Explain in brief NVIDIA workstation.

SPPU - May 16, May 17, May 18, Dec. 18, 8 Marks

Q. Write any four important features of NVIDIA gaming platform. Explain the need for NVIDIA workstation in gaming. **SPPU - Dec. 17, May 19, 5 Marks**

11.3.2.1 NVIDIA

Q. Draw block diagram of NVIDIA workstation and explain it in brief. **SPPU - Dec. 19, 7 Marks**

NVIDIA has released Tesla, Quadro and Maximus workstations which are based on GPU and CUDA.

1. Tesla Workstation

- Tesla workstations are targeted at general-purpose graphics processing unit (GPGPU). It was invented by the electrical engineer Nikola Tesla.
- They are programmable using OpenGL or CUDA APIs.
- Tesla workstations are backed up by NVIDIA. They are built by big players in the markets like Lenovo, Dell etc.
- Tesla products are primarily used in simulations and large scale calculations (especially floating-point calculations), and for high-end image generation for applications in professional and scientific fields.
- Compared to GeForce cards, Tesla cards have four times the double-precision performance of a Fermi-based NVIDIA GeForce card of similar single-precision performance. On the other hand, Tesla cards were originally unable to output images to a display. However, the latest Tesla C-class products include one Dual-Link DVI port.
- NVIDIA Tesla GPU Accelerators turn standard PCs and workstations into personal supercomputers.
- It uses CUDA parallel computing architecture and it is powered up by 2688 parallel processing cores per GPGPU. This configuration makes them almost 250 times faster than normal PCs.

2. Quadro Workstation

- NVIDIA Quadro brand is intended to use in the workstation running Computer-Aided Design (CAD), Computer Generated Imagery (CGI), Digital Content Creation (DCC), machine learning and other scientific computation.
- The graphics card used in the Quadro is identical to the GeForce branded cards. It differs in ECC memory and the floating-point precision, which makes them very robust against the calculation error.
- Quadro is a series of the workstation. Following are some of the important features of Quadro5000 workstation:
  - It was launched in 2011, with GF100GL (Fermi) core support.
  - It supports 2560 MiB (MebiByte) memory of type 320-bit GDDR5.
  - It has 352 CUDA cores which makes it excellent for computation.

TechKnowledge Publications

- Quadro 5000 comes with more than twice as much memory as the GeForce GTX 465.
- Reliability is core to all Quadro solutions. Every product is designed to deliver the peace of mind you need to focus on what you do best.
- Even small single-bit errors can have a tremendous impact on the final result. ECC allows the graphics card to detect and correct this type of error, just like server and workstation motherboards can with system memory.
- Bindless Textures that give users the ability to reference over 1 million textures directly in memory while reducing CPU overhead.
- Increased frame buffer capacity of 4GB, plus a next-generation PCIe-3 bus interconnect that accelerates data movement by 2x compared with PCIe-2.
- An all-new display engine capable of driving up to four displays simultaneously with a single K5000.
- Display Port 1.2 support for resolutions up to 3840x2160 @60Hz.

3. Maximus Workstation

- Tesla was targeting the computational efficiency and Quadro was designed to target the graphics processing. There are application domains which demand both, computation and graphics processing.
- In many simulations, intensive computation is followed by the rendering of the results in no time.
- NVIDIA Maximus archives both the goal by bringing together the 3D graphics processing capabilities of NVIDIA Quadro with the massive parallel computing power of the NVIDIA Tesla C2075.
- It has been certified by the industry-leading vendors that the Maximus transparently assigns the work to the right processor.
- Applications such as those from Adobe, Autodesk, Mathworks etc. are NVIDIA Maximus enabled. Due to this, the CPU compute tasks are assigned to NVIDIA Tesla processor and it frees NVIDIA Quadro to take care of graphics functions only, ensuring a high level of performance with quality.
- World's leading workstation OEMs like HP, Dell, Lenovo and Fujitsu are offering such NIVIDA Maximus enabled workstations.

Features of NVIDIA

Features of NVIDIA gaming platform are discussed here :

1. **Share your favourite moments :** NVIDIA allows to record and shares live streaming of game on various social media platforms like youtube, Facebook etc. NVIDIA Shadow Play lets you broadcast with a bare minimum performance overhead
2. **Ready drivers :** For major games, NVIDIA continuously works on performance-boosting, bug fixing etc. to improve the gaming experience. NVIDIA Game Works lets you optimize gameplay and bring you these improvements through a Game Ready driver update. New drivers are automatically notified to the user when they are available
3. **Capturing in-game photographs :** NVIDIA Ansel photo mode allows you to capture the professional-looking high-quality photographs of your games. It also allows you to directly share on various social media platforms. User can choose from super-resolution, 360, HDR, and stereo photographs in supported games.
4. **Freestyle game filters :** NVIDIA Freestyle game filter allows you to apply post-processing filters on your games while you play. Change the look and mood of your game with tweaks to color or saturation, or apply dramatic post-process filters like HDR. Freestyle is integrated at the driver level for seamless compatibility with supported games.

Technknowledge Publication

11.3.2.2 Intel i860

Q.	Explain i860 with a block diagram.	**SPPU - May 15, May 16, 6 Marks**
Q.	Write a short note on i860.	**SPPU - May 17, 6 Marks**
Q.	Explain the architecture of i860.	**SPPU - Dec. 17, May 19, 4 Marks**
Q.	Write a short note on Architecture of i860.	**SPPU - Dec. 19, 3 Marks**

- The intel i860 (also known as 8086) was developed by Intel and was first released in 1989. It was a RISC microprocessor.
- It was designed to overcome the issues and failure of the earlier model i960. However, due to its commercial failure, the project i860 was terminated in the 1990s.
- The i860 XR was the first implementation of i860 architecture.
- Modified diagram (Intel copyright) is shown below :

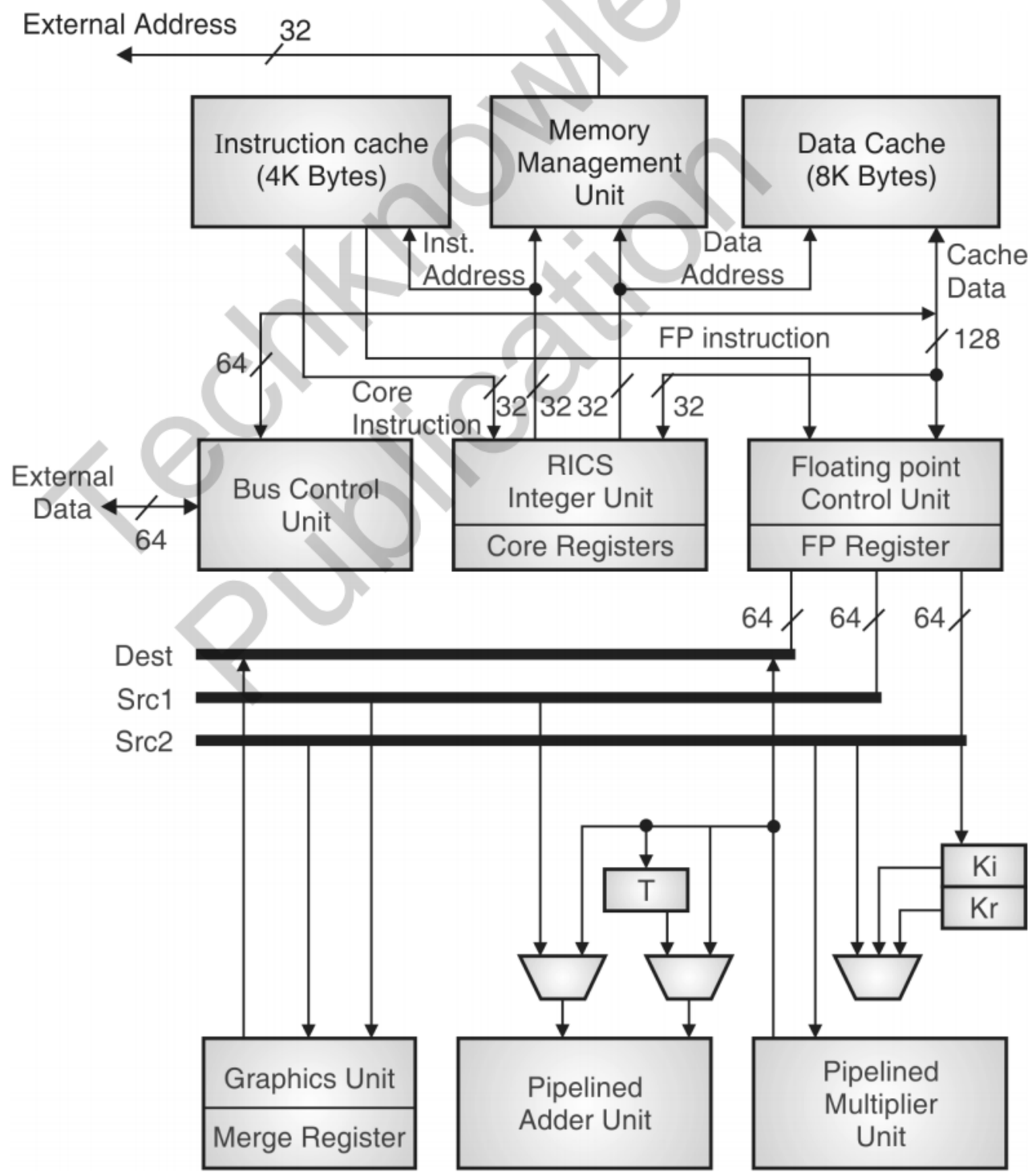


Fig. 11.3.1 : Functional units and data paths of the Intel i860 RISC microprocessor

- It's a very long instruction word (VLIW) architecture and the powerful support to high-speed floating-point computation was considered as the unique feature at the time.

- The design mounted a 32-bit ALU "Core" along with a 64-bit FPU that was itself built in three parts: an adder, a multiplier, and a graphics processor. The system had separate pipelines for the ALU, floating-point adder and multiplier, and could hand off up to three operations per clock.
- All of the buses were at least 64 bits wide. The internal memory bus to the cache, for instance, was 128 bits wide. Both units had thirty-two 32-bit registers, but the FPU used its set as sixteen 64-bit registers. Instructions for the ALU were fetched two at a time to use the full external bus. Intel referred to the design as the "i860 64-Bit Microprocessor"
- Intel i860 instructions acted on data sizes from 8-bit through 128-bit.
- The CPU could execute the majority of floating-point instructions either in pipelined mode or in scalar mode.
- In scalar mode, the instructions executed one after another, this took from 3 to 4 clock cycles per instruction.
- In pipelined mode instruction execution was broken into three or four stages, and the CPU could execute different stages of different instructions at the same time. As a result, one floating-point unit could provide a new result with every clock cycle.
- Two floating-point execution units (FP added and FP multiplier) could work in parallel, that is theoretically the CPU could provide two results with every cycle.
- Core execution Unit also worked in parallel with FP units.
- In addition to that, the processor could load and execute two instructions at the same time - one integer instruction and one floating-point instruction.
- The graphics unit was unique for the era. It was essentially a 64-bit integer unit using the FPU registers as eight 128-bit registers. It supported a number of commands for SIMD-like instructions in addition to basic 64-bit integer math. Experience with the i860 influenced the MMX functionality later added to Intel's Pentium processors.
- One unusual feature of the i860 was that the pipelines into the functional units were program-accessible (VLIW), requiring the compilers to order instructions carefully in the object code to keep the pipelines filled.
- In traditional architectures, these duties were handled at runtime by a scheduler on the CPU itself, but the complexity of these systems limited their application in early RISC designs.
- The i860 was an attempt to avoid this entirely by moving this duty off-chip into the compiler. This allowed the i860 to devote more room to functional units, improving performance.
- As a result of its architecture, the i860 could run certain graphics and floating-point algorithms with exceptionally high speed, but its performance in general-purpose applications suffered and it was difficult to program efficiently.

11.3.3 Advances in Gaming

Video games have come a long way since their inception in the 1980s. With new technology and computational power, some features have made gaming more attractive and exciting. Some of the advancements in gaming are discussed here :

1. **Facial recognition** : 3D scanning and facial recognition allow you to create avatar same like you in your gaming environment, or even you can transfer your expressions to built-in characters in the game. The 3D camera from Intel Real Sense allow the developer to transfer their emotions by scanning 78 landmark points on the face of the gamer.
2. **Voice recognition** : Voice recognition has brought the excitement level of game one step up. Today, you can issue the commands by voice rather than controller. User can search the game library, can turn the console on or off, interact with social media and partners etc.

Tech Knowledge Publications

3. **Gesture control** : Intel Real Sense 3D camera tracks 22 points of your hands and transfers the gestures into action in games. This free the user from operating the controller. For example, the *Warrior Wave* game employs Real Sense technology by which you can use your hand to lead a group of Ancient Greek soldiers to safety.

4. **Virtual reality** : VR technology has not fully exploited for gaming, but it has given the gamers a fully immersive gaming experience which they have never seen before. The player can feel himself in the scene with VR technology.

5. **Wearable gaming** : Earlier companies which were targeting the fitness applications in wearable devices are now using them for gaming. Wearables like a smartwatch, glasses make gaming portable and makes the game more exciting

6. **On-demand gaming** : Gamers can already watch and share live-streams of games, but what about playing them? Much like similar movie streaming services, the ability to stream video games is becoming more and more a reality, and it could lead game developers both big and small to compete for gaming glory.

7. **Cloud gaming** : Instead of creating video game systems that require more powerful hardware, developers are looking to lighten the load with the cloud. Games no longer need to be limited by the amount of memory that discs or consoles have to offer. Using the cloud opens games up to massive server-size limits where images are streamed to your screen through the Internet.

Review Questions

Q. 1 State the advantages of segments.

Q. 2 State the disadvantages of segments.

Q. 3 Write a short note on segment table.

Q. 4 Describe various operations performed on segment table.

Q. 5 Discuss various image transformation operations on segment.

Q. 6 Explain image transformation with example.

Q. 7 What is animation?

Q. 8 Discuss the steps of design of animation sequences.

Q. 9 Discuss various types of animation language.

Q. 10 Write a short note on key frame.

Q. 11 What is the importance of key frame? How it is useful in animation?

Q. 12 Explain the use of motion specification in animation.

Q. 13 Write a short note on motion specification methods based on :

(i) Geometric and kinematics information.

(ii) Specification methods based on physical information.

(iii) Motion specification methods based on behavioral information.

Q. 14 Explain the need of separate gaming platform.

Q. 15 Write a note on NVIDIA workstation.

Technical Publications

❑❑❑

Model Question Papers (In sem) Unit I & II

Technoknowledge Publication

Model Question Paper - I (In Sem.)

Computer Graphics (210244)

Semester - III (Computer Engineering)

Time : 1 Hour **Maximum Marks : 30**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Scan convert line with endpoints (10,5) and (16,10) using DDA algorithm. **[5]**

(b) Write and explain Bresenham's Circle drawing Algorithm. **[5]**

(c) Define Resolution, Aspect Ratio, Color Depth, Pixel, Frame buffer. **[5]**

OR

Q. 2 **(a)** Differentiate between Raster Scan and Random Scan. **[5]**

(b) Explain with the help of illustration Bresenham's Line drawing Algorithm. **[5]**

(c) With the help of block diagram explain Plasma Display. **[5]**

Q. 3 **(a)** What is polygon filling? Explain in detail scan fill Algorithm. **[5]**

(b) Explain Inside/Outside test of a polygon using winding number method. **[5]**

(c) Explain Sutherland-Hodgeman algorithm for polygon clipping. Compare it with Cohen Sutherland Clipping Algorithm. **[5]**

OR

Q. 4 **(a)** Explain the concept of generalized clipping with suitable example. **[5]**

(b) What are the types of Polygon? Explain Seed Fill Algorithm with suitable example. **[5]**

(c) Write algorithm to fill the polygon area using flood fill method. **[5]**

❑❑❑

Technical Knowledge Publication

Model Question Paper - II (In Sem.)

Computer Graphics (210244)

Semester - III (Computer Engineering)

Time : 1 Hour **Maximum Marks : 30**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Differentiate : Raster scan vs. Random scan. **[5]**

(b) Write a short note on Antialiasing. **[3]**

(c) Explain applications of computer graphics. **[7]**

OR

Q. 2 **(a)** Discuss the architecture of Open GL. **[5]**

(b) What is GLUT? **[3]**

(c) Draw a line from (0, 0) to (4, 6) using DDA line drawing algorithm. **[7]**

Q. 3 **(a)** What is a polygon? What are the different types of polygons? **[5]**

(b) Explain the boundary fill algorithm with pseudocode. Also mention its limitations, if any. **[5]**

(c) Derive transformation matrix for window-to-viewport transformation. **[5]**

OR

Q. 4 **(a)** Explain odd-even test with example. **[5]**

(b) Clip the line PQ having coordinates P(4,1)and Q(6,4) against the clip window having vertices A(3,2),B(7,2), C(7,6) and D(3,6) using Cohen-Sutherland line Clipping algorithm. **[7]**

(c) Differentiate : Flood fill vs. Boundary fill. **[3]**

❑❑❑

Technical Knowledge Publication

Model Question Paper - III (In Sem.)

Computer Graphics (210244)

Semester - III (Computer Engineering)

Time : 1 Hour **Maximum Marks : 30**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Discuss the characteristics of the ideal line. **[5]**

(b) Write a short note on shadow mask method. **[5]**

(c) Differentiate : Raster scan vs. Random scan. **[5]**

OR

Q. 2 **(a)** Explain Rendering Pipeline with respect to OpenGL. **[5]**

(b) Write an algorithm for Bresenham's circle drawing algorithm. **[5]**

(c) What do you understand by Texture mapping and what are the basic steps involved in it? **[5]**

Q. 3 **(a)** Explain Sutherland-Hodgeman polygon clipping algorithm. **[5]**

(b) Explain boundary fill approach. **[5]**

(c) Write and explain algorithm to fill the polygon area using flood fill method. **[5]**

OR

Q. 4 **(a)** Discuss generalized polygon clipping algorithm. **[5]**

(b) Explain scan line filling algorithm. **[5]**

(c) Derive transformation matrix for window to viewport. **[5]**

❑❑❑

Technical Knowledge Publication

Model Question Papers (End sem) Unit III to VI

Technoknowledge Publication

Model Question Paper - I (End Sem.)

Computer Graphics (210244)

Semester - III (Computer Engineering)

Time : $2\frac{1}{2}$ Hour **Maximum Marks : 70**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4, Q. 5 or Q. 6, Q. 7 or Q. 8.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Discuss DVST with its pros and cons. **[6]**

(b) State and explain the applications of computer graphics. **[6]**

(c) Explain the concept of odd and even field in interlaced display. **[6]**

OR

Q. 2 **(a)** Describe OpenGL architecture with block diagram. **[6]**

(b) Explain Rendering Pipeline with respect to OpenGL. **[6]**

(c) Explain and write steps for DDA line drawing algorithm. **[6]**

Q. 3 **(a)** Write a short note on window to view port transformation. **[6]**

(b) Write an algorithm for Bresenham's circle drawing algorithm. **[6]**

(c) Explain the boundary fill algorithm with pseudo code. Also mention its limitations, if any. **[5]**

OR

Q. 4 **(a)** Reflect a diamond shaped polygon whose vertices are A(– 1, 0), B(0, – 2), C(1, 0) and D(0, 2) about the line y = x + 2. **[6]**

(b) Comment on statement: "Two dimensional rotation and scaling are cumulative, if Sx = Sy or $\theta = n\pi$". **[6]**

(c) Derive transformation matrix for general pivot point rotation. **[5]**

Q. 5 **(a)** Derive transformation matrix for rotation about an arbitrary axis in space. **[6]**

(b) Find the transformation matrix for reflection through the plane passing from reference point R(xr, yr, zr) and having direction vector N = i + j + k. **[6]**

(c) Explain the CIE chromaticity diagram. **[6]**

OR

Q. 6 **(a)** Write short notes on Halftone shading. **[6]**

Technical Knowledge Publication

(b) Explain Phong specular reflection model in detail. **[6]**

(c) Explain depth sorting method. **[6]**

Q. 7 **(a)** Explain depth buffer method. **[6]**

(b) Write a short note on Hilbert's curve. **[6]**

(c) What are the limitations of the Bezier curve? **[5]**

OR

Q. 8 **(a)** Write a short note on Advances in Game playing. **[6]**

(b) Write a note on NVIDIA workstation. **[6]**

(c) Explain image transformation with example. **[5]**

Model Question Paper - II (End Sem.)

Computer Graphics (210244)

Semester - III (Computer Engineering)

Tech Knowledge Publications

Time : $2\frac{1}{2}$ Hour **Maximum Marks : 70**

Instruction to the candidates :

1) **Answer Q. 1 or Q. 2, Q. 3 or Q. 4, Q. 5 or Q. 6, Q. 7 or Q. 8.**
2) **Neat diagrams must be drawn wherever necessary.**
3) **Figure to the right indicates full marks.**
4) **Make suitable assumptions if necessary.**

Q. 1 **(a)** Define the terms: Computer Graphics, Scan conversion, pixel, resolution, aspect ratio, frame buffer. **[6]**

(b) Write a short note on shadow mask method. **[6]**

(c) State and explain the applications of computer graphics. **[6]**

OR

Q. 2 **(a)** Write short note on "Handling Keyboard Inputs with GLUT". **[6]**

(b) Discuss Open GL primitives. **[6]**

(c) What do you understand by Texture mapping and what are the basic steps involved in it? **[6]**

Q. 3 **(a)** Write an algorithm for Bresenham's Line drawing method. **[6]**

(b) Write and explain with an example Cohen-Sutherland line clipping algorithm. **[6]**

(c) Discuss the characteristics of the ideal line. **[5]**

OR

Q. 4 **(a)** Perform 90° rotation of triangle with vertices A(2, 2), B(4, 2) and C(3, 3).

1. About origin
2. About reference point (– 2, 2) **[6]**

(b) What is scaling ? Derive 2D transformation matrix for scaling with respect to origin and a reference point. **[6]**

(c) Derive transformation matrix for shearing with respect to line parallel to Y-axis. **[5]**

Q. 5 **(a)** Rotate a triangle with vertices A(0, 2, 1), B(2, 3, 0) and C(1, 2, 1) by 45° around a line passing through the points (0, 0, 0) and (1, 1, 1). **[6]**

(b) Explain orthographic parallel protection. **[6]**

(c) Explain the RGB color model. **[6]**

OR

Q. 6 **(a)** Explain Gouraud Shading algorithm. **[6]**

(b) Discuss Point and Distributed light source. **[6]**

(c) Write short note on Area Subdivision method. **[6]**

Q. 7 **(a)** Enlist the properties of Bezier curve. **[6]**

(b) Write a short note on uniform periodic B-spline curve. **[6]**

(c) Write short note on back surface detection method. **[5]**

OR

Q. 8 **(a)** State and discuss the features of NVIDIA. **[6]**

(b) Write a short note on key frame. **[6]**

(c) Write a short note on segment table. **[5]**

❑❑❑

Technical Knowledge Publication