

---

# Introduction and Application

---

## 1.1 Introduction

---

*Computer graphics* is a field or branch of science and technology related to creation, storage and manipulation of images of objects using computer. *Objects* can be of various fields. *Objects* may be the concrete real world objects or the abstract and synthetic objects such as mathematical surface, engineering structure, architectural design, survey results, etc. We see every day the images created by using computer in internet, books, magazines, movies, TV, etc.

Computer graphics is the rendering (making or servicing) tools for the generation and manipulation of images. These tools include both *hardware* and *software*.

*Hardware* comprises monitor, printer, plotter that displays graphics and input devices, such as, mouse, light pen, keyboard, scanner, etc. *Software* includes data structure, graphics algorithm and language. *Data structure* means a specialized format for organizing, processing, retrieving and storing data that are suitable for computer graphics. *Graphics algorithm* refers to algorithm for picture generation and transformation. *Language* means high level language for generation of graphics or pictures of objects.

*Graphics* is an image or a visual representation of an object. *Computer graphics* is the image or visual representation of an object on a computer screen. It is created by plotting *pixels* of the screen. A *pixel* is the smallest screen point which can be controlled by setting intensity values. A *pixel* is also called picture element or elementary part of the computer screen. It is the smallest addressable element of a picture represented on a screen.

**Computer Graphics** = Computer + Graphics

**Computer** = System = Hardware + Software

**Hardware** = Input devices + Output devices

**Software** = Data structure + Graphics algorithm + Language

**Graphics** = Graph + Pics

**Graph** = Pictorial representation or a diagram that represents data or values in an organized manner. It is the collection of all points whose coordinates satisfy a given relation (such as a function). Graph is mathematical figure like line, circle, ellipse etc

**Pics** = Images, photograph or motion picture

Computer graphics can be either two-dimensional or three-dimensional. These digital graphic files can be divided into two categories:

- i. Raster graphics
- ii. Vector graphics

## 1.2 Raster Graphics

A *raster graphic* is a pixel-based graphic. Raster graphics are rendered images on a pixel-by-pixel basis. Raster images use pix maps to store information or image definition. Raster graphics are well fit when handling shading and gradients.

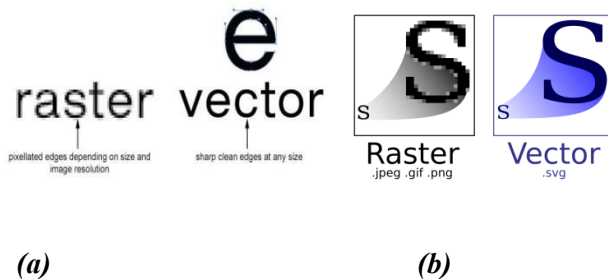
A raster graphics, such as a GIF or JPEG file is an array of pixels of various colors, which together form an image. Raster graphics are the most common and used for digital photos, web graphics, and other types of images. Most pictures that we see on our digital screen are *raster images*. The selfie, screenshot, and mspaints are examples of raster images.

When a raster image is scaled up, it loses quality. A raster image can be enlarged by either adding more pixels or enlarging the size of the pixel. In either way, original data is spread over a larger area at the risk to losing clarity. Enlarging a bitmap makes it pixelated or simply blurred.

### **Raster graphics based file format:**

- .jpg Joint Photographic Experts Group (JPEG)
- .png Portable Network Graphics (PNG)

- .gif Graphics Interchange Format (GIF)
- .tiff Tagged Image File Format (TIFF)
- .psd Adobe Photoshop File
- .pat Corel Paint File



**Figure 1.1:** *Raster and vector graphics*

### **Advantages of raster graphics:**

In raster graphics, complex images can be created with various color changes and variations as every pixel in a raster image can be of different color.

Raster graphics are useful for creating detailed images.

Many programs and software work with a simple raster file. Adobe Photoshop, MSpaint, Corel Painter and other software handles raster graphics.

Raster graphics is used in daily life, like screenshot, camera photo, etc. It is suitable for painting, editing and web application.

### **Disadvantages of raster graphics**

When raster images are scaled up, it looks grainy, distorted, and blurred. Raster images are created with a finite number of pixels. As the size of a raster image is increased, gaps are created between the pixels in the image. Resampling or interpolation distorts color.

Raster images hold more data, so it is slower to edit. Raster files are often quite large. Raster files contain all the information for every single pixel of the image. Each of these pixels has an X

and Y coordinates as well as color information associated with it. Therefore, raster graphics files tend to be very large. Increase of size of image increases size of file in raster graphics.

Raster graphics are not suitable for embroidery. Raster images are based on square pixels, so, the embroidery may look like it has jagged edges. If we want to embroider an image with smoother edges, it is best to use vector graphics instead of raster graphics.

### 1.3 Vector Graphics

*Vector graphics* (also called geometric modeling or object oriented graphics) is not pixel based graphics. Vector graphic is composed of path while raster graphics is composed of pixels. *Vector graphics* are made by using sequential commands or mathematical statements or programs that make lines or shapes.

Vector graphic is made up of anchor points and geometrical primitives such as points, lines, curves, and polygons which are all based upon mathematical equations. A vector has a magnitude and direction. In vector graphics, the file is created and saved as a sequence of vector statements. Rather than having a bit in the file for each bit of line drawing we use commands which describe series of points to be connected.

A *vector graphic* can be scaled to any size without losing quality. A vector graphics, such as an .eps file or Adobe Illustrator file, is composed of paths (geometrically calculated outline), or lines, that are either straight or curved. The data file for a vector image contains the points where the paths start and end, how much the paths curve, and the colors that either border or fill the paths. Vector graphics are often used for creating logos, signs, and other types of drawings. Raster image's dimensions are measured in pixels (pixel per inch-ppi). Vector graphics are resolution independent.

The most recognized applications which handle vector based graphics are Adobe illustrator, Macromedia freehand, Autocad and

Corel draw. *Vector graphics* are generally used for line art, illustrations, and embroidery.

All modern current computer video displays translate vector representation of an image to a raster format.

**Vector graphics based file format:**

- .eps    Encapsulated PostScript File (EPS)
- .svg    Scalable Vector Graphics (SVG)
- .ai    Adobe Illustrator File
- .cdr    Corel Draw File
- .pdf    Portable Document Format (PDF)

**Advantages of vector graphics:**

Vector files are small because they contain very less data than raster files. Vector graphics are more flexible than raster graphics because they can be easily scaled up and down without any loss of quality of the image. Vector graphics have smoother lines in comparison to raster graphics. In vector graphics, increase of size of image doesn't increase the size of file. It is suitable for drawing and designing less detail images like logo, icon, etc.

**Disadvantages of vector graphics:**

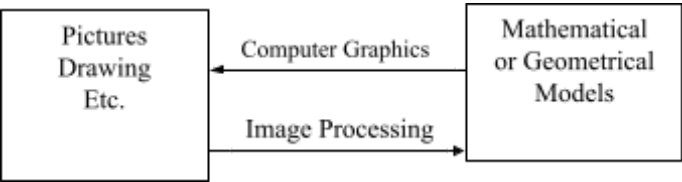
If there are small errors or faults in a vector graphic, these will be seen when the vector image is enlarged significantly. Vector graphics can't display the abundant color depth of a raster graphic. It is complicated as every anchor point for every object has to be placed in vector program

**1.4 Computer Graphics and Image Processing**

The difference between computer graphics and image processing can be synthesized as follows:

Computer graphics	Image processing
1. It is the field related to the generation of pictures using computers.	1. It applies technique to modify or interpret existing pictures.

Computer graphics	Image processing
2. It synthesizes pictures from mathematical or geometrical models.	2. It analyzes picture to derive description in mathematical or geometrical forms.
3. It includes the creation, storage, and manipulation of images of objects	3. It is the part of computer graphics that handles image manipulation or interpretation
4. E.g., drawing a picture	4. E.g., making blurred image visible.



**Figure 1.2:** Computer graphics and image processing

### 1.5 History of Computer Graphics

History of computer graphics dates from early days of computing, crude plotting of hard copy devices like teletype and line printer.

Some major development of computer graphics can be described as follows.

- In 1950's outputs were via teletypes, line printer and *Cathode Ray Tube (CRT)*. Using dark and light character, pictures were reproduced.
- In 1950, Massachusetts Institute of Technology (MIT) developed whirlwind computer having CRT displays for output. It was the first computer to display real time video and capable of displaying real time text and graphic on a large oscilloscope screen.

- In 1950, Been Laposky created the first graphic images on *oscilloscope* generated by an electronic (analog) machine. The image was produced by manipulating electronics beams and recording them onto high speed film.
- In 1951, UNIVAC-I, the first general purpose commercial computer; crude hardcopy devices, and line printer were invented.
- In middle 1950's SAGE (Semi-Automatic Ground Environment) air defenses system was developed. It was the first to use command and control CRT display consoles on which operator identifies target, with *light pen* (hand –held pointing devices that senses light emitted by objects on screen).
- In 1960's, modern *interactive graphics* was begun. Outputs were vector graphics and interactive graphics, but the problems were the cost and inaccessibility of machine.
- In 1960, William Fetter coined the *computer graphics* to describe new design methods.
- In 1961, Stove Russel made space wars, first video/computer game.
- In 1963, Douglas Englebart developed first *mouse*. Ivan Sutherland made sketch pad, interactive computer graphics System, a man-machine graphical communication system, its features: pop-up menus, constrained – based drawing, hierarchal modeling, and utilized light pen for interaction.
- He developed the dragging, rubber banding and transforming algorithms, He introduced data structures for storing. He is considered data founded of computer graphics.
- In 1964, William Felter developed first computer model of a human figure.
- In 1965, Jack Bresenham developed line drawing algorithm.
- In 1968, Tekronix made a special CRT, the direct view storage tube, with keyboard and mouse, a simple computer interface for \$15,000 which made graphics affordable.

- In 1969, John Warnock developed area subdivision algorithm, hidden –surface algorithms. Bell labs developed first frame buffer containing 3 bits per pixel. *CAD* (computer aided design), *CAM* (computer aided manufacturing) with enormous potentials for automating drafting and other drawing – intensive activities, were developed. The general motors' DAC system for automobile design and the Itek Digitek system for lens design were pioneering efforts that showed the utility of graphical interaction in the interactive design cycles common in engineering.

A number of commercial products using these systems were appeared. But hardware was expensive.

- In the early 1970's, Output start using raster displays, graphics capability was still fairly chunky.
- In 1971, Rendering model *Gouraud shading* was developed.
- In 1972, Nolan kay Bushnell Pong made video arcade game.
- In 1973, John whitney Jr. and Gary Demos made “Westworld”, first film with computer graphics.
- In 1974, Edwin catmuff developed texture mapping and z -buffer hidden surface algorithm. James Blim developed curved surfaces, refinement of texture mapping.
- In 1974 – 77, *Phong shading* (rendering model) was developed.
- In 1977, Steve Wozmak made Apple II, color graphics personal computer.
- In the 1980's, outputs were built in raster graphic, bitmap image and pixel, personal computers cost decrease drastically, trackball and mouse became the standard interactive devices..
- In late 1980's, Artists and graphic designers preferred to use Macintosh and PC.
- In 1982, Ray tracing (Illumination based rendering method) was developed. In 1982,Steven Lisberger made ‘Tron’, first



Disney movie which made extensive use of 3-D computer graphics. John Walkner and Dan Drake developed *Auto CAD*

- In 1983, Jaron Lanier made 'Data Glove', a virtual reality film features a glove installed with switches and sensors of detection hand motion.
- In 1984, Waveform tech developed Polhemus, first 3D graphics software
- In 1987, IBM introduced *VGA* (Video Graphics Array)
- In 1989, Video Electronics Standard Association (VESA) formed SVGA(Super VGA)
- In 1990's, since the introduction of VGA and SVGA personal computer could easily display photo realistic images and movies 3D image rendering were become the main advances and it stimulated cinematic graphics application.
- In 1990, Render man system that provides fast accumulate and high quality digital computer effects was developed.
- In 1992, Silicon graphic developed open GL specification.
- In 1993, Mosaic, first graphic web browser and Jurassicpark, a successful CG fiction film was made.
- In 1995, 'Toy story', first full length computer generated feature film was made.
- In 2003, ID software developed Doom graphics engine.

## **1.6 Applications of Computer Graphics**

Computer graphics is widely used in diverse areas; from display of information to design of industrial product, user interface to virtual reality, simulation to academic purpose, and entertainment to medical treatment. The major applications of computer graphics can be synthesized as follows:

1. User interfaces
2. Plotting graph and charts
3. Office automation and electronic publishing
4. Computer aided design and drafting

5. Scientific and business visualization
6. Simulation and virtual reality
7. Entertainment
8. Art and commerce
9. Civil engineering applications
10. Medical applications
11. Internet

## **1. User Interfaces**

The interface between the human and the computer has been radically changed by the use of computer graphics. Most applications have a Graphical User Interface (GUI) for user friendly and interactive operation. Today's software has user interfaces that rely on the desktop window systems to manage multiple simultaneous activities and on point and click facilities to select items menu, icon, and objects on the screen.

## **2. Plotting or visualization of measurement data**

Graphics is extensively used in plotting 2D and 3D graphs such as the histograms, bar diagram, pie chart, task scheduling charts of mathematical, physical, and economic functions. These plotting or visualization of measurement data is useful to analyze meaningfully and concisely the trend and pattern of complex data.

## **3. Office automation and electronic publishing**

Nowadays, the necessary document that contains text, tables, graphs, drawings, pictures can be easily printed or saved as electronic (softcopy) document. The office automation and electronic publishing became possible by the development of computer graphics.

## **4. Computer aided design and drafting**

A major use of computer graphics is in design process. Computer graphics is used to design components and systems (including building and other structural design) of architectural systems, mechanical, electrical, electrochemical

and electronic devices, automobile bodies, aircrafts, watercrafts, spacecrafts, very large scale integrated (VLSI) chips, optical systems, and telephone and computer networks.



*Figure 1.3: Computer aided design*

## **5. Scientific and business visualization**

Scientific visualization means generating computer graphics for scientific works and medical data sets. Business visualization is generating computer graphics for non scientific data sets such as economic data set. Visualization makes easier to understand the trends and pattern inherent in huge amount of data sets. It would otherwise be almost impossible to analyze those data numerically.

## **6. Simulation and virtual reality**

One of the most impressive and familiar uses of computer graphics is simulation and virtual reality. Simulation is the imitation of the conditions like those, which is encountered in real life. Virtual reality is an interactive computer-generated experience taking place within a simulated environment, that incorporates mainly auditory and visual, but also other types of sensory feedback like haptic. Simulation helps to learn or feel the condition one might have to face in near future without being in danger at the beginning of the course. Astronauts' simulator, flight simulator, military simulator, naval simulator, driving simulator, air traffic control simulator, and heavy duty

vehicle simulator are some of the mostly used simulators in practice.

## **7. Entertainment**

Computer graphics is used in making games, special effects in movies, music videos, television shows, etc. Sometimes, the graphics scenes are displayed totally using computer graphics and sometimes, graphics objects are combined with the real actors and live scenes. Computer and video games such as FIFA, Formula-1, Doom and Pools are few to name where computer graphics is used extensively. Disney movies such as Lion King, The Beauty and the Beast, and other scientific movies like Star Trek are the best examples of application of computer graphics in the field of entertainment.

## **8. Art and commerce**

Computer graphics are used in both fine art and commercial art or advertisement. The ability to create any shape and play with any color with the help of computer graphics opened the new realm of art and commerce. Computer graphics is used to produce a picture that expresses a message and attract attentions.

The pictures generated by computer graphics are frequently used at transportation terminal, super markets, hotel, etc. The slide production for commercial, scientific, or education presentation is another cost effective use of computer graphics. One of such graphics package is Power Point.

## **9. Civil engineering applications**

Civil engineering applications include cartography, GIS (geographical information system), etc. Computer graphics is used to produce both accurate and schematic representation of geographical and other natural phenomenon from measurement data. Cartography includes geographic map, oceanographic chart, weather map, color map, and population density map. Surfer is one of such graphic packages which is extensively used for cartography.

## **10. Medical applications**

Computer graphics has become a powerful tool for diagnosis and treatment in medical fields. X-ray, video x-ray, complex operation, etc. are done using computer graphics method and techniques in medical field.

## **11. Internet**

There is a large amount of multimedia content available on net. Internet became famous because of the development of computer graphics.

## **1.7 General Term and Terminologies**

### **1. Pixel (or pel)**

Pixel (picture element) is defined as the smallest screen element. It is the smallest piece of the display screen which can be controlled. The screen point is controlled by setting the intensity and color of the pixel.

### **1. Pixel density**

Pixel density is the number of physical pixels per inch on a screen or display of a device. It can be calculated by using Pythagorean Theorem, dividing the number of pixels in hypotenuse by length of hypotenuse measured in inch. If the resolution of the screen with 4 inch width and 3 inch height is  $1,920 \times 1,080$ , then its pixel density (ppi) is 440.

### **2. Aliasing**

Real objects or lines, polygon, edges are continuous but a raster device is discrete. The digitization of continuous signal produces jaggies i.e., a staircase problem. The sampling process digitizes the coordinate points and it produces staircase appearance. This process of distortion of information due to sampling is called aliasing.

### **3. Antialiasing**

It is defined as the process which compensates the consequences of under sampling process.

### **4. Pixel phasing**

It is a hardware based antialiasing technique in which the graphics system shifts individual pixels from their normal positions in the pixel grid by a fraction (typically 0.25 and 0.5) of the unit distance between the pixels. By moving pixels closer to the time line, this technique is very effective in smoothing out the stair steps without reducing the sharpness of the edges.

## **5. Interlacing**

It is used when the perpetual threshold is greater than the frequency of standard line voltage. If refresh rate is greater than phosphor's persistence, then moving objects become blurred. If refresh rate is lesser than phosphor's persistence, then it creates flickering. Interlacing is used to break the raster line into two sweep patterns consisting of half the number of raster lines in original patterns.

## **6. Bit depth (or color depth)**

It is defined as the number of bits needed (assigned) to a pixel in the image. It specifies the number of colors that a monitor can display. For example, if a pixel is denoted by a byte (8 bits), then the total number of color that can be displayed per pixel is  $2^8 = 256$ .

## **7. Fluorescence and phosphorescence**

When the electron beam, emitted from electron gun, strikes the phosphor-coated screen of the CRT, some of this energy is dissipated as heat but the rest of energy is used to make the electron of the phosphor atoms to jump to higher quantum energy level. Fluorescence is the light emitted by very unstable electrons while the phosphor is being struck by electrons. Phosphorescence is the light given off by stable excited electron to their unexcited state once the electron beam excitation is removed. Fluorescence usually last for a fraction of microsecond. Most of the light emitted is phosphorescence.

## **8. Persistence**

Persistence is how long phosphors continue to emit light (that is to have excited electron returning to ground state) after the CRT beam is removed. More precisely, persistence is the time to decay to  $1/10^{\text{th}}$  of its original intensity of the emitted light. That is, how long phosphorescence persists is the persistence. Lower persistence phosphors require higher refresh rates to maintain a picture on the screen without flicker. The phosphor with low persistence is useful for animation. A high persistence phosphor is useful to highly complex static pictures. Graphics monitors are usually constructed with persistence in the range from 10 to 60 micro second.

## **9. Refresh rate**

The light emitted by the phosphor fades very rapidly. So, some method is needed for maintaining the screen picture. One way to keep the phosphor glowing is to redraw the picture repeatedly by quickly directing the electron beam back over the same points. Refresh rate is the number of times the image is redrawn per second to give a feeling of picture without flick. It is the frequency at which the content of the frame buffer is sent to the display monitor. Refresh rate is usually 50 frames per second. Refresh rate above which flickering stops is called critical fusion frequency (CFF). The factors affecting CFF are persistence, image intensity, ambient room light, and wave length of emitted light.

## **10. Horizontal scan rate**

The horizontal scan rate is the number of scan lines per second. The rate is approximately the product of the refresh rate and the number of scan lines.

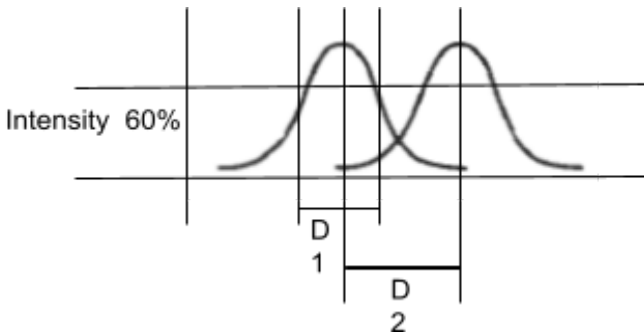
## **11. Resolution**

Resolution is the maximum number of pixels (points) that can be displayed horizontally and vertically without overlap on a display device. More precisely, it is the number of pixels per unit length that can be placed horizontally and

vertically. It is the number of pixels in horizontal direction  $\times$  number of pixels in vertical direction.

**Factors affecting the resolution:**

- i) **Spot profile:** The spot intensity has a gaussian distribution as depicted in figure. So, two adjacent spots on the display device appear distinct as long as their separation  $D_2$  is greater than the diameter of the spot  $D_1$  at which each spot has an intensity of about 60 percent of that at the center of the spot.



*Figure 1.4: Gaussian distribution of spot intensity*

- ii) **Intensity:** As the intensity of electron beam increases, the spot size on the display tends to increase because of spreading of energy beyond the point of bombardment. This phenomenon is called blooming, and consequently, the resolution decreases.

**12. Aspect ratio**

It is defined as the ratio of vertical points to horizontal points necessary to produce equal length lines in both directions on the screen. Aspect ratio  $3/4$  means that a vertical line plotted with three points has the same length as a horizontal line plotted with four points. It is the ratio of image's height to its width.

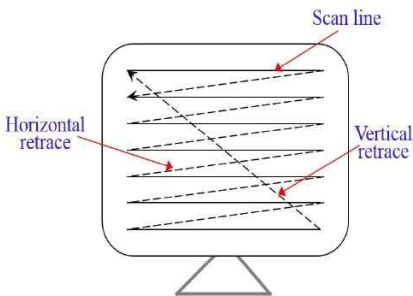
600 $\times$ 800 pixels in display has the aspect ratio

$$\text{A.R.} = 3:4$$

**13. Horizontal and vertical retrace**



Horizontal retrace meansat the end of each scan line, the returning of the electron beam to the left side of the screen to begin displaying the next scan line. The return to the left of the screen, after refresh each scan line is called the horizontal retrace of the electron beam.At the end of each frame ( of a second), the electron beam returns to the top left corner of the screen to begin the next frame. It is called vertical retrace.



**Figure 1.5:** Scan line, horizontal retrace, and vertical retrace

**14. Refresh buffer/ frame buffer/ bit map/ pix map:**

In raster-scan system, the electron beam is swept across the screen, one row at a time from top to bottom. As the electron beam moves across each the beam intensity turned on and off to create a pattern of illuminated spots.Picture definition is stored in a memory. The memory is called the refresh buffer or frame buffer.This memory area holds the set of intensity values for all the screen points. Stored intensity values are then retrieved from the refresh buffer and 'painted' on the screen one row (scan line) at a time. Each screen point is referred to as a pixel or pel (picture element). On black and white system with one bit per pixel, the frame buffer is commonly called a bit map. For system with multiple bits per pixel, the frame buffer is commonly called a pixmap.

Color depth	Number of color displayed	Byte of storage per pixel	Common name for color depth

4bit	16	0.5	Standard
8bit	256	1	256 color mode
16 bit	65,536	2	High color
24 bit	16,717,216	3	True color

## 1.8 Hardware Concepts

Hardware means physical parts used in computer graphics. Hardware includes input and output devices. Any device that feeds data into computer is called input devices and output devices are those physical parts that convert information into a human perceptible form.

### 1. Tablet

Tablet, a digitizer, is a device used to scan an object. A tablet digitizes an object detecting the position of a movable stylus (a pencil shaped device) or a puck (a mouse like device with cross hairs for sighting positions) held in the user's hand. These discrete co-ordinate positions can be then joined with straight line segments to approximate the shape of original object. A tablet is a flat surface and its size varies from 6 by 6 inches up to 48 by 72 inches or more. The accuracy of the tablet usually varies from about 0.2 mm on desktop models to about 0.05mm or less on larger models.

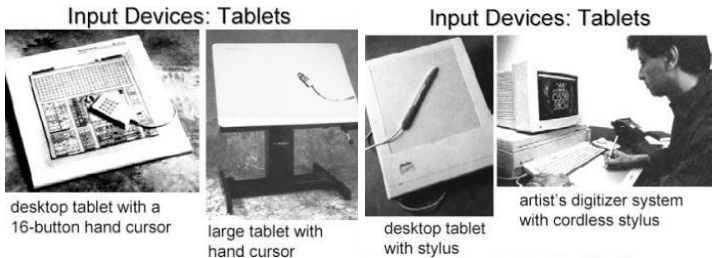
#### Types of tablet:

- i. Electrical tablet
- ii. Sonic (acoustic) tablet
- iii. Resistive tablet

#### i) Electrical tablet

In electrical tablet, a rectangular grid of wires is embedded in the tablet surface. Electromagnetic pulses are generated along the wires and electric signal is induced in a wire coil in the stylus or puck. The strength of the signal induced by each pulse is used to determine the position of the stylus. A signal is sent to the computer

when the tip of the stylus is pressed against the tablet or when any button on the puck is pressed. The information provided by the tablet repeats 30 to 60 times per second.



**Figure 1.6: Tablets**

## **ii) Sonic (acoustic) tablet**

Sonic (acoustic) tablet uses sound waves to detect the stylus position. Microphone is used to detect the sound emitted by an electrical spark from a stylus tip. The position of the stylus or the co-ordinate values is calculated using the delay between when the spark occurs and when its sound arrives at each microphone. The main advantage of sonic tablet is that it doesn't require a dedicated working area as the microphones can be placed on any surface to form the tablet work area. This facilitates digitizing drawing on thick books.

## **iii) Resistive tablet**

Resistive tablet is a piece of glass coated with a thin layer of conducting material. When a battery is powered, stylus is activated at certain position. The device emits high frequency radio signals which induces the radio signals on its conducting layer. The strength of the signal received at the edges of the tablet is used to calculate the position of the stylus. Several types of tablets are transparent, and thus can be backlit for digitizing x-ray films and photographic negatives. The resistive tablet can be used to digitize the objects on CRT because it can be curved to the shape of the CRT. The mechanism used in the

electrical or sonic tablets can also be used to digitize the 3D objects.

## **2. Touch panels**

The touch panel allows the user to select the screen positions directly with the touch of a finger to move the cursor around the screen or to interact with the icons. There are three types of touch panels:

- i. Optical touch panel
- ii. Electric touch panel
- iii. Sonic (acoustic) touch panel

### **i. Optical touch panel**

Optical touch panel employs a series of infrared light emitting diodes (LED) along one vertical edge and along one horizontal edge of the panel. The opposite vertical and horizontal edges contain light detectors which are used to record beams that are interrupted when the panel is touched. Touching the screen breaks one or two vertical and horizontal light beams. The interrupted beams identify the coordinate positions.

### **ii. Electrical touch panel**

Electrical touch panel consists of slightly separated two transparent panels one coated with a thin layer of conducting material and the other with resistive material. When the outer plate is touched with a finger, it is forced into contact with the inner plates by creating the voltage drop across the resistive plate which is then used to calculate the co-ordinate of the touched position.

### **iii. Sonic (acoustic) touch panel**

In sonic (acoustic) touch panel, high frequency sound waves traveling alternately horizontally and vertically are generated at the edge of the panel (glass plate). Touching the screen causes part of each wave to be reflected back to its source. The screen position at the point of contact is then calculated using the time interval between the

transmission of each wave and its reflection to the emitter.

### 3. **Light Pen**

It is a pencil shaped device used to select the co-ordinates of a screen point by detecting the light coming from the points on the CRT screen. In raster display 'Y' is set at  $Y_{\max}$  and 'X' changes from 0 to  $X_{\max}$  in the first scan line. For the second line, 'Y' decreases by one and 'X' again changes from 0 to  $X_{\max}$  and so on. When activated light pen sees a burst of light at certain position as the electron beam hits the phosphor coating at that position, it generates an electric pulse. This is used to save the video controller's 'X' and 'Y' registers and interrupt the computer. By reading the saved value, the graphics package can determine the co-ordinates of the position seen by the light pen.

### 4. **Keyboard**

Keyboard is used for entering text. It consists of alphanumeric key, function keys, cursor-control keys, and separate numeric pad. It is used to move the cursor, to select the menu, item, predefined functions. In computer graphics, keyboard is mainly used for entering screen co-ordinate and text to invoke certain functions. Nowadays, ergonomically designed keyboard (ergonomic keyboard) with removable palm rests is available. The slope of each half of the keyboard can be adjusted separately.

### 5. **Mouse**

Mouse is a small handheld device used to position the cursor on the screen. It can be picked up, moved in space, and then put down again without any change in the reported position. For this, the computer maintains the current mouse position, which is incremented or decremented by the mouse movements. Following are the mice, which are mostly used in computer graphics:

#### i. **Mechanical mouse**

When roller in the box of this mechanical mouse is moved, a pair of orthogonally arranged toothed wheels, each placed in between LED and a photo detector, interrupts the light path. The numbers it interrupts, so generated, are used to report the mouse movements to the computer.

## **ii. Optical mouse**

The optical mouse is used on a special pad having grid of an alternating light and dark lines. A LED in the bottom of the mouse directs a beam of light down onto the pad from which it is reflected and sensed by the detectors on the bottom of the mouse. As the mouse is moved the reflected light beam is broken each time a dark line is crossed. The number of pulses so generated, which is equal to the number of lines crossed, are used to report mouse movements to the computer.

## **6. Barcode reader**

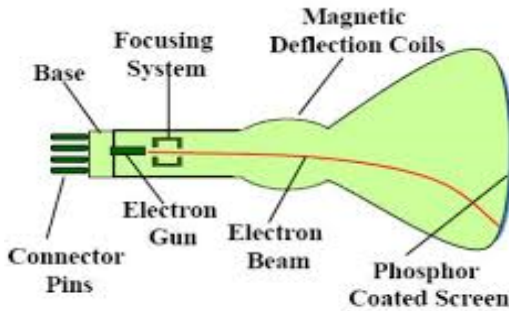
A barcode reader (or barcode scanner) is an electronic device for reading printed barcodes. Like a flatbed scanner, it consists of a light source, a lens, and a light sensor. Barcode reader translates optical impulses into electrical ones. Additionally, nearly all barcode readers contain decoder circuitry analyzing the barcode's image data provided by the sensor and sending the barcode's content to the scanner's output port.

## **7. Data glove**

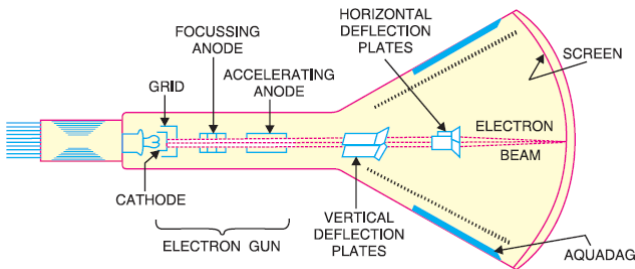
A data glove is an interactive device resembling a glove worn on the hand, which facilitates tactile sensing and fine-motion control in robotics and virtual reality. Data gloves are one of several types of electromechanical devices used in haptics applications. Tactile sensing involves simulation of the sense of human touch and includes the ability to perceive pressure, linear force, temperature, and surface texture. Fine-motion control involves the use of sensors to detect the movements of the user's hand and fingers and the translation of these motions into signals that

can be used by a virtual hand (for example, in gaming) or a robotic hand (for example, in remote-control surgery).

## 1.9 Refresh Cathode Ray Tube



**Figure 1.7:** Basic design of a magnetic-deflection CRT



**Figure 1.8:** Operation of an electron gun with an accelerating anode.

It consists of a CRT along with control circuits. CRT is a vacuum glass tube with the display screen at one end and connectors to the control circuits at the other. Inside of display screen is a special material called phosphor which emits light for a period of time when hit by a beam of electrons. The color of light and the time period vary from one type of phosphor to another.

**The major parts of refresh CRT are:**

### 1. Electron gun

Electron gun is made up of heated metal cathode and a control grid. It produces the beam of free electron.

#### i. Heated metal cathode

Heat is supplied to the cathode by directing a current through a coil by wire, called heating filament, inside the cylindrical cathode structure. This causes electron to be “boiled off” the cathode surface.

## **ii. Control grid**

Control grid is responsible for controlling the brightness of a display. By setting voltage level on control grid; the brightness emitted by phosphor coating depends on the number of electrons that strike the phosphor coat.

## **2. Focusing system/focusing anode**

It is required to force the electron beam to converge into small spot when it strikes the phosphor coat. Otherwise, the electrons would repel each other and the beam would spread out as it approaches the screen. There are two types of focusing system: electrostatic focusing and magnetic field focusing. Additional focusing hardware is used in high precision systems to keep the beam in focus at all screen positions. Because of radius of curvature for CRT monitor as beam moves to the outer edges of the screen displayed image may be blurred. To adjust this, the focusing is necessary.

## **3. Accelerating anode**

In the vacuum inside the CRT envelope, the freely negatively charged electrons are accelerated towards the phosphor coating by a high positive voltage (15000-20000). This high positive voltage can be generated by using accelerating anode.

## **4. Deflection system**

Deflection system is needed to direct the electron beam towards a particular point on the screen. It is done in two ways: electrostatic deflection system and magnetic deflection system. When electron beam passes through the horizontal and vertical deflection plates, it is bent or deflected by the electric fields between the plates. The horizontal plates control the beam to scan from left to right



and retrace from right to left (horizontal retrace). The vertical plates control the beam to go from the first scan line at the top to the last scan line at the bottom and retrace from the bottom back to the top (known as vertical retrace).

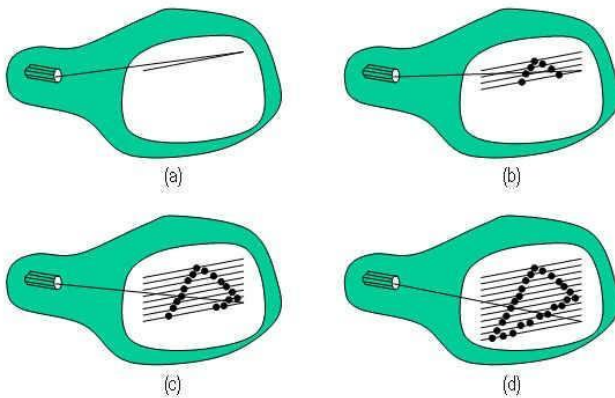
## 1.10 Raster and Random (Vector) Scan Display

On the basis of image generation process, display can be divided into two types.

i) Raster Scan Display

ii) Random (Vector) Scan Display

### 1.10.1 Raster Scan Display



**Figure 1.9:** Raster scan display

Most common type of graphics monitors are employing a Cathode Ray Tube (CRT), based on television technology. The electron beam is swept across the screen one row at a time from top to bottom. Picture definition is stored in memory area called “refresh buffer” or “frame buffer”. The set of intensity values is retrieved from refresh buffer and “painted” on the screen one row at a time. Each screen point is referred as a “pixel” or “pel”. For black and white system, each screen point (pixel) is represented by one bit either “on” or “off”. For color system, additional bit is

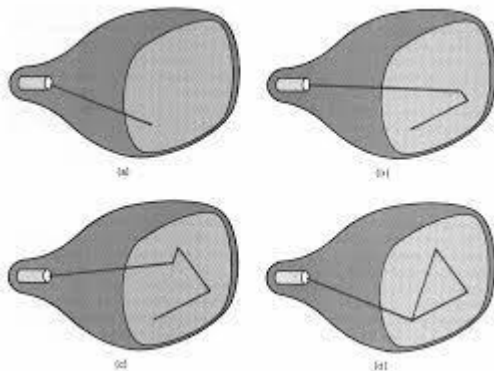
needed to represent single pixel (24 bit per pixel in high quality systems). For black and white system with one bit per pixel, the frame buffer is called "bitmap" whereas for system with multiple bits per pixel, the frame buffer is called "pixmap". Normal refresh rate is 50-60 frames per second.

## Interlacing

Some monitors use a technique called "interlacing" to double the refresh rate. In this case, only half of the scan lines in a frame are refreshed at a time, first the odd numbered lines, and then the even numbered lines. Thus, the screen is refreshed from top to bottom in half time it would have taken to sweep across all the scan line. This effect is quite effective in reducing flicker.

### 1.10.2 Random (Vector) Scan Display

---



**Figure 1.10:** Vector scan display

In random (vector or calligraphic or stroke writing) scan display, electron beam is directed and moved only to the points of the screen where a picture is to be drawn e.g., pen plotter (hard copy device). In this system, refresh rate depends on the number of lines to be displayed. Picture definition is stored as a set of line drawing commands in system memory called display file (or display program or display list). To display a specified picture, the system cycles through the set of commands in display file, drawing each component line in turn. After all line drawing commands have

been processed, the system cycles back to the first line command in the list. Random scan displays are designed to draw all the components of a picture 30 to 60 times each second.

Random scan display consists of CPU, a display processor, a monitor, system memory and peripheral. Random scan display was developed in 1960's and used as common display device till 1980's.

**1.10.3 Difference between Raster Scan Display and Random (Vector) Scan Display**

Base of difference	Raster scan display	Random (vector) scan display
1. Electron beam	The electron beam is swept across the screen one row at a time from top to bottom.	The electron beam is swept to the parts of the screen where a picture is to be drawn.
2. Resolution	It has lower or poor resolution because picture definition is stored as an intensity value.	It has high resolution because it stores picture definition as a set of line commands.
3. Picture definition	Picture definition is stored as a set of intensity values for all screen points (pixels) in a refresh buffer.	Picture definition is stored as a set of line in a display list or file.
4. Realistic display	The capacity of the system to store intensity values for pixels make it well suited for realistic display with shadow and color pattern.	These systems are designed for line-drawing and can't display realistic shaded scenes.

Base of difference	Raster scan display	Random (vector) scan display
<b>5. Image drawing</b>	Screen points or pixels are used to draw an image.	Mathematical functions are used to draw an image.
<b>6. Cost</b>	They are cheaper than random display.	It is more expensive than raster-scan display.
<b>7. Refresh rate</b>	Refresh rate is 60-80 fps.	All components are drawn 30 to 60 times per second.
<b>8. Interlacing</b>	It uses interlacing.	It doesn't use interlacing.
<b>9. Editing</b>	Editing is difficult.	Editing is easy.
<b>10. Refresh area</b>	Refresh area is independent of picture complexity.	Refresh area depends on complexity of picture.
<b>11. Smoothness</b>	Produce jagged line.	Produce smooth line.
<b>Example:</b>	CRT, TV, Printer	Pen Plotter

## 1.11 Color CRT Monitors

Color CRT monitors display color pictures using a combination of phosphors that emits different colored light.

Two basic techniques are available:

- i. Beam penetration method
- ii. Shadow mask method

### i) Beam penetration method

In this method, two different layers of phosphor coating used red (outer) and green (inner). It displays color depending on the depth of penetration of electron beam into the phosphor layer.

- i. A beam of slow electron excites only the outer red layer.

- ii. A beam of very fast electrons penetrates through the red phosphor and excites the inner green layer.
- iii. When quantity of red is more than green, then color appears as orange.
- iv. When quantity of green is more than red, then color appears as yellow.

Screen colors as controlled by the beam acceleration voltage. Only four colors are possible and picture quality is poor.

## ii) **Shadow mask method**

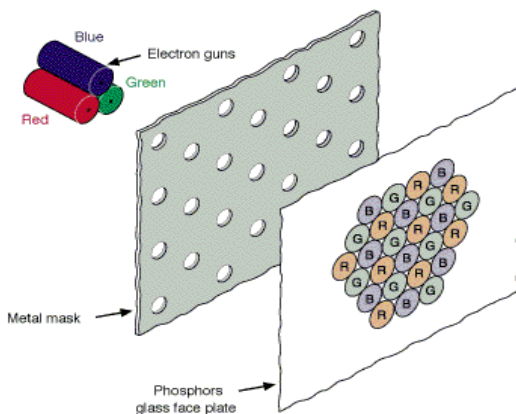
The inner side of the viewing surface of a color CRT consists of closely spaced groups of red, green, and blue phosphor dots. Each group is called a triad. A thin metal plate perforated with many small holes is mounted close to the inner side of the viewing surface. This plate is called shadow mask. The shadow mask is mounted in such a way that each hole is correctly aligned with a triad in color CRT. There are three electron guns one for each dot in triad. The electron beam from each gun therefore hits only the corresponding dot of a triad as the three electron beams deflect. A triad is so small that light emanatory from the individual dots is perceived by the viewer as a mixture of the three colors.

### **Two types of shadow mask method:**

- a. Delta-delta CRT
- b. Precision inline CRT

#### **a) Delta-delta CRT**

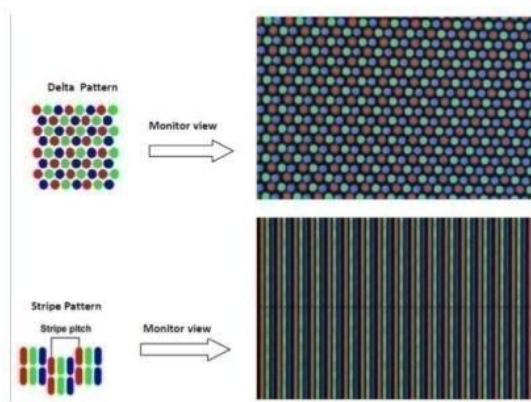
A triad has a triangular or delta pattern as are three electron guns. Main drawback of this method is that a high precision display is very difficult to achieve because of technical difficulties involves in the alignment to shadow mask holes and the triad on one to one basis.



**Figure 1.11:** Delta-delta CRT

## b) Precision inline CRT

A triad has an *in-line pattern* as are the three electron guns. The introduction of this type of CRT has eliminated the main drawback of a delta-delta CRT. Normally, 1000 scan lines can be achieved by this method.

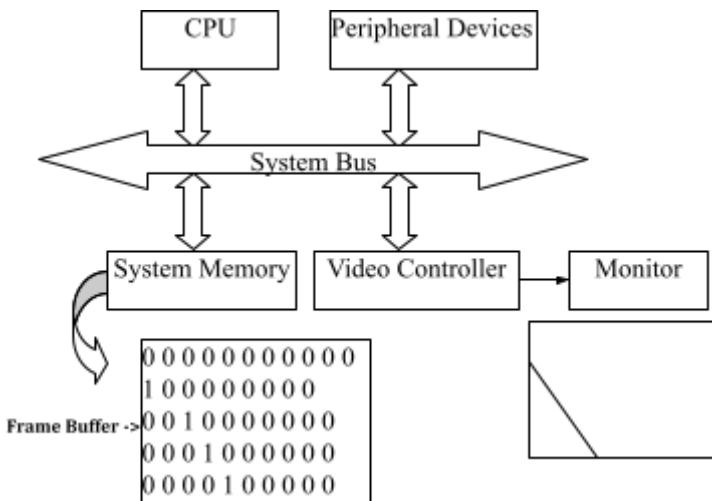


**Figure 1.12:** Delta-delta and precision inline CRT

But in this method, a slight reduction of image sharpness at the edges of the tube has been noticed. Normally, 1000 scan lines can be achieved. The necessity of triad has reduced the resolution of a color CRT. The distance between the centers of adjacent triads is called a *pitch*. In

very high resolution tubes, pitch measures 0.21 mm (0.61 mm for home TV tubes). The diameter of each electron beam is set at 1.75 times the pitch. For example, if a color CRT is 15.5 inches wide and 11.6 inches high and has a pitch of 0.01 inches. The beam diameter is therefore  $0.01 \times 1.75 = 0.018$  inches. Thus, the resolution per inch is about  $\approx 55$  lines. Hence, the resolution achievable for the given CRT is 850 ( $=15.5 \times 55$ ) by 638 ( $=11.6 \times 55$ ). The resolution of a CRT can therefore be increased by decreasing the pitch. But small pitch CRT is difficult to manufacture because it is difficult to set small triads and the shadow mask is more fragile owing to too many holes on it. Beside the shadow is more likely to warp from heating by the electrons.

## 1.12 Raster Scan Display System/ Architecture/ Technology



**Figure 1.13:** Raster Scan System

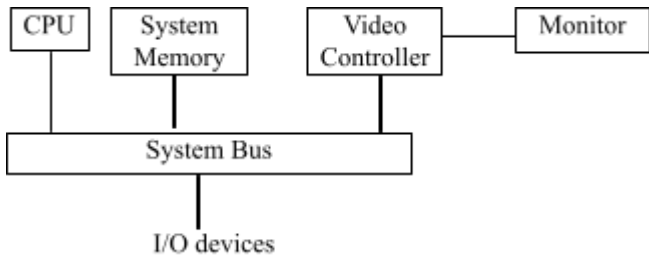
Raster scan system consists of CPU, a video controller (special-purpose processor), a monitor, system memory, and peripheral devices.

**Advantages:**

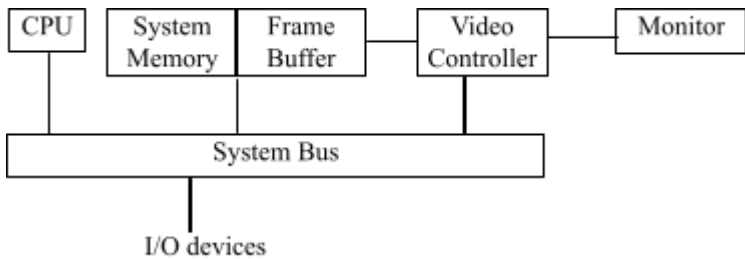
It has an ability to fill the areas with solid colors or patterns. The time required for refreshing is independent of the complexity of the image. It has low cost.

**Disadvantages:**

For real-time dynamics, not only the end points are required to move but all the pixels in between the moved end points have to be scan converted with appropriate algorithms which might slow down the dynamic process. Due to scan conversion, “jaggies” or “stair-casing” are unavoidable.

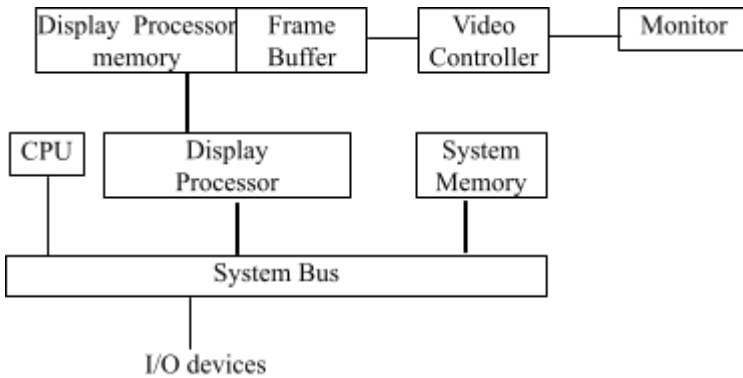


**Figure 1.15:** Architecture of a simple raster graphics system.



**Figure 1.14:** Architecture of RS with a fixed portion of the system memory reserved for the FB.



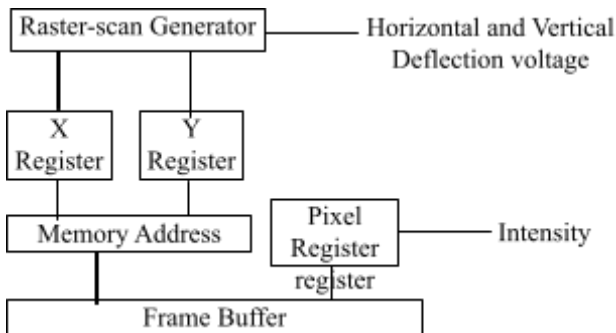


**Figure 1.15:** Architecture of a raster-graphics system with a display processor.

Application program and graphics subroutine package both reside in system memory and execute in CPU. When particular command (e.g., line  $(x_1, y_1, x_2, y_2)$ ) is called by application program, the graphics subroutine package sets the appropriate pixels in the frame buffer. The video controller then cycles through the frame buffer, one scan line at the time (50 fps). It brings a value of each pixel contained in the buffer.

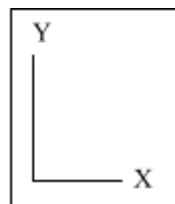
### Video Controller

A fixed area of the system memory is reserved for frame buffer and video controller is given direct access to the frame-buffer memory to refresh the screen.



**Figure 1.16:** Basic video-controller refresh operation.

The screen positions are referenced in Cartesian co-ordinate. Origin is defined as the lower left screen corner. The screen surface is then represented as the first quadrant of 2D system. Two registers are used to store co-ordinates of screen pixel.



**Fig. 1.17:**Screen position

Initially, Y-register is set to  $Y_{\max}$  and X to 0.

The value stored in frame buffer for this pixel is retrieved and used to set intensity of CRT beam. Then X-register is incremented by 1 and same process is repeated for each pixel along scan line as before. After cycling through all pixels along bottom scan-line ( $y = 0$ ), video controller resets register to first position on top scan line and refresh process starts again. Since the refreshing per pixel is slow process, the video controller retrieves the intensity values for a group of adjacent pixels from the frame buffer. This block of pixel intensity is stored in separate registers and used to control the CRT beam intensity for a group of adjacent pixels on the screen. Video controller can retrieve pixel intensities from different memory areas on different refresh cycles.

## Display Processing Unit (DPU)

Display processing unit (DPU) is also called graphics controller or display co-processor. The purpose of display processor is to free the CPU from graphic chores (manipulation). It has its own memory. The major task is digitizing a picture definition given in an application program into a set of pixel intensity values for storage in frame buffer. The digitization process is called scan conversion.

DPU also performs generating various line style (dashed, dotted, solid lines), displaying color areas, and performing various transformation.

1.13 Random Scan System/Architecture/Technology

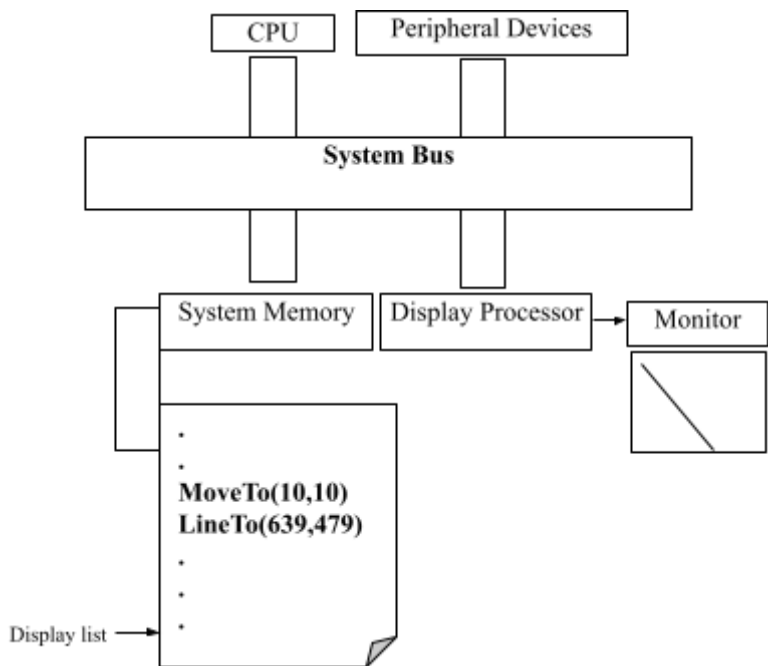


Figure1.18: Architecture of a simple randomscan system

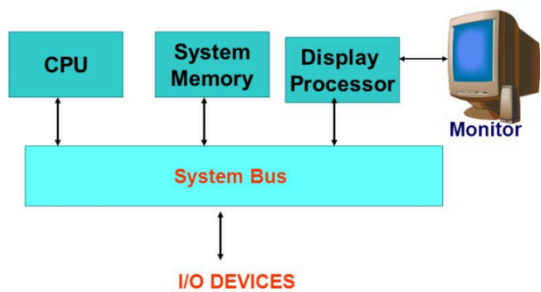


Figure 1.19: Architecture of a simple randomscan system.

Random scan system was developed in 60's and used as common display device until 80's. The system consists of CPU, a display processor (DPU or graphics controller), a CRT monitor, system memory, peripheral devices. An application program and graphics subroutine package both resides in system memory and

execute in CPU. Graphics commands in the application program are translated by graphics package into a display file. Display file or list contains point and line plotting commands with end points coordinates and character plotting commands. The display list or file is then accessed by the display processor to refresh the screen.

DPU interprets commands in display list and plots the respective output primitives like point, line and character. The display processor cycle through each command in the display file program once during every refresh cycle.

DPU sends digital point coordinates to a vector generator that converts digital coordinate values to analog voltage fir circuits that displace an electron beam hitting on CRT's phosphor coating. Beam is deflected from endpoint as dictated by arbitrary order of commands in display list, hence called random scan display. Since, light is decayed tens or hundreds of micro second, DPU must cycle through display list to refresh image around 50 times per second to avoid flicker.

### **Advantages:**

Vector graphics technology produces smooth primitives with higher resolution than raster technology. It is better than raster technology for animation. For transformation, only end point has to be moved.

### **Disadvantages:**

Vector graphics technology can not fill area with patterns manipulate bits. The time required for refreshing an image depends on complexity of the image.

## **1.14 Flat Panel Displays**

Flat panel displays have reduced volume, weight and power requirements compared to CRT. Current uses of flat panel displays are like, in TV monitor, calculator, pocket video games, laptop, and armrest viewing of movies on airlines.

### **Two types:**

- i. Emissive displays

## ii. Non-emissive displays

### 1.14.1 Emissive Displays (Emitter)

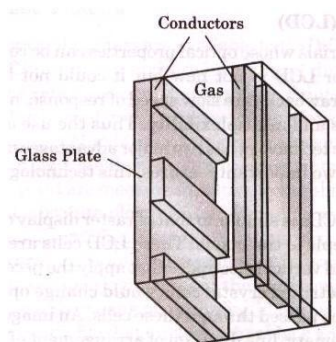
---

Emissive displays convert electrical energy into light. Plasma panel, thin-film electroluminescent displays, and light emitting diodes (LED) are example of emissive displays.

#### Plasma Panel

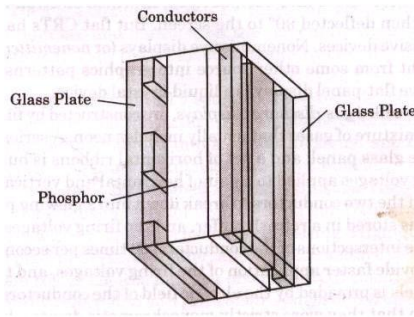
Plasma panel is larger display usually 30 inches (76 cm) or larger. It is called plasma because it uses cell containing electrically charged ionized gasses, known as plasma. So it is also known as gas-discharge display.

Plasma panel is made up of two glass plates. Region between two glasses plates is filled with mixture of gases usually neon. A series of vertical conducting ribbons is placed on one glass plate and horizontal ribbons in another. Firing voltage is applied to a pair of horizontal and vertical conductors cause the gas at the intersection of two conductors to break down into glowing plasma of electrons and ions. The temperature of plasma can reach to  $1200^{\circ}\text{C}$ . Picture definition is stored to refresh buffer and firing voltage is applied to refresh the pixel position 60 times per second. Separation between pixels is provided by electric field of conductors. Plasma panel is power consuming around 400-500 watt.



## Thin-Film Electroluminescent Displays

Thin film electroluminescent displays are similar to a plasma panel but difference is it is filled with phosphor, such as zinc sulfide doped with manganese instead of gas. It requires more power than plasma panel. When sufficiently high voltage is applied to a pair of crossing electrodes, the phosphor becomes a conductor in the area of the intersection of the two electrodes. Electrical energy is then absorbed by manganese atoms which release the energy as spot of light to glowing plasma affect in plasma panel.



## LED

In LED, a matrix of diodes is arranged to form the pixel positions in the display and picture definition is stored in a refresh buffer. Information is read from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.

New developed LEDs are:

- OLED (Organic LED)
- AMOLED (Active Matrix Organics LED)
- MicroLED
- QLED (Quantum LED)

### 1.14.2 Non-Emissive Displays (Non-Emitter)

It uses optical effects to convert sun (natural) light or light from some other source into graphic pattern.

Liquid crystal device (LCD) is the example of non - emissive flat - panel display.

### **LCD (Liquid Crystal Display)**

LCD is based on phenomenon of blocking or transimission of light than light emitting. It needs backlight. It can't emit light. These non-emissive devices produce a picture by passing polarized light from the surroundings or from an internal light source through a liquid crystal material that can be aligned to either block or transmit the light.

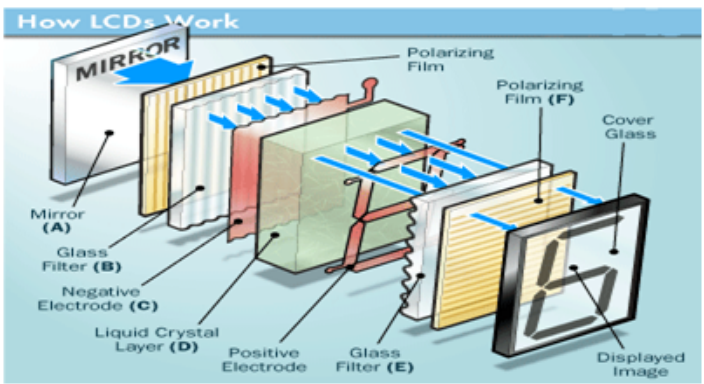
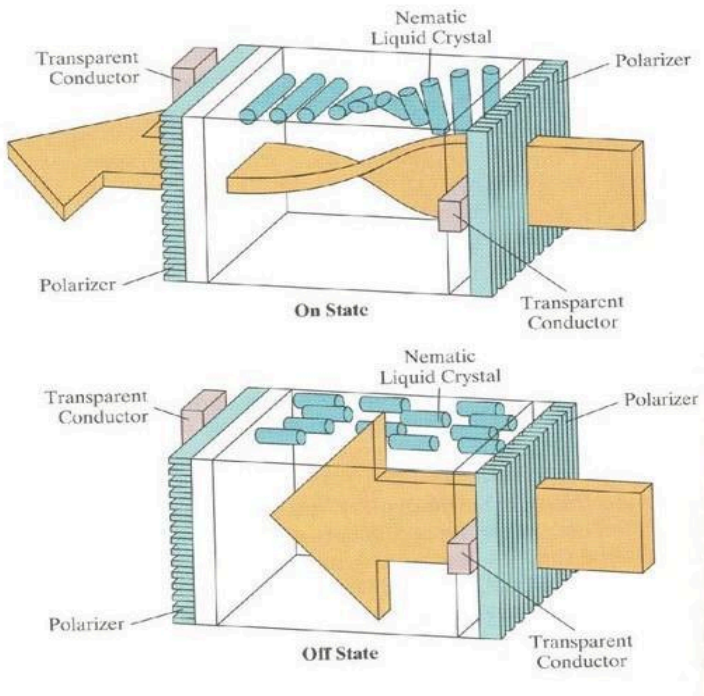
LCD includes mirror, vertical polarizing film, glass filter, negative electrode, liquid crystal, positive electrode, glass filter, horizontal polarizing film and cover glass. More precisely, LCD can be analysed as glass plate with vertical polarizer, horizontal transparent conductor, nematic (threadlike) liquid crystal, glass plate with horizontal polarizer, and vertical transparent conductor.

The light from the surrounding or internal source passes through polarizing film. Then it passes through conductor and then paases to liquid crystal. In on state, polarised light passing through the material is twisted so that it can pass through the opposite ploariser. Then the light paases through another polarizing film and transparent conductor. At last the light is then reflected back to viewer. To turn off the pixel, we apply a voltage to the two intersecting conductors to align the molecules so that light is not twisted. This type of LCD is called passive - matrix LCD. In active - matrix LCD a transistor is placed at each pixel position, used to control the voltage at pixel position.

LCD uses the light modulating properties of liquid crystals. Liquid crystals do not emit light directly, instead usesbacklight, or reflector to produce images. Liquid crystals refer to compounds which are in crystalline arrangement but can flow like liquid.

Two glass plates each containing a light polarizer at right angle to the other, sandwich a liquid crystal material. Rows of horizontal transparent conductors are built into one glass plate. Columns of vertcal conductors are put in other glass plate. The intersection of two conductors defines the pixel position.

LCD is commonly used in small systems, such as calculators, and portable, laptop computers.





- 
- 
- 
1. *If pixels are accessed from the frame buffer with an average access time 300ns. Then will this rate produce the flickering effects? (Screen resolution = 640×480)*

**Solution:**

Access time for 1 pixel = 300ns

Access time for 640×480 pixels = 640×480×300ns

Frequency = =

= 10.85 frame per second (fps)

This value is lesser than 50fps, so flicker occurs

2. *If the total intensity available for a pixel is 256 and the screen resolution is 640×480. What will be the size of the frame buffer?*

**Solution:**

To get 256 simultaneous colours, each pixel would have 8 bits

Because,  $256=2^8$

Size of frame buffer for 1 pixel = 8 bit

In general, total memory bits required =  $R_x * R_y * 8$ , where  $R_x$  and  $R_y$  are the resolutions in the x, y-directions, respectively.

For 640×480 pixels, size of frame buffer

= 640×480×8 bits

= 300KB

The size of the frame buffer is 300 KiloByte

3. *Consider 256 pixel × 256 scan lines image with 24-bit true color. If 10 minutes video is required to capture, calculate the total memory required?*

**Solution:**

Memory required for 1 sec = 256×256×3×50 Bytes

For 10 minutes, total memory required

$$= 5.49 \text{ GB}$$

The total memory required is 5.49 GB

4. *If we want to resize at 1024×768 image to one that is 640 pixels wide with the same aspect ratio, what would be the height of the resized image?* [2070 Ashadh]

**Solution:**

Aspect ratio =

Even after the image is resized, the aspect ratio remains same. So,

$$=$$

$$\therefore H = 480$$

The height of the resized image is 480

5. *How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution 1024×768 and refresh rate 60 frames per second?* [2070 Chaitra]

**Solution:**

Resolution = 1024×768

Refresh rate = 60fps

For 60 frames, it takes 1 second to scan

For 1 frame, it takes second

1 frame means total 1024×768 pixels

For 1024×768 pixels, it takes = 0.016667 second

For scanning 1 pixel, it takes second

For scanning 1 row of pixels i.e., 1024 pixels, it takes×1024  
= 0.0000217013 second

Time spent scanning across each row of pixels is  
0.0000217013 second.

6. *Consider a raster scan system having 12 inch by 10 inch screen with a resolution of 100 pixels per inch in each direction. If the display controller of this system refreshes the screen at the rate of 50 frames per second, how many*

***pixels could be accessed per second and what is the access time per pixel of the system?*** [2071 Shrawan]

***Solution:***

Total pixels =  $12 \times 100 \times 10 \times 100$

Refresh rate = 50 frames per second.

Pixels accessed per second (f) =  $12 \times 100 \times 10 \times 100 \times 50$   
= 60000000

Access time per pixel ==  $1.667 \times 10^{-8}$  second

Access time per pixel is  $1.667 \times 10^{-8}$  second

7. ***Calculate the frame buffer size (in KB) for a raster system recording a video for 1 min with resolution of  $1280 \times 1024$ , and storing 24 bits per pixel with a refresh rate of 25 fps.***

[2076 Ashwin]

***Solution:***

Screen resolution =  $1280 \times 1024$

Refresh rate = 25 fps

Bit required to represent a pixel = 24 bits

Memory required just for a frame =  $1280 \times 1024 \times 24$  bits

Memory required for 1 second =  $1280 \times 1024 \times 24 \times 25$  bits

Memory required for recording a video for 1 min is

=  $1280 \times 1024 \times 25 \times 24 \times 60$  bits = 5760,000 KB

8. ***Calculate the total memory required to store a 10 minutes video in a SVGA system with 24 bit true color and 60 fps refresh rate.***

[2078 Kartik]

***Solution:***

The resolution of SVGA system =  $800 \times 600$

Bit required to represent a pixel = 24 bit

Refresh rate = 60 fps

Memory required for one frame =  $800 \times 600 \times 24$

Memory required for 1 second =  $800 \times 600 \times 24 \times 60$  bit

Memory required for recording a video for 10 min is

=  $800 \times 600 \times 24 \times 60 \times 10 \times 60$  bit = 50,625,000 KB

7. *Let the resolution of screen is  $1024 \times 512$ . What is the memory captured by the frame buffer that uses primary color for display?* [2078 Chaitra]

***Solution:***

The resolution of screen =  $1024 \times 512$

For primary color to display, let the color depth = 24 bit

Size of frame buffer for 1 pixel = 24 bit

For  $1024 \times 512$  pixels, size of frame buffer

=  $1024 \times 512 \times 24$  bit

= 4 MB

The memory captured by the frame buffer is 4 MB

---



# Scan Conversion

---

## 2.1 Output Primitives

---

The basic geometric structures such as points, straight line segments, circles, conical sections, quadric surfaces, spline curves and surfaces, polygon color areas are called output primitives. These are used to describe a scène.

The graphic programming packages describe the scene in terms of output primitives and group sets of output primitives into more complex structures.

Scan conversion is the process of representing continuous graphics object as a collection of discrete pixels. Scan conversion is the process of converting vector-based or geometric primitives, such as lines, curves, or polygons, into a rasterized format suitable for display on a pixel-based output device, such as a computer monitor or printer.

The process typically involves algorithms that calculate the coordinates of the pixels that fall along the path of a primitive shape and then set the corresponding pixels in the frame buffer to the desired color or intensity.

Scan conversion of a point means determining which pixels on a display or grid should be activated to represent that point. Scan conversion of a line involves determining which pixels on a display or grid should be activated to represent that line accurately.

The specific scan conversion algorithms used depend on the type of primitive being processed. For example, the Bresenham's line algorithm is commonly used to rasterize lines efficiently, while algorithms like the midpoint circle algorithm are used for rasterizing circles or curves.

## 2.2 Line - Drawing Algorithm

### Direct method of line drawing

A straight line can be represented in a slope - intercept equation as:

$$y = mx + b \dots \dots \dots (1)$$

Where,

$m$  is slope of the line and

$b$  is y-intercept

A line is drawn by joining two end points,

For any two given points  $(x_1, y_1)$  and  $(x_2, y_2)$ ,

Slope can be calculated by,

$$m = \dots \dots \dots (2)$$

$$b = y_1 - m x_1 \dots \dots \dots (3)$$

At any point  $(x_k, y_k)$

$$y_k = mx_k + b$$

At any point  $(x_{k+1}, y_{k+1})$

$$y_{k+1} = mx_{k+1} + b \dots \dots \dots (4)$$

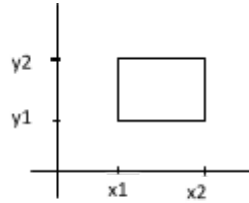
When the value of  $m$  is calculated using equation 2, the following test cases can be performed as,

**Case I:** For lines with slope magnitude  $|m| \leq 1$ , we set small increment in  $x$  axis ( $\Delta x$ ) equal to 1 and calculate corresponding  $y$  increment with the help of equation no. 4. i.e.,

$$x_{\text{next}} = x_{\text{previous}} + 1$$

$$y_{\text{next}} = mx_{\text{next}} + b$$

**Case II:** For lines with slope magnitude  $|m| > 1$ , we set small increment in  $y$  axis ( $\Delta y$ ) equal to 1 and calculate corresponding  $x$  increment with the help of equation no. 4.



**Fig. 2.1:** Line path between two endpoint positions

$$y_{\text{next}} = y_{\text{previous}} + 1$$

$$x_{\text{next}} =$$

On raster systems, we must sample a line at discrete positions and determine the nearest pixel to the line at each sampled position.

**Q) Digitize the line joining the points (1, 2) and (5, 3) with direct method.**

Hints;

$$m =$$

$$b = y_1 - m x_1 =$$

here,  $m < 1$

So,

$$x_{\text{next}} = x_{\text{previous}} + 1$$

$$y_{\text{next}} = mx_{\text{next}} + b$$

x	y = mx+b	Plot pixel (x, y)
1	2	(1, 2)
2	2.25	(2, 2)
3	2.5	(3, 3)
4	2.75	(4, 3)
5	3	(5, 3)

**Q) Digitize the line joining the points (3, 4) and (9, 5) with direct method.**

Hints,

$$m =$$

$$b = y_1 - m x_1 =$$

here,  $m < 1$

so,

$$x_{\text{next}} = x_{\text{previous}} + 1$$

$$y_{\text{next}} = mx_{\text{next}} + b$$

X	y = mx + b	Plotpixel(x, y)
3	4	(3,4)
4	4.17	(4,4)
5	4.33	(5,4)
6	4.50	(6,5)
7	4.66	(7,5)
8	4.83	(8,5)
9	5	(9,5)

### 2.2.1 Digital Differential Analyzer Algorithm

Digital Differential Analyzer Algorithm is based on calculating either x interval ( $\Delta x$ ) or y interval  $\Delta y$ . The sampling of line is done at unit interval in one coordinate and corresponding value nearest to the line path is determined in another coordinate.

The equation of line is

$$y = mx + b$$

And, slope is calculated by,  $m = \frac{\Delta y}{\Delta x}$

At any point  $(x_k, y_k)$

$$y_k = mx_k + b \dots (A)$$

At point  $(x_{k+1}, y_{k+1})$

$$y_{k+1} = mx_{k+1} + b \dots (B)$$

From equation (A) and (B), we get,

$$y_{k+1} - y_k = m(x_{k+1} - x_k)$$

$(y_{k+1} - y_k)$  is small increment in y and  $(x_{k+1} - x_k)$  is correspondingly small increment in x.

$$y_{k+1} - y_k = m(x_{k+1} - x_k)$$

$$m = (y_{k+1} - y_k) / (x_{k+1} - x_k)$$

$$m = \Delta y / \Delta x$$



For given x interval ( $\Delta x$ ) along a line, we can compute the corresponding y interval ( $\Delta y$ ) from the following equation.

$$\Delta y = m \Delta x$$

Similarly, we can obtain the x interval  $\Delta x$  corresponding to specified  $\Delta y$  as

$$\Delta x =$$

Digital differential analyzer (DDA) algorithm is scan conversion line algorithm based on calculating either  $\Delta x$  or  $\Delta y$  using the relation.

$$\Delta y = m \Delta x \dots\dots\dots (i)$$

$$\Delta x = \dots\dots\dots (ii)$$

We sample the line at unit interval in one co-ordinate and determine corresponding integer values nearest the path for the other co-ordinate.

#### **Different cases:**

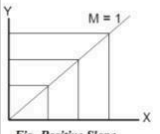
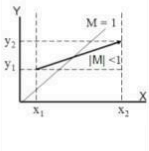
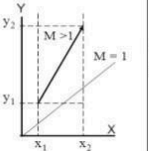
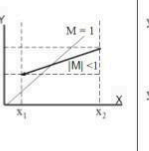
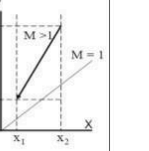
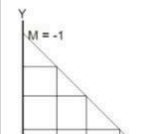
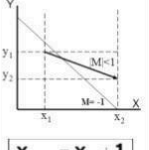
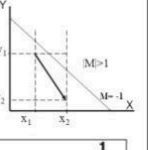
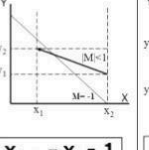
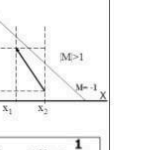
**Case 1:** Line with positive slope and magnitude less than or equal to 1 ( $|m| \leq 1$ )

**Case 2:** Line with positive slope and magnitude more than 1 ( $|m| > 1$ )

**Case 3:** Line with negative slope and magnitude less than or equal to 1 ( $|m| \leq 1$ )

**Case 4:** Line with negative slope and magnitude more than 1 ( $|m| > 1$ )

The possible combinations can be shown as follows:

	Moving Left to Right		Moving Right to Left	
	Slope (m) < 1	Slope (m) > 1	Slope (m) < 1	Slope (m) > 1
<b>Positive Slope</b>  <i>Fig. Positive Slope</i>	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k + 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k - 1$
<b>Negative Slope</b>  <i>Fig. Negative Slope</i>	 $x_{k+1} = x_k + 1$ $y_{k+1} = y_k - m$	 $x_{k+1} = x_k + \frac{1}{m}$ $y_{k+1} = y_k - 1$	 $x_{k+1} = x_k - 1$ $y_{k+1} = y_k + m$	 $x_{k+1} = x_k - \frac{1}{m}$ $y_{k+1} = y_k + 1$

## 1. Line with a positive slope

**Case I:** If slope is positive with magnitude lesser than or equal to 1,  $|m| \leq 1$ , then we sample at unit x intervals ( $\Delta x = 1$ ) and compute each successive y value because the increment in x is more than increment in y.

So, set  $\Delta x = 1$ .

Now, from equation (1), we get

$$\Delta y = m$$

$$\text{i.e., } x_{k+1} = x_k + \Delta x = x_k + 1 \dots \dots \dots (3)$$

$$y_{k+1} = y_k + \Delta y = y_k + m \dots \dots \dots (4)$$

where subscript k takes integer values starting from 1, for the first point, and increases by 1 unit until the final end point is reached. Since m can be any real number between 0 to 1, the calculated y values must be rounded to the nearest integer.

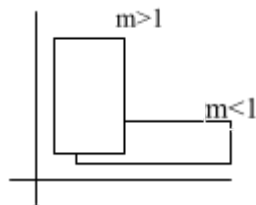
**Case II:** If slope is positive with magnitude greater than 1,  $|m| > 1$ , then the increment in x ( $\Delta x$ ) is smaller than increment in y ( $\Delta y$ ). So, we sample at unit y intervals ( $\Delta y = 1$ ) and calculate each succeeding x value as,

$$\Delta x =$$

That is,

$$x_{k+1} = x_k + \dots\dots\dots(5)$$

$$y_{k+1} = y_k + 1 \dots\dots\dots(6)$$



**Fig. 2.2:** Line with positive slope  $|m| < 1$  and  $|m| > 1$

Here, in the both case  $m \leq 1$  and  $m > 1$ , we consider the algorithm (i.e., equations 3, 4, 5, 6) based on the assumption that **lines are to be processed from the left end point to the right end point**.

If this process is reversed (that is, lines are to be processed from **right to left**), relation is required to be changed in both case.

**Case I:** If slope is positive with magnitude lesser than or equal to 1,  $|m| \leq 1$ , then  $\Delta x = -1$ , so,  $\Delta y = -m$

$$\text{i.e., } x_{k+1} = x_k - 1$$

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

**Case II:** If slope is positive with magnitude greater than 1,  $|m| > 1$ , then  $\Delta y = -1$  and  $\Delta x = -$

$$\text{i.e., } x_{k+1} = x_k - \dots\dots\dots(7)$$

$$y_{k+1} = y_k - 1 \dots\dots\dots(8)$$

## 2. Line with negative slope

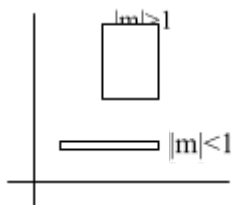
**Case I:** If slope is negative with magnitude lesser than or equal to 1,  $|m| \leq 1$ , then assume start end point is the left.

$$\Delta x = 1 \text{ and } \Delta y = m \text{ (m is negative)}$$

That is,

$$x_{k+1} = x_k + 1 \dots\dots\dots(9)$$

$$y_{k+1} = y_k + m \dots\dots\dots(10)$$



**Fig 2.3:** line with negative slope  $|m| < 1$  and  $|m| > 1$

If algorithm is required to proceed **right to left**, then set on

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

$$\text{i.e., } x_{k+1} = x_k - 1 \dots\dots\dots(11)$$

$$y_{k+1} = y_k - m \dots\dots\dots (12)$$

**Case II:** If slope is negative with magnitude greater than 1,  $|m| > 1$ , then assume start end is at left and set  $\Delta y = -1$  and  $\Delta x = -$

$$\text{i.e., } x_{k+1} = x_k - \dots\dots\dots (13)$$

$$y_{k+1} = y_k - 1 \dots\dots\dots (14)$$

If the algorithm is required to proceed **right to left**, then set  $\Delta y = 1$  and  $\Delta x = .$

$$\text{i.e., } x_{k+1} = x_k + \dots\dots\dots (15)$$

$$y_{k+1} = y_k + 1 \dots\dots\dots (16)$$

## DDA Algorithm

Step 1: Start Algorithm

Step 2: Declare  $x_1, y_1, x_2, y_2, dx$ , step as integer variable and  $x, y, x_{inc}, y_{inc}$  as floating point

Step 3: Read the values of two end points  $x_1, y_1, x_2, y_2$

Step 4: Calculate  $dx = x_2 - x_1$

Step 5: Calculate  $dy = y_2 - y_1$ .

Step 6: Calaulate the value of step

If absolute  $(dx) > \text{absolute } (dy)$

Then step = absolute  $(dx)$

Else step = absolute  $(dy)$

Step 7: Select the raster unit

$$x_{inc} =$$

$$y_{inc} =$$

$$\text{Assign } x = x_1$$

$$\text{Assign } y = y_1$$

Step 8: Set pixel  $(x, y)$

Step 9:  $x = x + x_{inc}$

$$y = y + y_{inc}$$

Plot pixels  $(\text{Round } (x), \text{Round } (y))$

Step 10: Repeat step 9 until  $x = x_2$

Step 11: End Algorithm

**Advantages:**

- It is faster method than direct line drawing equation  $y = mx + c$  for calculating pixel position as it eliminates multiplication.
- It is simple to understand, and it doesn't require special knowledge to implement.

**Disadvantages:**

- The floating point addition is still needed in determining each successive point which is time consuming. The value of slope 'm' is usually stored in floating point number. So, there could be round off error.
- The line will move away from the true line path, especially when it is long due to successive round of error.
- Accumulation of round off error, in successive additions of floating point increment.

Scan converting a line by different line drawing method		
Direct Method	Simple DDA	Incremental DDA
$y = mx + b$ $m =$  $\text{if } m \leq 1$ $x_{\text{next}} = x_{\text{previous}} + 1$  $y_k = mx_k + b$ $\text{else}$ $y_{\text{next}} = y_{\text{previous}} + 1$ $x_k = (y_k - b)/m$	$m =$  $\text{if } m \leq 1$ $x_{\text{next}} = x_{\text{previous}} + 1$ $y_{\text{next}} = y_{\text{previous}} + m$  $\text{else}$ $y_{\text{next}} = y_{\text{previous}} + 1$ $x_{\text{next}} = x_{\text{previous}} + (1/m)$	$m =$  $\text{if } m \leq 1 (\text{i.e. absolute(dy)} < \text{absolute(dx)})$ $\text{step} = \text{absolute(dx)}$ $\text{else}$ $\text{step} = \text{absolute(dy)}$ $x_{\text{inc}} = \text{dx}/\text{step}$ $y_{\text{inc}} = \text{dy}/\text{step}$ $x_{\text{next}} = x_{\text{previous}} + x_{\text{inc}}$ $y_{\text{next}} = y_{\text{previous}} + y_{\text{inc}}$

**Q. Consider a line from (2, 1) to (8, 3). Using simple DDA algorithm, rasterize this line.**

**Solution,**

Given,

Starting Point :  $(x_1, y_1) = (2, 1)$

Ending Point:  $(x_2, y_2) = (8, 3)$

Now,

Slope  $m = (y_2 - y_1) / (x_2 - x_1) = (3 - 1) / (8 - 2) = (1/3) = 0.333$

Since  $|m| < 1$ , From DDA algorithm we have

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

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

X	y	x-Plot	y-Plot	(x,y)
2	1	2	1	(2, 1)
3 (2+1)	1.33 (1+0.33)	3	1	(3, 1)
4 (3+1)	1.66 (1.33+0.33)	4	2	(4, 2)
5	1.993	5	2	(5, 2)
6	2.326	6	2	(6, 2)
7	2.659	7	3	(7, 3)
8	2.999	8	3	(8, 3)

### **1. Code in C to draw a line using DDA Algorithm**

```
#include<stdio.h>
```

```

#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
    int gd = DETECT, gm;
    int x1, y1, x2, y2, stepsize, dx, dy, i;
    float x, y, xinc, yinc;
    initgraph(&gd,&gm,"c:\\tc\\bgi");
    printf("Digital      Differntial      Line      Drawing
    Algorithm\n");
    printf("put the values of x1 and y1\n");
    scanf("%d %d",&x1, &y1);
    printf("put the values of x2 and y2\n");
    scanf("%d %d", &x2, &y2);
    dx = x2 - x1;
    dy = y2 - y1;
    x = x1;
    y = y1;
    if (abs(dy) > abs(dx))
    {
        Stepsize = abs(dy);
    }
    else
    {
        Stepsize = abs(dx);
    }
    xinc = dx/(float)stepsize;
    yinc = dy/(float)stepsize;
    putpixel(x, y, RED);
    for(i = 0; i < stepsize; i++)

```

```

    {
        x = x + xinc;
        y = y + yinc;
        putpixel((int)(x + 0.5), (int)(y + 0.5), RED);
    }
    getch();
    closegraph();
}

```

## **2. Code in C to draw a checkbox using DDA Algorithm**

```

#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<math.h>
void dda (int, int, int, int);
void main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\tc\\bgi");
    dda(100, 100, 200, 100);
    dda(200, 100, 200, 200);
    dda(200, 200, 100, 200);
    dda(100, 200, 100, 100);
    dda(100, 100, 200, 200);
    dda(100, 200, 200, 100);
    dda(100, 125, 200, 125);
    dda(100, 150, 200, 150);
    dda(100, 175, 200, 175);
    dda(175, 100, 175, 200);
    dda(150, 100, 150, 200);
}

```



```

        dda(125, 100, 125, 200);
        dda(100, 150, 150, 100);
        dda(150, 100, 200, 150);
        dda(200, 150, 150, 200);
        dda(150, 200, 100, 150);
        getch();
        closegraph();
    }
void dda(int x1, int y1, int x2, int y2)
{
    int i, stepsize, dx, dy;
    float x, y, xinc, yinc;
    dx = x2 - x1;
    dy = y2 - y1;
    x = x1;
    y = y1;
    if (abs(dy) > abs(dx))
    {
        stepsize = abs(dy);
    }
    else
    {
        stepsize = abs(dx);
    }
    xinc = dx/(float) stepsize;
    yinc = dy/(float) stepsize;
    putpixel(x, y, RED);
    for(i = 0; i < stepsize; i++)
    {
        x = x + xinc;

```

```

    y = y + yinc;
    putpixel((int)(x + 0.5), (int)(y + 0.5), RED);
    delay(10);
}
}

```

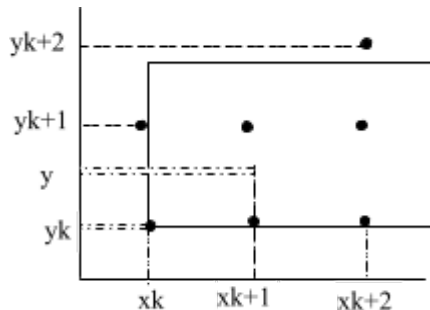
### 2.2.2 Bresenham's Line Drawing Algorithm

---

Bresenham's line drawing algorithm is an accurate and efficient line drawing algorithm. It uses only integer arithmetic to find the next position to be plotted. It avoids incremental error. The major concept of Bresenham's algorithm is to determine the nearest pixel position. Great advantage of this algorithm is that it can be used to display circles and other curves.

In Bresenham's algorithm, we calculate the decision parameter which decides which pixel to select and which function is used for next decision parameter. Here, we derive the Bresenham's Line Algorithm for different type of lines.

#### 1. Line with positive slope and magnitude $|m| < 1$



**Figure 2.4:** Line with  $m < 1$

- Pixel positions are determined by sampling at unit x intervals.

- Starting from left end position  $(x_0, y_0)$  of a given line, we step to each successive column (x-position) and plot the pixel whose scan line y value is closed to the line path.

Assuming the pixel at  $(x_k, y_k)$  to be displayed is determined, we next to decide which pixel to plot in column  $x_{k+1}$ , our choices are the pixels at positions.

$(x_k + 1, y_k)$  and  $(x_k + 1, y_k + 1)$

At sampling position  $x_k + 1$ , we label vertical pixel separations from the mathematical line path  $d_1$  and  $d_2$ .

The y-co-ordinate on the mathematical at pixel column position  $x_k + 1$  is calculation.

As,

$$y = m(x_k + 1) + b \dots \dots (1)$$

Then,

$$d_1 = y - y_k$$

$$d_1 = m(x_k + 1) + b - y_k \dots \dots (2)$$

And,

$$d_2 = (y_k + 1) - y$$

$$d_2 = y_k + 1 - m(x_k + 1) - b \dots \dots (3)$$

Now,

$$\begin{aligned} d_1 - d_2 &= m(x_k + 1) + b - y_k - y_k - 1 + m(x_k + 1) + b \\ &= 2m(x_k + 1) - 2y_k + 2b - 1 \\ &= 2(x_k + 1) - 2y_k + 2b - 1 \end{aligned}$$

Defining decision parameter  $p_k = \Delta x(d_1 - d_2)$

$$\begin{aligned} p_k &= \Delta x(d_1 - d_2) = 2\Delta y(x_k + 1) - 2\Delta x y_k + 2\Delta x b - \Delta x \\ &= 2\Delta y x_k + 2\Delta y - 2\Delta x y_k + \Delta x(2b - 1) \\ &= 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2b - 1) \\ &= 2\Delta y x_k - 2\Delta x y_k + c \dots \dots (4) \end{aligned}$$

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

Here, sign of  $p_k$  is same as the sign of  $d_1 - d_2$  since  $\Delta x > 0$ .

**Case I:**

If  $p_k < 0$  then  $d_1 < d_2$  which implies pixel at  $y_k$  is nearer than pixel at  $(y_k + 1)$ . So, pixel at  $y_k$  is better to choose which reduce error than pixel at  $(y_k + 1)$ . This determines next pixel co-ordinate to plot is  $(x_k + 1, y_k)$

**Case II:**

If  $p_k \geq 0$ , then  $d_2 < d_1$ , which implies that  $y_k + 1$  is nearer than  $y_k$ . So, pixel at  $(y_k + 1)$  is better to choose which reduce error than pixel at  $y_k$ . This determines next pixel co-ordinate to plot is  $(x_k + 1, y_k + 1)$ .

Now, similarly, pixel as  $(x_k + 2)$  can be determined whether it is  $(x_k + 2, y_k + 1)$  or  $(x_k + 2, y_k + 2)$  by looking the sign of deciding parameter  $p_k + 1$  assuming pixel as  $(x_k + 1)$  is known.

$p_{k+1} = 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c$  where  $c$  is same as in  $p_k$

Now,

$$\begin{aligned} p_{k+1} - p_k &= 2\Delta y x_{k+1} - 2\Delta x y_{k+1} + c - (2\Delta y x_k - 2\Delta x y_k + c) \\ &= 2\Delta y (x_{k+1} - x_k) - 2\Delta x (y_{k+1} - y_k) \text{ where } x_{k+1} = x_k + 1 \\ &= 2\Delta y (x_k + 1 - x_k) - 2\Delta x (y_{k+1} - y_k) \\ &= 2\Delta y - 2\Delta x (y_{k+1} - y_k) \end{aligned}$$

This implies that decision parameter for the current column can be determined if the decision parameter of the last column is known.

Here,  $(y_{k+1} - y_k)$  could either 0 or 1 which depends on sign of  $p_k$ .

**Case I:**

If  $p_k < 0$  (i.e.,  $d_1 < d_2$ ),

$y_{k+1} = y_k$  which implies  $(y_{k+1} - y_k = 0)$

i.e., at  $p_k < 0$ , then pixel to be plotted is  $(x_k + 1, y_k)$

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

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

and,

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

## Case II:

If  $p_k \geq 0$  (i.e.,  $d_2 < d_1$ ),  $y_{k+1} = y_k + 1$  which implies  $(y_k + 1 - y_k) = 1$

That is, at  $p_k \geq 0$ ,

the pixel to plot is  $(x_k + 1, y_k + 1)$

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

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

and,

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

## Initial decision parameter ( $p_0$ )

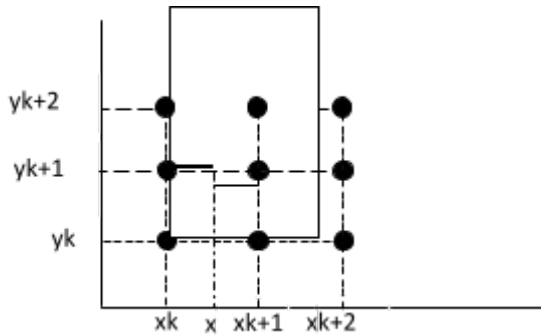
$$\text{Here, } p_k = 2\Delta y x_k - 2\Delta x y_k + 2\Delta y + \Delta x(2b-1)$$

$$p_0 = 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + \Delta x(2b-1)$$

$$\text{But } b = y_0 - mx_0 = y_0 - x_0$$

$$\begin{aligned} p_0 &= 2\Delta y x_0 - 2\Delta x y_0 + 2\Delta y + 2\Delta x y_0 - 2\Delta y x_0 - \Delta x \\ &= 2\Delta y - \Delta x \end{aligned}$$

## 2. Line with positive slope and magnitude $|m| > 1$



**Figure 2.5:** Line with  $m > 1$

- Pixel positions are determined by sampling at unit  $y$  intervals.

- Starting from left end position  $(x_0, y_0)$  of a given line, we step to each successive row (y-position) and plot the pixel whose scan line x value is closed to the line path.

Assuming the pixel at  $(x_k, y_k)$  to be displayed is determined, we next to decide which pixel to plot in row  $y_{k+1}$ , our choices are the pixels at positions.

$(x_k, y_k + 1)$  and  $(x_k + 1, y_k + 1)$

At sampling position  $y_k + 1$ , we label horizontal pixel separations from the mathematical line path  $d_1$  and  $d_2$ .

The x-co-ordinate on the mathematical at pixel row position  $y_k + 1$  is calculation.

As,

$$y_k + 1 = mx + b$$

$$x = (y_k + 1 - b)/m \dots \dots \dots (1)$$

Then,

$$d_1 = x - x_k \dots \dots \dots (2)$$

And,

$$d_2 = (x_k + 1) - x \dots \dots \dots (3)$$

Now,

$$d_1 - d_2 = x - x_k - x_k - 1 + x$$

$$= 2x - 2x_k - 1$$

$$x = (y_k + 1 - b)/m$$

$$d_1 - d_2 = 2(y_k + 1 - b)/m - 2x_k - 1$$

$$= 2\Delta x (y_k + 1 - b)/\Delta y - 2x_k - 1$$

$$\Delta y (d_1 - d_2) = 2\Delta x y_k + 2\Delta x - 2\Delta x b - 2\Delta y x_k - \Delta y$$

Defining decision parameter  $p_k = \Delta y (d_1 - d_2)$

$$P_k = \Delta y (d_1 - d_2) = 2\Delta x y_k + 2\Delta x - 2\Delta x b - 2\Delta y x_k - \Delta y$$

$$= 2\Delta x y_k - 2\Delta y x_k + c \dots \dots \dots (4)$$

Where,  $c = 2\Delta x - 2\Delta x b - \Delta y$

Here, sign of  $p_k$  is same as the sign of  $d_1 - d_2$  since  $\Delta y > 0$ .

**Case I:**

If  $p_k < 0$  then  $d_1 < d_2$  which implies pixel at  $x_k$  is nearer than pixel at  $(x_k + 1)$ . So, pixel at  $x_k$  is better to choose which reduce error than pixel at  $(x_k + 1)$ . This determines next pixel co-ordinate to plot is  $(x_k, y_k + 1)$

**Case II:**

If  $p_k \geq 0$ , then  $d_1 > d_2$ , which implies that  $x_k + 1$  is nearer than  $x_k$ . So, pixel at  $(x_k + 1)$  is better to choose which reduce error than pixel at  $x_k$ . This determines next pixel co-ordinate to plot is  $(x_k + 1, y_k + 1)$ .

Now, similarly, pixel as  $(y_k + 2)$  can be determined whether it is  $(x_k + 1, y_k + 2)$  or  $(x_k + 2, y_k + 2)$  by looking the sign of deciding parameter  $p_{k+1}$  assuming pixel as  $(y_k + 1)$  is known.

$P_{k+1} = 2\Delta x y_{k+1} - 2\Delta y x_{k+1} + c$  where  $c$  is same as in  $p_k$

Now,

$$\begin{aligned} p_{k+1} - p_k &= 2\Delta x y_{k+1} - 2\Delta y x_{k+1} + c - 2\Delta x y_k + 2\Delta y x_k - c \\ &= 2\Delta x(y_{k+1} - y_k) - 2\Delta y(x_{k+1} - x_k) \\ &= 2\Delta x(y_{k+1} - y_k) - 2\Delta y(x_{k+1} - x_k) \\ &= 2\Delta x(y_{k+1} - y_k) - 2\Delta y(x_{k+1} - x_k) \\ &= 2\Delta x - 2\Delta y(x_{k+1} - x_k) \end{aligned}$$

This implies that decision parameter for the current row can be determined if the decision parameter of the last row is known.

Here,  $(x_{k+1} - x_k)$  could either 0 or 1 which depends on sign of  $p_k$ .

**Case I:**

If  $p_k < 0$  (i.e.,  $d_1 < d_2$ ), then  $x_{k+1} = x_k$  which implies  $(x_{k+1} - x_k = 0)$

i.e., at  $p_k < 0$ , then pixel to be plotted is  $(x_k, y_k + 1)$

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

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

and,

$$p_{k+1} = p_k + 2\Delta x$$

### Case II:

If  $p_k \geq 0$  (i.e.,  $d_1 > d_2$ ), then  $x_{k+1} = x_k + 1$  which implies  $(x_k + 1 - x_k) = 1$

That is, at  $p_k \geq 0$ , the pixel to plot is  $(x_k + 1, y_k + 1)$

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

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

and,

$$p_{k+1} = p_k + 2\Delta x - 2\Delta y$$

### Initial decision parameter ( $p_0$ )

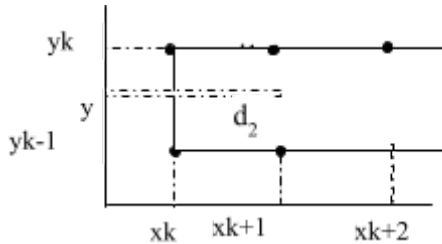
$$P_k = 2\Delta x y_k + 2\Delta x b - 2\Delta y x_k - \Delta y$$

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

$$b = y_0 - mx_0$$

$$\begin{aligned} P_0 &= 2\Delta x y_0 + 2\Delta x - 2\Delta x(y_0 - \Delta y x_0/\Delta x) - 2\Delta y x_0 - \Delta y \\ &= 2\Delta x y_0 + 2\Delta x - 2\Delta x y_0 + 2\Delta x \times \Delta y x_0/\Delta x - 2\Delta y x_0 - \Delta y \\ &= 2\Delta x - \Delta y \end{aligned}$$

### 3. Line with negative slope and magnitude $|m| < 1$ .



$$d_1 = y_k - y$$

$$d_2 = y - (y_k - 1) = y - y_k + 1$$

$$d_1 - d_2 = y_k - y - y + y_k - 1 = 2y_k - 2y - 1$$

$$\text{Again, } y = m(x_k + 1) + b$$

$$d_1 - d_2 = 2y_k - 2(m(x_k + 1) + b) - 1$$



$m = -$  (∵  $m$  is negative)

$$\begin{aligned} d_1 - d_2 &= 2y_k - 2\left(\frac{-\Delta y}{\Delta x}\right)(x_k+1) - 2b - 1 \\ &= \frac{2y_k \Delta x + 2\Delta y x_k + 2\Delta y - \Delta x(2b) - \Delta x}{\Delta x} \end{aligned}$$

$$p_k = (d_1 - d_2) \Delta x = 2\Delta x y_k + 2\Delta y x_k + 2\Delta y - \Delta x(2b) - \Delta x$$

$$p_k = 2\Delta x y_k + 2\Delta y x_k + c$$

$$\text{where } c = 2\Delta y - \Delta x(2b) - \Delta x$$

For next decision parameter,

$$p_{k+1} = 2\Delta x y_{k+1} + 2\Delta y x_{k+1} + c$$

Now,

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

### Case I:

if  $p_k < 0$ ,  $d_1 < d_2$  then

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

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

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

### Case II:

if  $p_k \geq 0$ ,  $d_1 \geq d_2$  then

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

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

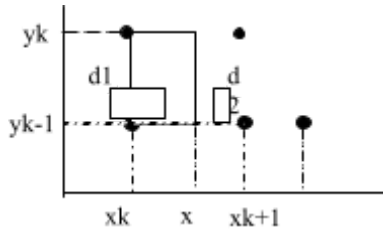
$$, \quad p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

**Now, initial decision parameter,**

$$p_k = p_0, \quad x_k = x_0, \quad y_k = y_0$$

$$\begin{aligned} p_0 &= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x \left( 2 \left( y_0 - \left( \frac{-\Delta y}{\Delta x} \right) x_0 \right) + 1 \right) \\ &= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - \Delta x \left( \frac{2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y}{\Delta x} \right) \\ &= 2\Delta x y_0 + 2\Delta y x_0 + 2\Delta y - 2\Delta x y_0 - 2\Delta y x_0 - \Delta x \\ &= 2\Delta y - \Delta x \end{aligned}$$

#### 4. Line with negative slope and magnitude $|m| > 1$



- Pixel positions are determined by sampling at unity intervals.
- Starting from left end position  $(x_0, y_0)$  of a given line, we step to each successive row (y position) and plot the pixel whose x value is closest to the line path.
- Assuming the pixel at  $(x_k, y_k)$  to be displayed is determined, we next need to decide which pixel to plot in row  $y_k - 1$ .
- The candidate pixels are at position  $(x_k, y_k - 1)$  and  $(x_k + 1, y_k - 1)$
- At sampling position  $y_k - 1$ , we label horizontal pixel separation from the mathematical line path  $d_1$  and  $d_2$ .
- The x-coordinate on the mathematical line at pixel row position  $y_k - 1$  is calculated as

$$y_k - 1 = mx + b$$

$$\text{or, } x = \frac{y_k - 1 - b}{m}$$

where  $m = -\frac{\Delta y}{\Delta x}$  since m is negative.

Then,

$$d_1 = x - x_k \dots\dots\dots (ii) \quad d_2 = x_k + 1 - x \dots\dots\dots (iii)$$

And,

$$d_1 - d_2 = x - x_k - x_k - 1 + x = 2x - 2x_k - 1$$

Now, defining decision parameter  $p_k = \Delta y(d_1 - d_2)$

$$p_k = \Delta y(d_1 - d_2) = -2\Delta \times (y_k - 1 - b) - (2x_k + 1)\Delta y$$

Where  $C = 2\Delta x + 2\Delta x b - \Delta y$  (All are constant)

Here, the sign of  $p_k$  is same as the sign of  $d_1 - d_2$ , as  $\Delta y > 0$

### Case I:

If  $p_k < 0$ , then  $d_1 < d_2$  which implies that  $x_k$  is nearer than  $x_k + 1$ , so the next pixel to select is  $(x_k, y_k - 1)$

### Case II:

If  $p_k \geq 0$ , then  $d_1 > d_2$  which implies that  $x_k + 1$  is nearer than  $x_k$ , so the next pixel to choose is  $(x_k + 1, y_k - 1)$

Now,

Similarly, pixel at  $(y_k - 2)$  can be determined whether it is  $(x_k + 1, y_k - 2)$  or  $(x_k + 2, y_k - 2)$  by looking the sign of deciding parameter  $p_{k+1}$  assuming pixel at  $(y_k - 1)$  is known.

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

where  $C$  is same as in  $p_k$ .

Now,

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

This implies that decision parameter for the current row can be determined if the decision parameter of the last row is known.

Here  $(x_k + 1 - x_k)$  could either 0 or 1 which depends on sign of  $p_k$ .

**Case I:**

If  $p_k < 0$ , then

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

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

$$p_{k+1} = p_k + 2\Delta x$$

**Case II:**

If  $p_k \geq 0$  (i.e.,  $d_1 > d_2$ ) then

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

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

$$p_{k+1} = p_k + 2\Delta x - 2\Delta y$$

**Now, Initial decision parameter**

$$p_k = -2\Delta x y_k - 2\Delta y x_k + 2\Delta x b - 2\Delta x - \Delta y$$

Now,

$$p_0 = -2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x b - 2\Delta x - \Delta y$$

=

$$-2\Delta x y_0 - 2\Delta y x_0 + 2\Delta x \left( y_0 + \frac{\Delta y}{\Delta x} x_0 \right) + 2\Delta x - \Delta y$$

$$\left[ b = y - mx, m = \frac{-\Delta y}{\Delta x} \right]$$

=

2

$$\Delta x y_0 - 2\Delta y x_0 + \Delta x \left( \frac{2\Delta x y_0 + 2\Delta y x_0}{\Delta x} \right) + 2\Delta x - \Delta y$$

=

2

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

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

**BLA Algorithm**

Step 1. Start

Step 2. Declare variables  $x_1, y_1, x_2, y_2, l_x, l_y, \Delta x, \Delta y, p_0, p_k, p_{k+1}$

Step 3. Read values of  $x_1, y_1, x_2, y_2$

Step 4. Calculate  $\Delta x = \text{absolute}(x_2 - x_1)$

$$\Delta y = \text{absolute}(y_2 - y_1)$$

Step 5. If ( $x_2 > x_1$ )

assign  $l_x = 1$

else

assign  $l_x = -1$

Step 6. if ( $y_2 > y_1$ )

assign  $l_y = 1$

else

assign  $l_y = -1$

Step 7. Plot ( $x_1, y_1$ )

Step 8. if  $\Delta x > \Delta y$  (i.e.,  $|m| < 1$ )

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

starting at  $k = 0$  to  $\Delta x$  times, repeat

if ( $p_k < 0$ )

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

$y_{k+1} = y_k$

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

else

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

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

$p_{k+1} = p_k + 2\Delta y - 2\Delta x$

plot( $x_{k+1}, y_{k+1}$ )

else

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

starting at  $k = 0$  to  $\Delta y$  times, repeat

if( $p_k < 0$ )

$x_{k+1} = x_k$

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

$p_{k+1} = p_k + 2\Delta x$

else

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

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

$$p_{k+1} = p_k + 2\Delta x - 2\Delta y$$

$$\text{Plot}(x_{k+1}, y_{k+1})$$

Step 9. Stop

### Advantage of BLA over DDA:

- In DDA algorithm, each successive point is computed in floating point. While in BLA, each successive point is calculated in integer value. So, BLA requires less time and less memory space.
- In DDA, the calculated point value is in floating point number, so, it should be rounded at the end of calculation. But in BLA, round off is not necessary. So, there is no accumulation of rounding error.
- Due to rounding error, the line drawn by DDA algorithm is not accurate, while by BLA algorithm, line is accurate.
- DDA algorithm cannot be used in other application except line drawing, but BLA can be implemented in other application such as circle, ellipse, and other curves.

### Comparison between DDA and BLA

Base of comparison	Digital differential analyzer line drawing algorithm	Bresenham's line drawing algorithm
Arithmetic	DDA algorithm uses floating points	BLA uses integer
Operations	DDA algorithm uses multiplication and division in its operations	BLA uses only subtraction and addition in its operation

Base of comparison	Digital differential analyzer line drawing algorithm	Bresenham's line drawing algorithm
Speed	DDA algorithm is slower than BLA because it uses floating points	BLA is faster than DDA because it uses subtraction, addition and integer only
Accuracy and Efficiency	DDA algorithm is not as accurate and efficient as BLA as error and error accumulation occurs on it	BLA is more efficient and much accurate than DDA algorithm
Drawing	DDA algorithm can't draw curves and circles as accurate as by BLA	BLA can draw curves and circles much more accurately
Round off	DDA algorithm round off the co-ordinates to integer that is nearest to the line	BLA does not round off but takes the incremental value in its operation
Expensive	DDA is expensive as it uses floating point	BLA is cheaper than DDA as it uses only addition and subtraction

### **Code in C to draw a line using BLA**

```

#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
void main()
{
    int gd = DETECT, gm, x1, y1, x2, y2, lx, ly, dy, dx, pk, i;
    initgraph(&gd,&gm,"c:\\tc\\bgi");
    printf("put the values of x1 and y1\n");
    scanf("%d %d",&x1, &y1);

```

```

printf("put the values of x2 and y2\n");
scanf("%d %d",&x2, &y2);
dx = abs(x2 - x1);
dy = abs(y2 - y1);
if(x2 > x1)
{
    Lx = 1;
}
else
{
    Lx = -1;
}
if(y2 > y1)
{
    ly = 1;
}
else
{
    ly = -1;
}
putpixel(x1, y1, RED);
if(dx > dy)
{
    pk=2 * dy - dx;
    for(i=0; i < dx; i++)
    {
        if(pk < 0)
        {
            x1 = x1 + lx;
            y1 = y1;

```



```

        pk = pk + 2 * dy;
    }
else
    {
        x = x1 + lx;
        y1 = y1 + ly;
        pk = pk + 2 * dy - 2 * dx;
    }
    putpixel(x1, y1, RED);
}
else
    {
        pk = 2 * dx - dy;
        for(I = 0; i < dy; i++)
        {
            if(pk < 0)
            {
                x1 = x1;
                y1 = y1 + ly;
                pk = pk + 2 * dx;
            }
            else
            {
                x1 = x1 + lx;
                y1 = y1 + ly;
                pk = pk + 2 * dx - 2 * dy;
            }
            putpixel(x1, y1, RED);
        }
    }
}

```

```

}
getch();
closegraph ();
}

```

## 2.3 Circle

A circle is the set of all points that lie at equal distance from a fixed point. The equal distance is called radius and the fixed point is called center. A circle is a symmetric figure. A circle has 4 way symmetry i.e. symmetry in quadrant and 8 way symmetry i.e. symmetry in octant. If we know the coordinate of a quadrant, then we can draw the other quadrant by reflection about axis in 4 way symmetry. In 8 way symmetry, if we know the octant, then we can plot whole circle by finding the corresponding coordinate just by reflecting with respect to axis and  $45^\circ$ .

The equation of circle in Cartesian form is

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

$$y = y_c \pm$$

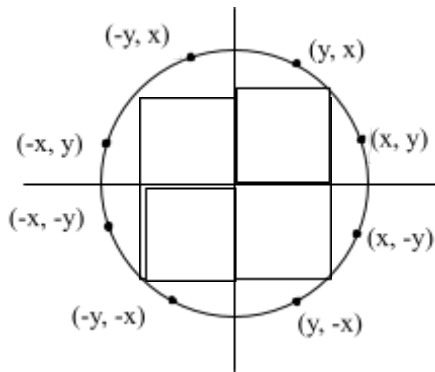
### There are two problems in drawing a circle;

- i. Non-uniform spacing of plotted pixels i.e. large gap may be seen when slope approaches vertical
  - ii. Computational complexity or square root operations problem
- The problem of non-uniform spacing is eliminated by using polar coordinates  $r$  and  $\theta$

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

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

The problem of computational complexity can be solved by using symmetry property in circle.



**Figure 2.6:** Circle with symmetry points

- We sample at unit intervals and determine the closest pixel position to the specified circle at each step.
- For a given radius  $r$  and center  $(x_c, y_c)$ , we first set up our algorithm to calculate pixel positions around a circle path centered at coordinate origin  $(0, 0)$ .
- Then each calculated position is moved to its proper screen position by adding  $x_c$  in  $x$  component and adding  $y_c$  in  $y$  component, i.e. by  $(x_c + x)$  and  $(y_c + y)$
- Along the section of circle from  $x = 0$  to  $x = y$  slope of the curve varies from  $0$  to  $-1$
- Therefore we take unit step in positive  $x$  direction over octant and use decision parameter to determine which of possible  $y$  position is closer to the circle path at each step
- We interchange the rate to  $x$  and  $y$  wherever the absolute value of slope of the circle tangent greater than  $1$ .
- Position in other seven octants are obtained by symmetry property

- We test the halfway position between two pixels to determine if this midpoint is inside or outside the circle boundary.

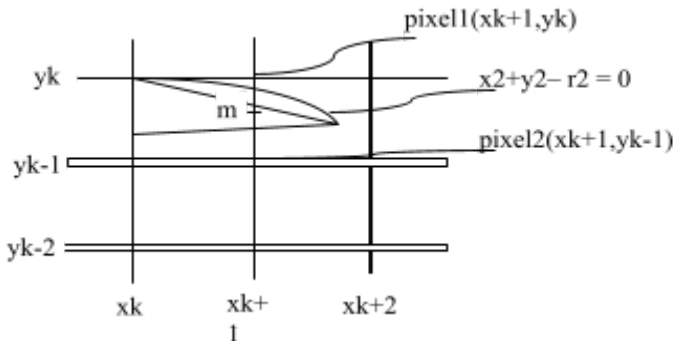
### 2.3.1 Midpoint Circle Algorithm (Derivation)

---

The equation of circle is  $x^2 + y^2 = r^2$

With center (0, 0) and radius r, let's define a circle function as circle  $(x, y) = x^2 + y^2 - r^2$

The distance of pixel to adjacent pixel is unit.



**Figure 2.7:** Mid-point circle pixels

In figure, length between pixel 1 and pixel 2 is  $(y_k - y_{k+1}) = 1$  and half the length = .

Circle  $(x, y) \{ \{$

- Assume  $(x_k, y_k)$  is plotted, then next point close to the circle is  $(x_{k+1}, y_k)$  or  $(x_k + 1, y_k - 1)$
- Decision parameter is the circle function evaluated at the midpoint between these two points.

$$p_k = f_{\text{circle}}(x_k + 1, y_k - )$$

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

So, if  $p_k < 0$ , then midpoint is inside the circle and  $y_k$  is closer to circle boundary. Else, midpoint is outside the circle. And  $y_k - 1$  is closer to the circle boundary

Successive decision parameters are obtained using incremental calculations, i.e., next decision parameter is obtained at

$$x_{k+1} + 1 = x_k + 1 + 1 \text{ and } y_{k+1} -$$

$$p_{k+1} = f_{\text{circle}}(x_{k+1} + 1, y_{k+1} -)$$

$$p_{k+1} = (x_k + 1 + 1)^2 + (y_{k+1} -)^2 - r^2 \dots\dots\dots (3)$$

Subtracting equation (2) from (3)

$$\begin{aligned} p_{k+1} - p_k &= (x_k + 1 + 1)^2 + (y_{k+1} -)^2 - r^2 - (x_k + 1)^2 - (y_k -)^2 + r^2 \\ &= (x_k + 1)^2 + 2(x_k + 1) + 1 + y_{k+1}^2 - 2y_{k+1} \times - - (x_k + 1)^2 - y_k^2 + 2y_k \times - \\ &= 2(x_k + 1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \\ &= 2x_{k+1} + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1 \end{aligned}$$

Where  $y_{k+1}$  is either  $y_k$  or  $y_k - 1$  depending on sign of  $p_k$

#### Case I:

If  $p_k < 0$ , then next pixel is at  $(x_k + 1, y_k)$

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

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

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

#### Case II:

If  $p_k \geq 0$ , then next pixel is at  $(x_k + 1, y_k - 1)$

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

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

$$\begin{aligned} p_{k+1} &= p_k + 2x_{k+1} + [(y_k - 1)^2 - y_k^2] - (y_k - 1 - y_k) + 1 \\ &= p_k + 2x_{k+1} + (y_k^2 - 2y_k + 1 - y_k^2) + 1 + 1 \\ &= p_k + 2x_{k+1} - 2y_k + 1 + 1 + 1 \\ &= p_k + 2x_{k+1} - 2(y_k - 1) + 1 \\ &= p_k + 2x_{k+1} - 2y_{k+1} + 1 \end{aligned}$$

## Initial Decision Parameter

The initial decision parameter is obtained by evaluating the circle function at starting point  $(x_0, y_0) = (0, r)$

$$\begin{aligned} p_0 &= f_{\text{circle}}((x_0 + 1), (y_0 - )) \\ &= f_{\text{circle}}((0 + 1), (r - )) \\ &= 1^2 + (r - )^2 - r^2 \\ &= 1 + r^2 - 2r \times + - r^2 \end{aligned}$$

$$p_0 = - r$$

If the radius  $r$  is specified as an integer, we can simply round to  $p_0 = 1 - r$

## Mid-point Circle Algorithm

Step 1. Start

Step 2. Declare variables  $x_c, y_c, r, x_0, y_0, p_0, p_k, p_{k+1}$ .

Step 3. Read values of  $x_c, y_c, r$ .

Step 4. Initialize the  $x_0$  and  $y_0$  i.e., set the co-ordinates for the first point on the circumference of the circle centered at origin as

$$x_0 = 0$$

$$y_0 = r$$

Step 5. Calculate initial value of decision parameter

$$p_0 = - r$$

Step 6. At each  $x_k$  position, starting from  $k = 0$

If  $p_k < 0$

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

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

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

else

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

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

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$$

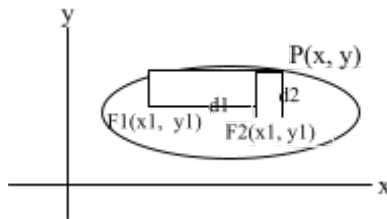
- Step 7. Determine the symmetry in other seven octants.
- Step 8. Move each calculated pixel position  $(x, y)$  onto the circular path centered on  $(x_c, y_c)$
- Step 9. Plot the co-ordinates values
- $$x = x + x_c$$
- $$y = y + y_c$$
- Step 10. Repeat steps 6 to 9 until  $x \geq y$ .
- Step 11. Stop

## 2.4 Ellipse

An ellipse is defined as the set of points such that the sum of the distances from two fixed point/positions (foci) is same for all points.

**Major axis:** The longer straight line segment extending from one side of the ellipse to the other through foci.

**Minor axis:** The shorter dimension line segment of the ellipse, bisecting the major axis at the halfway position (ellipse center) between the two foci.



**Figure 2.8: Ellipse**

General equation of ellipse,

$$\frac{x^2}{r_x^2} + \frac{y^2}{r_y^2} = 1$$

$r_x$  = radius of the ellipse in x direction i.e. center to boundary

$r_y$  = radius of the ellipse in y direction i.e. center to boundary

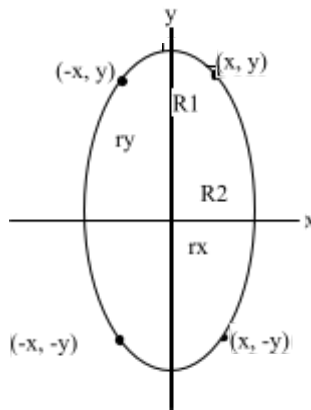
In polar form,

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

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

### 2.4.1 Mid-Point Ellipse Algorithm

---



**Figure 2.9:** Ellipse with symmetry points

An ellipse has 4 way symmetry, i. e., symmetric in quadrant only. So, we have to find the coordinate of a complete quadrant to use symmetry properties to plot the whole ellipse. Each quadrant is divided into two regions. The slope of curve in region 1 is lesser than  $-1$ . The slope of curve in region 2 is greater than  $-1$ . In region 1 we increase unit step in  $x$  direction and we determine whether to decrease or not in  $y$  direction. In region 2 we decrease unit step in  $y$  direction and we determine whether to increase or not in  $y$  direction.

Above figure shows the division of 1<sup>st</sup> quadrant according to the slope of an ellipse with  $r_x < r_y$ . We process this quadrant by taking unit step in  $x$ -direction where slope of curve is lesser than 1, and taking unit step in  $y$ -direction where slope has magnitude greater than 1. Region 1 and 2 can be processed in different ways.

- i. Start at position  $(0, r_y)$  and step clock wise along the elliptical path in first quadrants, shifting from unit step in  $x$  to unit step in  $y$  when slope becomes greater than  $-1$
- ii. Alternatively, start at position  $(r_x, 0)$  and select points in counter clockwise, shifting from unit step in  $y$  to unit step in  $x$  when the slope becomes less than  $-1$ .



Here, we start at position  $(0, r_y)$

we define an ellipse function with  $(x_c, y_c) = (0, 0)$

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

With properties

$$f_{\text{ellipse}}(x, y) = \begin{cases} 0, & \text{if } (x, y) \text{ is inside the ellipse boundary} \\ > 0, & \text{if } (x, y) \text{ is on the ellipse boundary} \\ < 0, & \text{if } (x, y) \text{ is outside the ellipse boundary} \end{cases}$$

**Thus ellipse function serves as the decision parameter.**

- At each sampling position, we select the next pixel along the ellipse path according to the sign of the ellipse function evaluated at the midpoint between the two candidate pixels.
- At each step, we test the value of the slope of the curve,
- Slope can be calculated as:  $r_y^2 x^2 + r_x^2 y^2 - r_x^2 r_y^2$

Differentiating with respect to  $x$ ,

$$2r_y^2 x + 2r_x^2 y = 0$$

$$\therefore \frac{dy}{dx} = -\frac{r_y^2 x}{r_x^2 y} \dots\dots\dots (ii)$$

At the boundary between region 1 and region 2, slope =  $-1$ . So,

$$-\frac{r_y^2 x}{r_x^2 y} = -1$$

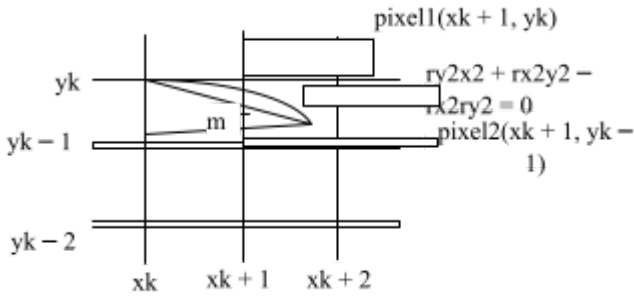
$$\text{or, } 2r_y^2 x = 2r_x^2 y$$

We move out of region 1, whenever

$$2r_y^2 x > 2r_x^2 y$$

We move out from region 2, whenever

$$2r_y^2 x < 2r_x^2 y$$



**Figure 2.10: Mid-point in region 1**

Assuming  $(x_k, y_k)$  has been illuminated (selected) we determine the next position along the ellipse path by evaluating the decision parameter at the midpoint  $(x_k + 1, y_k - 1)$ .

We have to determine the next point is  $(x_k + 1, y_k)$  or  $(x_k + 1, y_k - 1)$

We define decision parameter at mid-point as

$$p_{1k} = f_{\text{ellipse}}(x_k + 1, y_k - 1)$$

$$p_{1k} = r_y^2(x_k + 1)^2 + r_x^2 - r_x^2 r_y^2 \dots \dots \dots \text{(iii)}$$

If  $p_{1k} < 0$ , the mid-point is inside the ellipse and the pixel on scan line  $y_k$  is closer to the ellipse boundary

Otherwise, the mid-point is outside or on the boundary and we select the pixel on scan line  $y_k - 1$

At the next sampling position  $(x_{k+1} + 1 = x_k + 2)$ , the decision parameter for region 1 is evaluated as

$$p_{1k+1} = f_{\text{ellipse}}(x_{k+1} + 1, y_{k+1} - 1)$$

$$p_{1k+1} = r_y^2[(x_k + 1) + 1]^2 + r_x^2(y_{k+1} - 1)^2 - r_x^2 r_y^2 \dots \dots \dots \text{(iv)}$$

Subtracting equation (iii) from (iv),

$$p_{1k+1} - p_{1k} = r_y^2[(x_k + 1 + 1)^2 - (x_k + 1)^2] + r_x^2 - r_x^2 r_y^2 + r_x^2 r_y^2$$

$$= r_y^2[(x_k + 1)^2 + 2(x_k + 1) + 1 - (x_k + 1)^2] + r_x^2[y_{k+1}^2 - 2y_{k+1}x + -y_k^2 + 2y_kx -]$$

$$= 2r_y^2(x_k + 1) + r_y^2 + r_x^2 [(y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)]$$

$$= 2r_y^2 x_{k+1} + r_y^2 + r_x^2 [(y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k)]$$

Where  $y_{k+1}$  is either  $y_k$  or  $y_k - 1$ , depending on the sign of  $p_{1k}$

If  $p_{1k} < 0$  i.e.,  $y_{k+1} = y_k$ , then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

If  $p_{1k} \geq 0$  i.e.,  $y_{k+1} = y_k - 1$ , then decision parameter is

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 + r_x^2 [(y_k - 1)^2 - y_k^2] - (y_k - 1 - y_k)$$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 + r_x^2 [(y_k^2 - 2y_k + 1 - y_k^2) - (-1)]$$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 + r_x^2 [-2y_k + 1 + 1]$$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 + r_x^2 [-2(y_k - 1)]$$

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2 - 2r_x^2 y_{k+1}$$

The initial decision parameter is evaluated at start position  $(x_0, y_0) = (0, r_y)$  as

$$p_{10} = f_{\text{ellipse}}$$

$$= r_y^2 + r_x^2 (r_y - 1/2)^2 - r_x^2 r_y^2$$

$$p_{10} = r_y^2 - r_x^2 r_y + r_x^2 \dots \dots \dots (v)$$

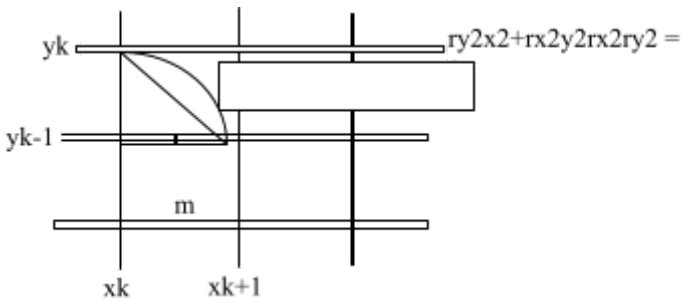
## Region 2

We sample at unit step in the negative y direction and the midpoint is now taken between horizontal pixels at each step. The decision parameter is

$$p_{2k} = f_{\text{ellipse}}$$

$$p_{2k} = r_y^2 + r_x^2 (y_k - 1)^2 - r_x^2 r_y^2 \dots \dots \dots (vi)$$

Put  $k = 0$  for initial decision parameter for region 2



**Figure 2.10: Midpoint in region 2**

If  $p_{2k} > 0$ , the midpoint is outside the ellipse boundary and we select the pixel  $x_k$ . If  $p_{2k} < 0$ , the midpoint is inside or on the ellipse boundary and we select pixel position  $x_k + 1$

Now, at next sampling position  $y_{k+1} - 1 = y_k - 2$

$$p_{2k+1} = f_{ellipse}\left(x_{k+1} + \frac{1}{2}, y_{k+1} - 1\right)$$

$$p_{2k+1} = r_{y^2}\left(x_{k+1} + \frac{1}{2}\right)^2 + r_{x^2}(y_{k+1} - 1)^2 - r_{x^2}r_{y^2}$$

.....(vii)

Subtracting equation (vi) from (vii)

$$\begin{aligned} p_{2k+1} &= p_{2k} + r_{y^2}\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] + r_{x^2}\left[(y_k - 1 - 1)^2 - (y_k - 1)^2\right] \\ &= p_{2k} + r_{x^2}\left[(y_k - 1)^2 - 2(y_k - 1) + 1 - (y_k - 1)^2\right] + r_{y^2}\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \\ &= p_{2k} + r_{x^2}\left[-2(y_k - 1) + 1\right] + r_{y^2}\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \\ &= p_{2k} + r_{x^2}\left[-2(y_{k+1}) + 1\right] + r_{y^2}\left[\left(x_{k+1} + \frac{1}{2}\right)^2 - \left(x_k + \frac{1}{2}\right)^2\right] \end{aligned}$$

If  $p_{2k} > 0$ , then

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

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

$$p_{2k+1} = p_{2k} - 2r_{x^2}y_{k+1} + r_{x^2}$$

If  $p_{2k} \leq 0$ , then

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

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

$$p_{2k+1} = p_{2k} + 2r_{y^2}x_{k+1} - 2r_{x^2}y_{k+1} + r_{x^2}$$

## Mid-point ellipse algorithm

Step 1. Start

Step 2. Declare variables  $x_c, y_c, r_x, r_y, x, y, p_0, p_{1k}, p_{1k+1}, p_{2k}, p_{2k+1}$

Step 3. Read Values of  $x_c, y_c, r_x, r_y,$

Step 4. Obtain the first point on an ellipse centered on origin ( $x, y$ ) by initializing the  $x$  and  $y$  as

$$x = 0$$

$$y = r_y$$

Step 5. Calculate the initial value of the decision parameter in region 1 as

$$p_{10} = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$$

Step 6. For each  $x_k$  position in region 1, starting at  $k = 0$ , perform the following test.

If  $p_{1k} < 0$ ,

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

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

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

else

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

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

$$p_{1k+1} = p_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

and continue until  $2r_y^2 x \geq 2r_x^2 y$

Step 7. Calculate the initial decision parameter in region 2 using the last point ( $x_0, y_0$ ) calculated in region 1 as

$$p_{20} = r_y^2 \left( x_0 + \frac{1}{2} \right)^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

Step 8. At each  $y_k$  position in region 2, starting at  $k = 0$ , perform the following test.

If  $p_{2k} > 0$ ,

the next point along the ellipse centered on (0, 0) is ( $x_k$ ,  $y_k - 1$ ) and

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

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

$$p_{2k+1} = p_{2k} - 2r_{x^2}y_{k+1} + r_{x^2}$$

else

the next point along the ellipse is

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

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

$$p_{2k+1} = p_{2k} + 2r_{y^2}x_{k+1} - 2r_{x^2}y_{k+1} + r_{x^2}$$

Use the same incremental calculations for x and y as in region 1.

Step 9. Determine the symmetry points in the other three quadrants.

Step 10. Move each calculated pixel position( $x$ ,  $y$ ) onto the elliptical path centered on ( $x_c$ ,  $y_c$ ) and plot the coordinate values.

$$x = x + x_c$$

$$y = y + y_c$$

Step 11. Repeat the steps for region 2 until  $y < 0$ .

Step 12. Stop.

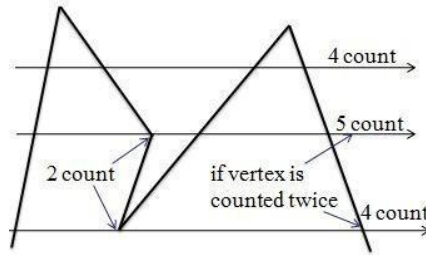
## 2.5 Filled Area Primitive

A standard output primitive in general graphics is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries. The main idea behind the 2D or 3D object filling procedure is that it provides us more realism on the object of interest. There are two basic approaches to area filling in raster systems.

- One way to fill an area is to determine the overlap intervals for scan lines that crosses the area.
- Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

### 2.5.1 SCAN-LINE Polygon Fill Algorithm:

---



In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it locates the intersection points of the scan line with the polygon edges. These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified color. In this method, the scan line must be even. At two edge connecting point called the vertex, we count the scan line two to handling the problem for scan line passing through vertex, but this consideration also may creates problem for some instant as shown in figure. To handle such problem shown by count-5, we keep the vertex blank and hence the count become 1, so that overall count in that scan line become even as shown in figure

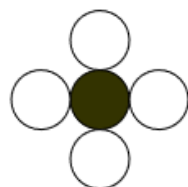
### 2.5.2 Boundary-fill Algorithm:

---

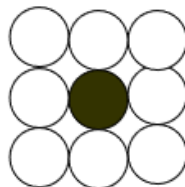
In Boundary filling algorithm starts at a point called seed pixel inside a region and paint the interior outward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by until the boundary color is reached. Starting from  $(x, y)$ , the procedure tests neighboring positions to determine whether they are of boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixel up to the boundary color area

havetested. The neighboring pixels from current pixel are proceeded by two method:

- 4-Connected (if they are adjacent horizontally and vertically.)
- 8-Connected (if they are adjacent horizontally, vertically and diagonally.)



4- Connected



8- Connected

#### **Algorithm for Boundary fill 4- connected:**

```
void Boundary_fill4(int x,int y,int b_color, int fill_color)
{
    int value=getpixel (x,y);
    if (value != b_color && value != fill_color)
    {
        putpixel (x,y,fill_color);
        Boundary_fill4(x-1,y, b_color, fill_color);
        Boundary_fill4(x+1,y, b_color, fill_color);
        Boundary_fill4(x,y-1, b_color, fill_color);
        Boundary_fill4(x,y+1, b_color, fill_color);
    }
}
```

#### **Algorithm for Boundary fill 8- connected:**

```
void Boundary-fill8(int x,int y,int b_color, int fill_color)
{
    int current =getpixel(x,y);
    if (current != b_color && current != fill_color)
```



```

{
    putpixel (x,y,fill_color); Boundary_fill8(x-1,y,b_color,fill_color);
    Boundary_fill8(x+1,y,b_color,fill_color);
    Boundary_fill8(x,y-1,b_color,fill_color);
    Boundary_fill8(x,y+1,b_color,fill_color);
    Boundary_fill8(x-1,y-1,b_color,fill_color);
    Boundary_fill8(x-1,y+1,b_color,fill_color);
    Boundary_fill8(x+1,y-1,b_color,fill_color);
    Boundary_fill8(x+1,y+1,b_color,fill_color);
}
}

```

Recursive boundary-fill algorithm not fills regions correctly if some interior pixels are already displayed in the fill color. Encountering a pixel with the fill color can cause a recursive branch to terminate, leaving other interior pixel unfilled. To avoid this we can first change the color of any interior pixels that are initially set to the fill color before applying the boundary fill procedure.

### **2.5.3 Flood-fill Algorithm:**

---

Flood\_fill Algorithm is applicable when we want to fill an area that is not defined within a single color boundary. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called flood fill algorithm. We start from a specified interior pixel (x,y) and reassign all pixel values that are currently set to a given interior color with desired fill\_color. Using either 4-connected or 8-connected region recursively starting from input position, The algorithm fills the area by desired color.

#### **Algorithm:**

```

void flood_fill4(int x,int y,int fill_color,int old_color)
{

```

```

    int current;

```

```

    current = getpixel (x,y); if (current == old_color)

```

```

{
    putpixel (x,y,fill_color);
    flood_fill4(x-1,y, fill_color, old_color); flood_fill4(x,y-1,
        fill_color, old_color); flood_fill4(x,y+1, fill_color,
        old_color); flood_fill4(x+1,y, fill_color, old_color);
}
}

```

---

1. Consider a line from (3, 7) to (8, 3). Using simple DDA algorithm, rasterize this line.

**Solution:**

Here,

Starting point  $(x_1, y_1) = (3, 7)$

Ending point  $(x_2, y_2) = (8, 3)$

Slope  $(m) =$

$=$

$= -0.8$

Since  $|m| < 1$ , from DDA Algorithm we get,

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

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

K	$x_k$	$y_k$	$x_{plot}$	$y_{plot}$	$(x_{k+1}, y_{k+1})$
1	3	7	3	7	(3,7)
2	4	6.2	4	6	(4,6)
3	5	5.4	5	5	(5,5)
4	6	4.6	6	5	(6,5)
5	7	3.8	7	4	(7,4)
6	8	3	8	3	(8,3)

2. Use Bresenham's algorithm to scan convert a straight line connecting the end points (20, 10) and (30, 18).

[074 Ashwin]

**Solution:**

$$(x_0, y_0) = (20, 10)$$

$$\Delta x = |x_2 - x_1| = 10$$

$$\Delta y = |y_2 - y_1| = 8$$

$$p_0 = 2\Delta y - \Delta x \text{ (since } \Delta x > \Delta y \text{ i.e., } m < 1) \\ = 6$$

For  $\Delta x > \Delta y$ ,

if  $p_k < 0$ ,

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

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

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

if  $p_k > 0$ ,

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

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

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

K	$p_k$	$(x_{k+1}, y_{k+1})$
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)

3. **Determine the raster location along the circle octant in the first quadrant for a circle with radius 10.**

**Solution:**

$$r = 10$$

Initial decision parameter,

$$p_0 = 1 - r = 1 - 10 = -9$$

$$(x_0, y_0) = (0, 10)$$

$$2x_0 = 0, 2y_0 = 20$$

If  $p_k < 0$

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

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

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

else

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

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

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$$

K	$p_k$	$(x_{k+1}, y_{k+1})$	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14

4. *Using midpoint circle algorithm, calculate the co-ordinate on the first quadrant of a circle having radius 6 and centre at (20, 10).*

**Solution:**

$$r = 6$$

$$(x_c, y_c) = (20, 10)$$

$$p_0 = 1 - r = 1 - 6 = -5$$

$$(x_0, y_0) = (0, 6)$$

If  $p_k < 0$

$$(x_{k+1}, y_{k+1}) = (x_k + 1, y_k)$$

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

Else

$$(x_{k+1}, y_{k+1}) = (x_k + 1, y_k - 1)$$

$$p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$$

For first octant,

k	$p_k$	$x_{k+1}, y_{k+1}$	$2x_{k+1}$	$2y_{k+1}$	Pixel to plot ( $x_{k+1} + x_c, y_{k+1} + y_c$ )
		(0, 6)	0	12	(20, 16)
0	-5	(1, 6)	2	12	(21, 16)
1	-2	(2, 6)	4	12	(22, 16)
2	3	(3, 5)	6	10	(23, 15)
3	0	(4, 4)	8	8	(24, 14)

For second octant, using symmetry  $(x, y) \rightarrow (y, x)$

$(0, 6) \rightarrow (6, 0)$ ,  $(1, 6) \rightarrow (6, 1)$ ,  $(2, 6) \rightarrow (6, 2)$ ,  $(3, 5) \rightarrow (5, 3)$   
and  $(4, 4) \rightarrow (4, 4)$

At last we need to shift the co-ordinates in the center (20, 10) so,

$(6, 0) \rightarrow (26, 10)$ ,  $(6, 1) \rightarrow (26, 11)$ ,  $(6, 2) \rightarrow (26, 12)$ ,  $(5, 3) \rightarrow (25, 13)$  and  $(4, 4) \rightarrow (24, 14)$

5. **Determine the pixel positions of following curve in first quadrant using mid-point algorithm.**

$$+ = 1$$

[2076 Ashwin]

**Solution:**

$$\text{Here, } r_x^2 = 64, r_y^2 = 36$$

$$\text{Here, half of major axis } (r_x) = 8$$

$$\text{Half of minor axis } (r_y) = 6$$

First point of the ellipse starting with  $(0, r_y)$  is

$$x_0 = 0$$

$$y_0 = r_y$$

$$\text{So, } x_0 = 0, y_0 = 6$$

Now,

Initial decision parameter in region 1 is

$$\begin{aligned} P_{10} &= r_y^2 - r_x^2 r_y + r_x^2 \\ &= 6^2 - 8^2 \times 6 + 8^2 = -332 \end{aligned}$$

Now, for each  $x_k$  position in region 1, starting at  $k = 0$ , we perform the following test

If  $P_{1k} < 0$ ,

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

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

$$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} + r_y^2$$

Else

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

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

$$P_{1k+1} = P_{1k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_y^2$$

and will continue until  $2r_y^2 x \geq 2r_x^2 y$

Similarly, we calculate the initial decision parameter in region 2 using the last point  $(x_0, y_0)$  calculated in region 1 as

$$P_{20} = r_y^2 + r_x^2 (y_0 - 1)^2 - r_x^2 r_y^2$$

Now, at each  $y_k$  position in regions, starting at  $k = 0$ , we perform following test

If  $P_{2k} > 0$ ,

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

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

$$P_{2k+1} = P_{2k} - 2r_x^2 y_{k+1} + r_x^2$$

Else

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

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

$$P_{2k+1} = P_{2k} + 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$$

We repeat the steps for region 2 until  $y < 0$

$(x_k, y_k)$	$P_{1k}$	$(x_{k+1}, y_{k+1})$	$2r_y^2 x_{k+1}$	$2r_x^2 y_{k+1}$	$P_{1k+1}$
(0, 6)	-332	(1, 6)	72	768	-224
(1, 6)	-224	(2, 6)	144	768	-44
(2, 6)	-44	(3, 6)	216	768	208
(3, 6)	208	(4, 5)	288	640	-108
(4, 5)	-108	(5, 5)	360	640	288
(5, 5)	288	(6, 4)	432	512	244
(6, 4)	244	(7, 3)	504	384	

The point (7, 3) will be the initial point for region 2. The initial decision parameter for region 2 is  $P_{2k}$

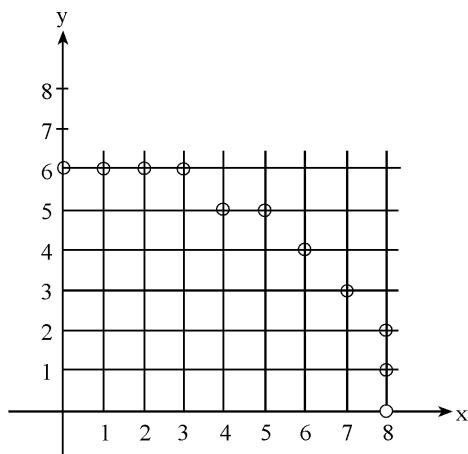
$$= 6^2 + 8^2 (3 - 1)^2 - 8^2 * 6^2$$

$$= -23$$

$(x_k, y_k)$	$P_{2k}$	$(x_{k+1}, y_{k+1})$	$P_{2k+1}$
(7, 3)	-23	(8, 2)	361
(8, 2)	361	(8, 1)	297
(8, 1)	297	(8, 0)	

So, the pixel positions or points to plot the given curve in first quadrant using mid-point algorithm are

(0, 6), (1, 6), (2, 6), (3, 6), (4, 5), (5, 5), (6, 4), (7, 3), (8, 2), (8, 1) and (8, 0)



6. *Digitize a line with end points (10, 20) and (15, 2) using Bresenham's algorithm.* [079 Baishakh]

**Solution:**

Here,

Starting point  $(x_1, y_1) = (10, 20)$

End point  $(x_2, y_2) = (15, 2)$

$\Delta x = \text{absolute}(x_2 - x_1) = |15 - 10| = 5$

$\Delta y = \text{absolute}(y_2 - y_1) = |2 - 20| = 18$

$p_0 = 2 \times \Delta x - \Delta y$  (since  $\Delta y > \Delta x$  i.e.,  $m > 1$ )

$$= 2 \times 5 - 18$$

$$= -8$$

For  $\Delta y > \Delta x$ ,

if  $p_k < 0$ ,

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

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

$$p_{k+1} = p_k + 2 \times \Delta x$$

if  $p_k > 0$ ,

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

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

$$p_{k+1} = p_k + 2 \times \Delta x - 2 \times \Delta y$$

K	$p_k$	$(x_{k+1}, y_{k+1})$
0	-8	(10, 19)
1	2	(11, 18)
2	-24	(11, 17)
3	-14	(11, 16)
4	-4	(11, 15)
5	6	(12, 14)
6	-20	(12, 13)
7	-10	(12, 12)



8	0	(13, 11)
9	- 26	(13, 10)
10	- 16	(13, 9)
11	- 6	(13, 8)
12	4	(14, 7 )
13	- 22	(14, 6)
14	- 12	(14, 5)
15	- 2	(14, 4)
16	8	(15, 3)
17	-14	(15, 2)

So, the pixel positions or points to plot the given line are (10, 20), (10, 19), (11, 18), (11, 17), (11, 16), (11, 15), (12, 14), (12, 13), (12, 12), (13, 11), (13, 10), (13, 9), (13, 8), (14, 7), (14, 6), (14, 5), (14, 4), (14, 3), 15, 3), and (15, 2)

---

# Two-Dimensional Transformations

---

## 3.1 Introduction

---

Picture can be treated as a combination of straight line, circle, ellipse, etc. and if we are able to generate these basic figures, we can also generate combinations of them. Transformation is the process that changes the position, orientation, size or shape of the original graphics by applying rules. When transformation occurs in two dimensional plane then it is called two dimensional transformations. Geometrical transformation is a mathematical procedure which changes/alters orientation, size, and shape of objects i.e., the co-ordinate description of objects.

## 3.2 Basic Transformations

---

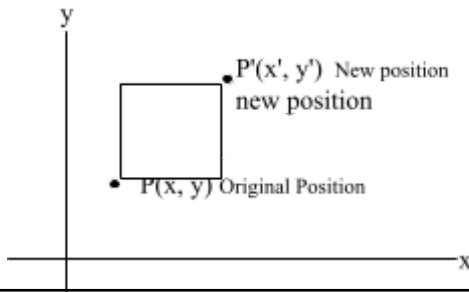
The basic transformations are:

- Translation
- Scaling
- Rotation

### 3.2.1 Translation/Shifting

---

Translation repositions an object along a straight line path from one co-ordinate location to another. We can translate a two dimensional point by adding translation distances  $t_x$  and  $t_y$  to the original co-ordinate position to move the point to a new position  $(x', y')$ .



**Figure 3.1:** Translation of a point

$$x' = x + t_x$$

$$y' = y + t_y$$

Translation distance pair  $(t_x, t_y)$  is called translation vector or shift vector.

In matrix form, we can represent translation as

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} + \begin{bmatrix} t_x & t_y \end{bmatrix}$$

$$P' = P + T$$

Where,

$P'$  = New translated object matrix

$T$  = Translation matrix

$P$  = Original object matrix

In translation, just position is changed but shape and size are not changed. Straight line is translated by applying transformation equation to both end points and redrawing the line between those translated end points. Polygon is translated by adding translation vector to the co-ordinates position of each vertex and new polygon is regenerated using that new vertices.

Circle and ellipse are translated by translating the center co-ordinates and then, circle and ellipse are redrawn in new location.

### 3.2.2 Scaling

---

A scaling is a basic transformation that changes the size or shape (elongation or shrinking) of object.

Scaling with respect to origin can be done by multiplying the coordinate values  $(x, y)$  of each vertex of a polygon or each point of a line by scaling factors  $s_x$  and  $s_y$  respectively to produce the coordinates  $(x', y')$ .  $s_x$  is scaling factor in  $x$  direction.  $s_y$  is scaling factor in  $y$  direction.

Transformation equations are:

$$x' = x \cdot s_x$$

$$y' = y \cdot s_y$$

In matrix form,

$$[x' \ y'] = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = s \cdot P$$

If  $s_x = s_y$ , it is uniform scaling. If  $s_x \neq s_y$ , it is differential scaling.

If  $(s_x, s_y) < 1$ , then size is reduced.

$= 1$ , size is unchanged (remains same).

$> 1$ , size is enlarged.

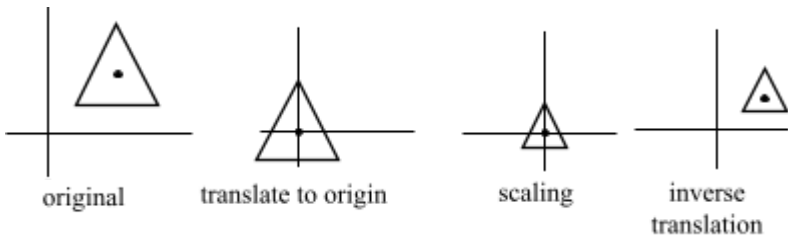
If scaling factor is less than 1, then it moves the object closer to origin. If scaling factor is greater than 1, then it moves the objects away from origin.

### Fixed point scaling

The location of the scaled object can be controlled by choosing a position called fixed point that is to remain unchanged after the scaling transformation. Fixed point  $(x_f, y_f)$  can be chosen as one of the vertices, centroid of the object, or any other position.

#### Steps:

1. Translate object so that the fixed point coincides with the co-ordinate origin.
2. Scale the object with respect to the co-ordinate origin.
3. Use the inverse translation of steps 1 to return the object to its original position.



**Figure 3.2:** Fixed point scaling of a triangle

$$\begin{bmatrix} 1 & 0 & x_f & 0 & 1 & y & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 & s_y & 0 & 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & -x_f & 0 & 1 & -y_f & 0 & 0 & 1 \end{bmatrix} = \\ \begin{bmatrix} s_x & 0 & x_f(1-s_x) & 0 & s_y & y_f(1-s_y) & 0 & 0 & 1 \end{bmatrix}$$

$$T(x_f, y_f). S(s_x, s_y). T(-x_f, -y_f) = S(x_f, y_f, s_x, s_y)$$

$$x' = x_f + (x - x_f) s_x$$

$$y' = y_f + (y - y_f) s_y$$

OR

$$x' = x.s_x + x_f(1 - s_x)$$

$$y' = y.s_y + y_f(1 - s_y)$$

Where the terms  $x_f(1 - s_x)$  and  $y_f(1 - s_y)$  are constant for all points in object.

### 3.2.3 Rotation

Rotation repositions an object along a circular path in the xy plane. To generate rotation, we specify a rotation angle  $\theta$  through which the coordinates are to be rotated.

Besides the rotation angle ( $\theta$ ), there should be a pivot point through which the object is to be rotated.

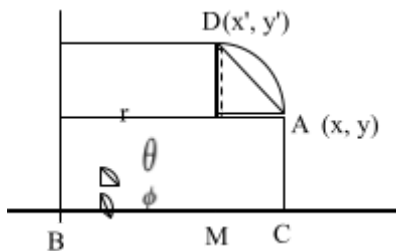
Rotation can be made either clockwise or anticlockwise. If  $\theta$  is positive, object is rotated in counterclockwise direction and if  $\theta$  is negative, object is rotated in clockwise direction.

A line perpendicular to rotating plane and passing through pivot point is called axis of rotation.

#### 1. Transformation equation for rotation when pivot point is origin

Let  $A(x, y)$  is a point in xy-plane which is to be rotated with angle  $\theta$ . Let  $OA = r$  is constant distance from origin. Let  $r$  makes  $\phi$  with positive x-direction as in fig 3.3 (a). When  $OA$  is rotated through angle  $\theta$  about the origin, i.e. making origin

a pivot point for rotation, then OD makes an angle  $\theta + \phi$  with x-axis.



**Figure 3.3(a):** Rotation of a point

From figure, using standard trigonometric identities  
In  $\triangle ABC$ ,

$$x = r \cos\phi \dots\dots\dots (i)$$

$$y = r \sin\phi \dots\dots\dots (ii)$$

In  $\triangle DBM$ ,

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

$$x' = r \cos\theta \cos\phi - r \sin\theta \sin\phi$$

Now, substituting value of equation (i) and (ii)

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

again,

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

$$y' = r \sin\theta \cos\phi + r \cos\theta \sin\phi$$

Now, substituting value of equation (i) and (ii)

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

Representing in matrix form,

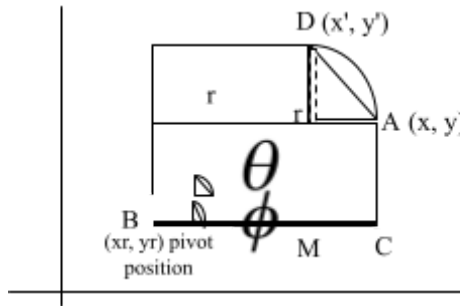
$$[x' \ y'] = [\cos\theta \ -\sin\theta \ \sin\theta \ \cos\theta][x \ y]$$

$$P' = R.P$$

## 2. Rotation of a point about an arbitrary pivot position

For a fixed point rotation, we can generate rotations about any selected pivot point  $(x_r, y_r)$  by performing the composite transformation as follows:

1. Translate the pivot point  $(x_r, y_r)$  to the coordinate origin.
2. Rotate object about origin with rotation angle  $\theta$
3. Translate back so that the pivot point is returned to its original position.



**Figure 3.3(b):** Rotation of a point

Composite matrix, C.M. =  $T^{-1} R(\theta) T$

$$= \begin{bmatrix} 1 & 0 & x_r & 0 & 1 & y & 0 & 0 & 1 \\ \cos\theta & -\sin\theta & 0 & \sin\theta & \cos\theta & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r & 0 & 1 & -y & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & x_r(1 - \cos\theta) + y_r \sin\theta & \sin\theta \cos\theta & y_r(1 - \cos\theta) - x_r \sin\theta & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x' = x_r + (x - x_r) \cos\theta - (y - y_r) \sin\theta$$

$$y' = y_r + (x - x_r) \sin\theta + (y - y_r) \cos\theta$$

### 3.3 Homogeneous Co-ordinates and Matrix Representations

The matrix representation for translation, scaling, and rotation are respectively:

$$P' = T + P$$

$$P' = S \cdot P$$

$$P' = R \cdot P$$

Translation involves addition of matrices, whereas scaling and rotation involve multiplication. We may have to perform more than one transformation in same object like scaling the object, then rotate the same object, and finally translation. For this, first co-ordinate positions are scaled, then this scaled co-ordinates are rotated and finally translated. A more efficient approach would be to combine the transformation so that final positions are obtained directly from initial co-ordinates thereby eliminating the calculation of intermediate co-ordinates. This allows us to represent all geometric transformation as matrix multiplication. Homogeneous coordinate system treats all the transformation in terms of matrix multiplication. A general homogeneous coordinate representation of any point  $P(x, y)$  is  $(x_h, y_h, h)$ .

where  $x = \frac{x_h}{h}$ ,  $y = \frac{y_h}{h}$

Thus, general homogenous co-ordinate representation can also be written as

$$(x_h, y_h, h) = (h \cdot x, h \cdot y, h)$$

where  $h$  may be any non-zero value. But for convenience,  $h = 1$  is used.

So, 2D homogeneous co-ordinates is represented as  $(x, y, 1)$ .

Expressing position in homogeneous co-ordinates allows us to represent all geometric transformation equation as matrix multiplication.

#### 1. Translation

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x & 0 & 1 & t_y & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$



$$P' = T(t_x, t_y).P$$

Inverse of translation matrix can be obtained by replacing the translation parameter  $t_x$  and  $t_y$  with their negatives. i.e.,  $-t_x$  and  $-t_y$ .

## 2. Rotation

$$[x' \ y' \ 1] = [\cos\theta \ -\sin\theta \ 0 \ \sin\theta \ \cos\theta \ 0 \ 0 \ 0 \ 1] \cdot [x \ y \ 1]$$

$$P' = R(\theta) \cdot P$$

Inverse rotation matrix can be obtained by replacing  $\theta$  with  $-\theta$ .

## 3. Scaling

$$[x' \ y' \ 1] = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot [x \ y \ 1]$$

$$P' = S(s_x, s_y) \cdot P$$

Inverse translation matrix can be obtained by replacing  $s_x$  and  $s_y$  with  $1/s_x$  and  $1/s_y$  respectively.

### Code in C for 2D Transformation

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void draw(int a[2][5]);
void translate(int a[2][5]);
void scale(int a[2][5]);
void rotate(int a[2][5]);
void main()
{
    int a[2][5], i, gd = DETECT, gm;
    initgraph(&gd,&gm,"c:\\tc\\bgi");
    printf("Enter the coordinate positions of the lines");
    for(i=0; i < 5; i++)
    {
        scanf("%d %d",&a[0][i], &a[1][i]);
    }

    draw(a);
```

```
translate(a);
draw(a);
scale(a);
draw(a);
rotate(a);
draw(a);
getch();
closegraph();
}
```

```
void draw(int a[2][5])
{
int i=0;
for(i=0; i < 4; i++)
{
line(a[0][i], a[1][i], a[0][i+1], a[1][i+1]);
}
}
```

```
void translate(int a[2][5])
{int i;
int tx,ty;
scanf("%d %d", &tx, &ty);
for(i=0; i < 5; i++)
{
a[0][i]=a[0][i] + tx;
a[1][i]=a[1][i] + ty;
}
}
```

```

void scale(int a[2][5])
{int i;
int sx, sy;
scanf("%d%d", &sx, &sy);
for(i=0; i < 5; i++)
{
a[0][i]=a[0][i] × sx;
a[1][i]=a[1][i] × sy;
}
}

```

```

void rotate(int a[2][5])
{int i, temp1, temp2;
float angle;
scanf("%f",&angle);
for(i=0; i < 5; i++)
{
temp1=a[0][i];
temp2=a[1][i];
a[0][i]=temp1 × cos(angle) - temp2 × sin(angle);
a[1][i]=temp1 × sin(angle) + temp2 × cos(angle);
}
}

```

### 3.4 Composite Transformation

A matrix can be made for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations.

### 3.4.1 Translation

---

If two successive transformation vectors  $(t_{x1}, t_{y1})$  and  $(t_{x2}, t_{y2})$  are applied to a point P, the final position or transformed location P' is calculated as:

$$\begin{aligned} P' &= T(t_{x2}, t_{y2})\{T(t_{x1}, t_{y1}).P\} \\ &= \{T(t_{x2}, t_{y2})T(t_{x1}, t_{y1})\}.P \\ &= T(t_{x1} + t_{x2}, t_{y1} + t_{y2}).P \end{aligned}$$

The composite transformation matrix for this sequence of transformation is

$$\begin{bmatrix} 1 & 0 & tx_2 & 0 & 1 & ty_2 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & tx_1 & 0 & 1 & ty_1 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx_1 + tx_2 & 0 & 1 & ty_1 + ty_2 & 0 & 0 & 1 \end{bmatrix}$$

This shows that two successive translations are additive.

### 3.4.2 Rotation

---

$$P' = R(\theta_2). \{R(\theta_1).p\} = \{R(\theta_2).R(\theta_1)\}.P = R(\theta_1 + \theta_2) * P$$

$$\begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & \sin\theta_2 & \cos\theta_2 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & \sin\theta_1 & \cos\theta_1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Successive rotations are additive.

### 3.4.3 Scaling

---

$$P' = S(Sx_2, Sy_2)\{S(Sx_1, Sy_1).P\} = \{S(Sx_2, Sy_2).S(Sx_1, Sy_1)\}.P = S(Sx_1 \cdot Sx_2, Sy_1 \cdot Sy_2).P$$

$$\begin{bmatrix} sx_2 & 0 & 0 & 0 & sy_2 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} sx_1 & 0 & 0 & 0 & sy_1 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} sx_1 \cdot sx_2 & 0 & 0 & 0 & sy_1 \cdot sy_2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Two successive scaling are multiplicative

Composite transformation indicates combination of different basic transformation in sequence to obtain desired result or combination could be a sequence of two or more successive transformation (e.g., two successive translations), two successive rotations or two or more successive scaling, etc.

## 3.5 Other Transformations

Other transformations are:

- Shearing
- Reflection

### 3.5.1 Shearing

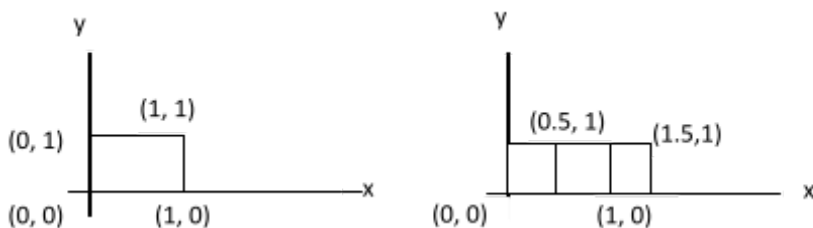
Shearing is the transformation that distorts the shape of the object such that the transformed shape appears as if the object was composed of internal layers that had been caused to slide over each other is called shearing.

#### i) Shearing toward x-direction relative to x-axis

The x-direction shear relative to x-axis is given by following equation,

$$x' = x + sh_x \cdot y$$

$$y' = y$$



**Figure 3.4:** Shearing towards x-direction

In matrix form,

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$

- $sh_x$  is any real number
- co-ordinate position is shifted horizontally by an amount proportional to its distance  $y$  value from the axis.
- If  $sh_x$  is negative, object shearing is towards left.

**ii) Shearing towards y - direction relative to y - axis**

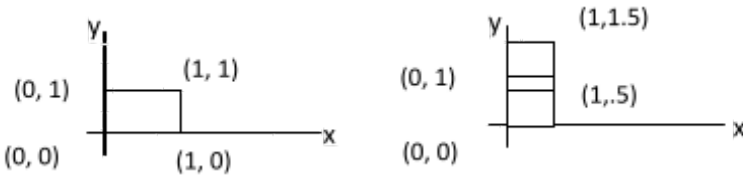
The y-direction shear relative to y-axis is given by following equation,

$$x' = x$$

$$y' = x \cdot sh_y + y$$

In matrix form,

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & sh_y & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$



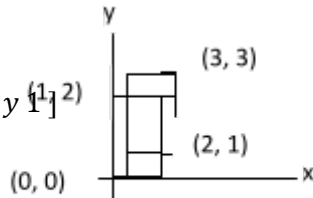
**Figure 3.5: Shearing towards y-direction**

**iii) Shearing in both direction is given by**

$$x' = x + sh_x \cdot y$$

$$y' = x \cdot sh_y + y$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 & sh_y & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$



**iv) x - direction shearing relative to other reference line ( $y-y_{ref}$ ) is**

$$x' = x + sh_x \cdot (y - y_{ref})$$

$$y' = y$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$

**v) y - direction shearing relative to other reference line ( $x-x_{ref}$ ) is**

$$x' = x$$

$$y' = sh_y(x - x_{ref}) + y$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & sh_y & 1 & -sh_y \cdot x_{ref} & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$

This transformation shifts a co-ordinate position vertically by an amount proportional to its distance from the reference line  $x = x_{ref}$

### 3.5.2 Reflection

It is the transformation that produces mirror image of an object. The mirror image for a two-dimensional reflection is obtained by rotating the object  $180^\circ$  about the reflection axis.

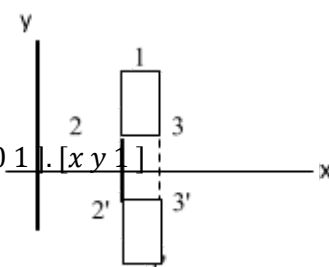
#### i) Reflection about x - axis or about line $y = 0$

The line representing x-axis is  $y = 0$ . The reflection of a point P (x, y) on x-axis, changes the y-coordinate sign and x-coordinate remains same. So, if we reflect P (x, y) in x-axis the 'x' value remains same but sign of 'y' value changes.

$$x' = x$$

$$y' = -y$$

then we get, P' (x, -y)

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


#### ii) Reflection about y - axis or about line $x = 0$

The line representing y-axis is  $x = 0$ . The reflection of a point P (x, y) on y-axis, changes the x-coordinate sign and y-coordinate remains same. So, if we

reflect P (x, y) in y-axis the 'y' value remains same but sign of 'x' value changes.

$$x' = -x$$

$$y' = y$$

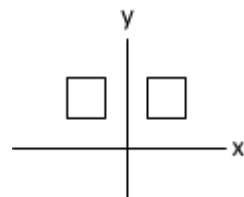


Fig. 3.8: Reflection about y-axis



$$P' = R_{fy}.P$$

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

### iii) Reflection about origin

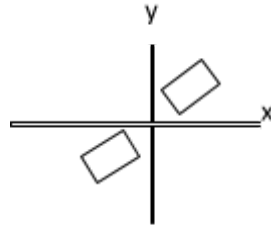
We choose the mirror line as the axis perpendicular to the xy-plane and passing through the origin.

After reflection, both x and y value changes

i.e.,

$$x' = -x$$

$$y' = -y$$



**Fig. 3.9:** Reflection about origin

### iv) Reflection about the line $y = x$

If we reflect the point about the line  $y = x$ , we get,  $x' = y$  and  $y' = x$ , we achieve this reflection by following steps:

1. Rotate about origin in clockwise direction by  $45^\circ$ , rotates the line  $y = x$  to x-axis.
2. Take reflection against x-axis
3. Rotate in anti-clockwise direction by same angle.

$$Rf_{(y=x)} = R(\theta)^{-1} \times Rf_x \times R(\theta)$$



**Figure 3.10:** Reflection about the line  $y = x$

$$R(\theta)^{-1} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & \sin\theta & \cos\theta & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rf_x = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R(\theta) = [\cos\theta \sin\theta \ 0 \ -\sin\theta \cos\theta \ 0 \ 0 \ 0 \ 1]$$

$$Rf_{(y=x)} = [0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]$$

**v) Reflection about the line  $y = -x$**

Reflecting point  $P(x, y)$  in the line  $y = -x$ , we get  $P'(-y, -x)$ .  
We get it following steps:

1. Rotate about origin in clockwise direction by  $45^\circ$ , it rotates line  $y = x$  to y-axis.
2. Take reflection against y-axis.
3. Rotate in clockwise direction by same angle.

$$Rf_{(y=-x)} = R(\theta)^{-1} \times Rf_y \times R(\theta)$$

where

$$R(\theta)^{-1} = [\cos\theta \ -\sin\theta \ 0 \ \sin\theta \ \cos\theta \ 0 \ 0 \ 0 \ 1]$$

$$Rf_y = [-1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1]$$

$$R(\theta) = [\cos\theta \ \sin\theta \ 0 \ -\sin\theta \ \cos\theta \ 0 \ 0 \ 0 \ 1]$$

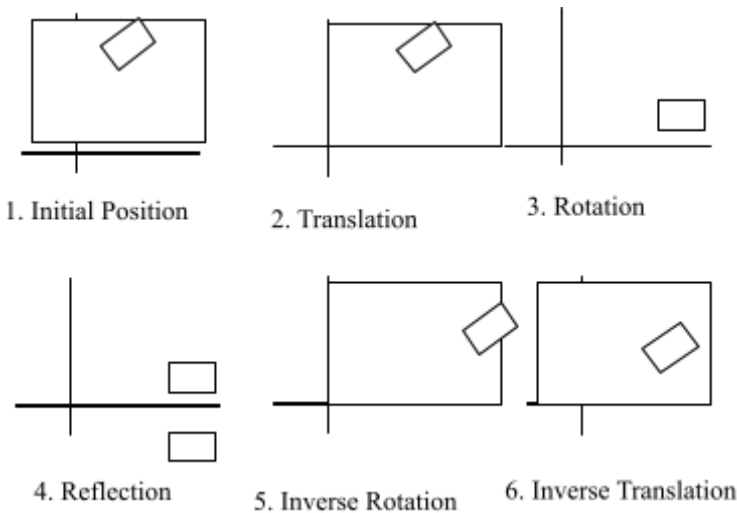
$$Rf_{y=-x} = [0 \ -1 \ 0 \ -1 \ 0 \ 0 \ 0 \ 0 \ 1]$$

**vi. Reflection about  $y = mx + c$**

Reflection about any line  $y = mx + c$  in the  $xy$  – plane can be accomplished by following steps:

1. Translate line and object so that line passes through origin.
2. Rotate the line and object about origin until the line coincides with one of the co-ordinate axis.
3. Reflect the object through about that axis.
4. Apply inverse rotation about that axis.
5. Translate back to original location.

$$C.M. = T^{-1} R(\theta)^{-1} R_f R(\theta) \cdot T$$



**Figure 3.11:** Reflection about the line  $y = mx + c$

Here,

$$\begin{aligned}
 [x' \ y' \ 1] &= [1 \ 0 \ 0 \ 0 \ 1 \ c \ 0 \ 0 \ 1] \\
 &[ \cos\theta \ -\sin\theta \ 0 \ \sin\theta \ \cos\theta \ 0 \ 0 \ 0 \ 1 ] [1 \ 0 \ 0 \ 0 \ -1 \ 0 \ 0 \ 0 \ 1] \\
 &[ \cos\theta \ \sin\theta \ 0 \ -\sin\theta \ \cos\theta \ 0 \ 0 \ 0 \ 1 ] [1 \ 0 \ 0 \ 0 \ 1 \ -c \ 0 \ 0 \ 1] \\
 &[ \ ] [ \ ]
 \end{aligned}$$

OR,

$$\begin{aligned}
 [x' \ y' \ 1] &= \left[ \frac{1-m^2}{1+m^2} \ \frac{2m}{1+m^2} \ \frac{-2cm}{1+m^2} \ \frac{2m}{1+m^2} \ \frac{m^2-1}{1+m^2} \ \frac{2cm}{1+m^2} \ 0 \ 0 \ 1 \right] \\
 &[ \ ] [ \ ]
 \end{aligned}$$

### 3.6 Two - Dimensional Viewing

*Two-dimensional viewing* is the method to display two dimensional objects in display device. It is the formal mechanism for displaying views of picture on an output device. Graphics package allows a user to specify which part of a defined picture is to be displayed and where is to be displayed in device. Any convenient Cartesian coordinate system, referred to as the world co-ordinate reference frame, can be used to define the pictures.

For a two dimensional picture, a view is selected by specifying a sub area of the total picture area. A user can select a

single area for display, or several areas could be selected for simultaneous display or for an animated panning sequence across a scene. The picture parts within the selected areas are then mapped onto specified areas of the device co-ordinates. When multiple view areas are selected, these areas can be placed in separated display locations, or some areas could be inserted into other, larger display areas.

Transformation from world to device co-ordinate involve translation, rotation, and scaling operations as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area, also known as clipping. Windowing is the process of selecting and enlarging the portions of a drawing.

### **3.7 Coordinate Representation**

The general packages use Cartesian co-ordinate system. If some package is designed to use other coordinate system, it must be converted to Cartesian co-ordinate system .

#### **Modeling coordinate/ local coordinate/ master coordinate**

The image of individual object can be constructed in a scene within separate coordinate system. It is known as modeling or local or master coordinate.

#### **World coordinate**

Once individual objects have been identified or specified, those objects are placed into appropriate position within a scene using reference frame and this reference frame is called world co-ordinate. It contains many objects with one unit.

#### **Viewing coordinate (Camera coordinate)**

Viewing coordinate is used to define window in the world co-ordinate plane with any possible orientation, i.e. viewing some objects or items at a time.

## Normalized device coordinate

Generally, a graphical system, world coordinate positions are converted to normalized device coordinates before final conversion to specified device coordinate. In normalized coordinate the coordinates are in range 0 to 1. This makes the system independent of the various devices that might be used at a particular workstation.

## Device coordinate

When the world coordinate description of the scene is transformed to one or more output device reference frame for display; the display coordinate system is referred to a device coordinate or screen coordinates in the case of video monitor.

### 3.8 The Viewing Pipeline (Viewing Transformation)

Displaying an image of a picture involves mapping the coordinates of the points and lines that form the picture into the appropriate coordinates on the device or workstation where the image is to be displayed. This is done through the use of coordinate transformations called viewing transformations.

## Window

A world coordinate area selected for display is called window.

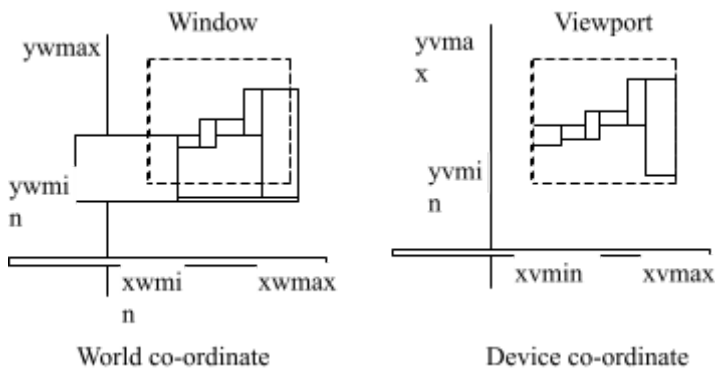
## View port

An area on a display device to which a window is mapped is called view port.

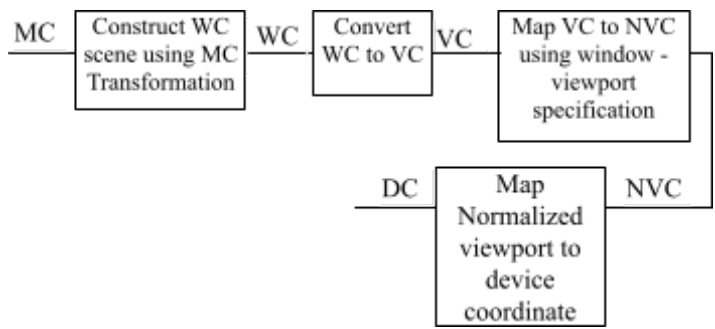
Window defines what is to be viewed. View port defines where is to be displayed. Window and viewport are rectangular in standard position, with the rectangular edge parallel to co-ordinate axis. Other shapes are also possible but take longest time to process.

The mapping of a part of world co-ordinate scene to device co-ordinates is referred to as a *viewing transformation*. Sometimes,

two dimensional viewing transformations is simply referred to as *window to view port or windowing transformation*.



**Figure 3.12:** A viewing transformation using standard rectangles for the window and viewport.



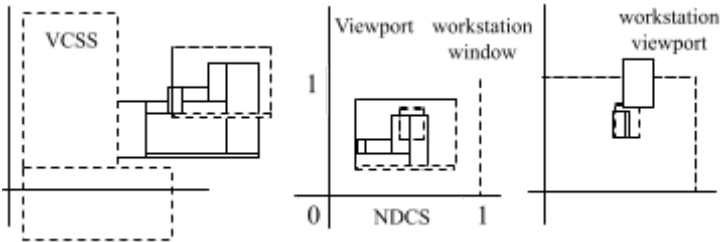
**Figure 3.13:** 2D viewing transformation pipeline

As in the real life, we see through a small window or the view finder of a camera, a computer-generated image often depicts a partial view of a large scene. Objects are placed into the scene by modeling transformations to a master coordinate system, commonly referred to as the world coordinate systems (WCS).

A rectangular window with its edges parallel to the axis of the WCS is used to select the portion of the scene for which an image is to be generated (displayed). Sometimes an additional coordinate system called the viewing coordinate system (VCS) is introduced to simulate the effort of moving or/and tilting the

camera. On the other hand, an image representing a view often becomes part of a larger image, like a photo on an album page, which models a computer monitor's display area.

Since monitor sizes differ from one system to another, we introduce a device-independent tool to describe the display area called normalized device coordinate system (NDCS) in which a unit (1X1) square whose lower left corner is at the origin of the coordinate system that defines the display area of the display device. A rectangular viewport with its edges parallel to the axes of the NDCS is used to specify a sub-region of the display area that embodies the image. The process that convert object coordinates in WCS to normalized device coordinate is called window to viewport mapping or normalization transformation.



**Figure 3.14:** *WCS, VCS, NDCS and workstation*

The process of mapping normalized device coordinates to discrete device coordinates is called workstation transformation, which is essentially a second window to viewport mapping, with a workstation window in the normalized device coordinate system and a workstation viewport in the device coordinate system. Collectively these two coordinate mapping operations are referred to as viewing transformation.

### 3.9 Window to Viewport Coordinate Transformation (Mapping)

*Window to Viewport Transformation* is the process of transforming two dimensional world-coordinate objects to device coordinates. Objects inside the world or clipping window are

mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

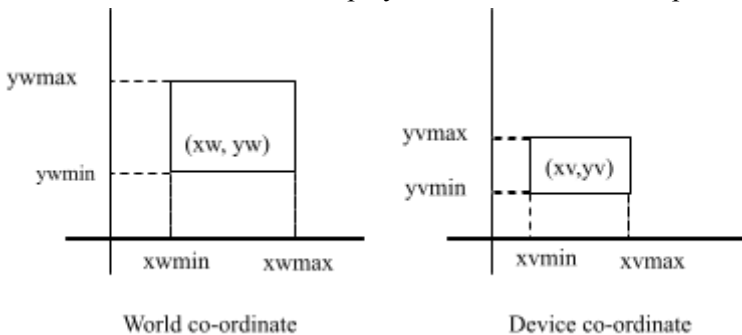
*Viewing transformation* is the mapping of coordinates of scene of world coordinates to device coordinate (screen pixels).

- **World coordinate** – It is the Cartesian coordinate w.r.t which we define the diagram, like  $X_{wmin}$ ,  $X_{wmax}$ ,  $Y_{wmin}$ ,  $Y_{wmax}$
- **Device Coordinate** – It is the screen coordinate where the objects are to be displayed, like  $X_{vmin}$ ,  $X_{vmax}$ ,  $Y_{vmin}$ ,  $Y_{vmax}$
- **Window** – It is the area on the world coordinate selected for display.
- **ViewPort** – It is the area on the device coordinate where graphics is to be displayed.

A point at position  $(x_w, y_w)$  in the window is mapped into position  $(x_v, y_v)$  in the associated viewport.

We transfer object description to normalized device coordinates using a transformation that maintains the same relative placement of objects in normalized space they had in viewing coordinates.

If a coordinate position is at the center of the viewing window for instance it will be displayed at the center of viewport.



**Figure 3.15:** Window to viewport mapping



## Mathematical Calculation of Window to Viewport:

It may be possible that the size of the Viewport is much smaller or greater than the Window. In these cases, we have to increase or decrease the size of the Window according to the Viewport and for this, we need some mathematical calculations.

A window is specified by four world co-ordinates  $x_{wmin}$ ,  $x_{wmax}$ ,  $y_{wmin}$ ,  $y_{wmax}$ . A viewport is described by four device co-ordinate  $x_{vmin}$ ,  $x_{vmax}$ ,  $y_{vmin}$ ,  $y_{vmax}$ .

$(x_w, y_w)$ : A point on Window

$(x_v, y_v)$ : Corresponding point on Viewport

We have to calculate the point  $(x_v, y_v)$

Normalized point in window is,

$$\frac{x_w - x_{wmin}}{x_{wmax} - x_{wmin}}, \frac{y_w - y_{wmin}}{y_{wmax} - y_{wmin}}$$

Normalized point in viewport is,

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}}, \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}$$

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

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

And

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

Solving the equation for viewport

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

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

where, scaling factor,

$$S_x = \frac{x_{v_{max}} - x_{v_{min}}}{x_{w_{max}} - x_{w_{min}}}$$

$$S_y = \frac{y_{v_{max}} - y_{v_{min}}}{y_{w_{max}} - y_{w_{min}}}$$

Q) For window,  $X_{wmin} = 20$ ,  $X_{wmax} = 80$ ,  $Y_{wmin} = 40$ ,  $Y_{wmax} = 80$  and for viewport,  $X_{vmin} = 30$ ,  $X_{vmax} = 60$ ,  $Y_{vmin} = 40$ ,  $Y_{vmax} = 60$ . Now a point  $(X_w, Y_w)$  be  $(30, 80)$  on the window. Calculate the point on the viewport.

First of all, we calculate the scaling factor of x coordinate  $S_x$  and the scaling factor of y coordinate  $S_y$  using the above-mentioned formula.

$$S_x = (60 - 30) / (80 - 20) = 30 / 60$$

$$S_y = (60 - 40) / (80 - 40) = 20 / 40$$

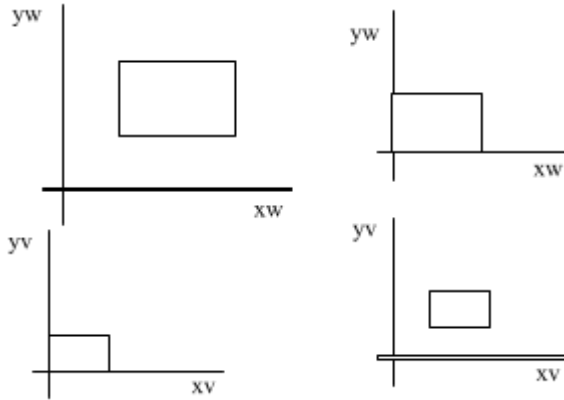
So, now calculate the point on the viewport  $(X_v, Y_v)$ .

$$X_v = 30 + (30 - 20) * (30 / 60) = 35$$

$$Y_v = 40 + (80 - 40) * (20 / 40) = 60$$

So, the point on window  $(X_w, Y_w) = (30, 80)$  will be  $(X_v, Y_v) = (35, 60)$  on viewport.

## Steps (for Window to Viewport Transformation):



**Figure 3.16:** Window to viewport transformation

1. The object together with its window is translated until the lower left corner of the window is at origin.
2. Object and window are scaled until window has dimension of viewport.

Perform a scaling transformation using a fixed-point position of  $(x_{wmin}, y_{wmin})$  that scales the window area to the size of the viewport

3. Again, translate to move viewport to its correct position.

### Viewing Transformation:

1. Translate window to origin by

$$T_x = -x_{wmin}$$

$$T_y = -y_{wmin}$$

2. Scale window such that its size is matched to viewport.

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

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

3. Retranslate it by

$$T_x = x_{vmin}$$

$$T_y = y_{vmin}$$

$$\text{Composite matrix (CM)} = T_v \times S_{wv} \times T_w$$

$T_w$  = Translate window to origin =

$$\begin{bmatrix} 1 & 0 & -x_{wmin} & 0 & 1 & -y_{wmin} & 0 & 0 & 1 \end{bmatrix}$$

$S_{wv}$  = Scaling of window to viewport

$$= \begin{bmatrix} s_x & 0 & 0 & 0 & s_y & 0 & 0 & 0 & 1 \end{bmatrix}$$

$T_v$  = Translate viewport to original position

$$= \begin{bmatrix} 1 & 0 & x_{vmin} & 0 & 1 & y_{vmin} & 0 & 0 & 1 \end{bmatrix}$$

**Q.** Consider the window is located from  $X_{wmin} = 20$ ,  $X_{wmax} = 80$ ,  $Y_{wmin} = 40$ ,  $Y_{wmax} = 80$ , and a point is located in 30, 80. Identify the new location of the point in the view port considering viewport size  $X_{vmin} = 30$ ,  $X_{vmax} = 60$ ,  $Y_{vmin} = 40$ ,  $Y_{vmax} = 60$ .

**Solution:**

$$X_{wmin} = 20$$

$$X_{wmax} = 80$$

$$Y_{wmin} = 40$$

$$Y_{wmax} = 80$$

$$X_{vmin} = 30$$

$$X_{vmax} = 60$$

$$Y_{vmin} = 40$$

$$Y_{vmax} = 60$$

$$S_x = \frac{X_{vmax} - X_{vmin}}{X_{wmax} - X_{wmin}}$$

$$S_y = \frac{Y_{vmax} - Y_{vmin}}{Y_{wmax} - Y_{wmin}}$$

$$X_v = X_{vmin} + (X_w - X_{wmin})S_x$$

$$= 30 + (80 - 20) \times \frac{60 - 30}{80 - 40} = 30 + 5 \times 5 = 55$$

$$Y_v = Y_{vmin} + (Y_w - Y_{wmin})S_y$$

$$= 40 + (80 - 40) \times \frac{60 - 40}{80 - 40} = 40 + 4 \times 5 = 60$$

**By 2<sup>nd</sup> method**

1. Translate window to origin

$$T_x = -X_{wmin}$$

$$T_y = -Y_{wmin}$$

2. Scale window such that its size is matched to viewport

$$S_x =$$

$$S_y =$$

3. Retranslate it

$$T_x = X_{vmin}$$

$$T_y = Y_{vmin}$$

$$C.M. =$$

$$=$$

$$=$$

$$==$$

$$P^1 = C.M. \cdot P$$

$$=$$

$$==$$

$$P^1 = (35, 60)$$

## 3.10 Clipping Operations

### 3.10.1 Introduction

*Clipping* is the process that cuts off the portions of a picture which are outside of the specified region. The specified region against which an object is clipped is called a '*clip window*' or '*clipping region*'. *Clipping algorithm* is procedure that identifies those portions of a picture that are either inside or outside of the window.

During clipping, we examine each portion of the image to find out whether it lies completely inside or completely outside or crosses the window boundary. If it is completely inside, then it is displayed, if it is completely outside then it is discarded. If it crosses the boundary, then we have to determine the point of intersection and display only the portions which are inside the boundary.

### 3.10.2 Applications of clipping

---

Clipping is used for extracting part of a defined scene to view and identify visible surfaces. It is used for drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing or duplicating. Clipping is used for creating objects using solid-modeling procedures. Clipping algorithm can be applied to window coordinates, so that only the content of the window interior is mapped to device coordinates. Alternatively, the complete world co-ordinate picture can be mapped first to device coordinate or NDC, then clipped against viewport boundaries.

WC clipping removes those primitives outside the window from further consideration, thus eliminating the processing necessary to transform those primitives to device space. Viewport clipping on the other hand, can reduce calculations by allowing concatenation of viewing and geometric transformation matrices. But viewport clipping does require that the transformation to device co-ordinates be performed for all objects, including those outside the window area.

### 3.10.3 Some Primitive Types of Clipping

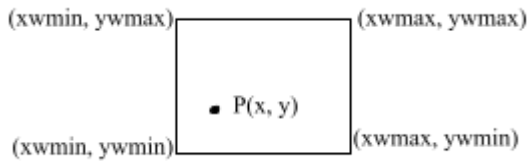
---

- Point clipping
- Line clipping (straight line segment)
- Area clipping (polygon)
- Curve clipping
- Text clipping

#### 1. Point Clipping

Point clipping is the process that determines whether the point is inside or outside of the window and discards if it is outside and displays if it is inside the window. Assuming that the clip window is a rectangle in standard position, we save a point  $P(x, y)$  for display, if the following inequalities are satisfied:

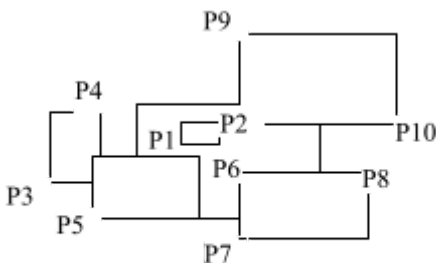
$$xw_{min} \leq x \leq xw_{max} \quad yw_{min} \leq y \leq yw_{max}$$



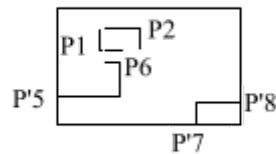
**Figure 3.17:** Point clipping

Where the the edges of the clip window can be either world coordinate window boundary ( $xw_{min}$ ,  $yw_{min}$ ,  $xw_{max}$ ,  $yw_{max}$ ) or viewport window boundary ( $xv_{min}$ ,  $yv_{min}$ ,  $xv_{max}$ ,  $yv_{max}$ ). If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

## 2. Line Clipping



**Fig(a):** Before Clipping



**Fig(b):** After clipping

**Figure 3.18:** Line clipping

The process of cutting of the lines which are outside the window is known as line clipping. A line-clipping procedure involves several parts. First, a given segment is tested to determine whether it lies completely inside the clipping window. If it does not, it is tested to determine whether it lies completely outside the window. Finally, if we cannot identify a line as complete inside or outside, we must perform intersection calculation with one or more clipping boundaries.

We process line through the 'Inside-outside' test by checking the line endpoints. A line with both endpoints outside anyone of the clip boundaries (line  $P_3$  to  $P_4$ ) is outside the window. All other

lines cross one or more clipping boundaries and may require calculation of multiple intersection points. For a line segment with end points  $(x_1, y_1)$  and  $(x_2, y_2)$  and one or both end points outside the clipping rectangle, the parametric representation could be used.

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

$$y = y_1 + u(y_2 - y_1), \text{ where } 0 \leq u \leq 1$$

If the value of  $u$  is outside the range 0 to 1, the line does not enter the interior of the window at that boundary. If the value of  $u$  is within the range from 0 to 1, the line segment does indeed cross into the clipping area. This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed.

### **3.10.3.1 Cohen-Sutherland Line Clipping**

---

Cohen-Sutherland line clipping algorithm is one of the oldest and most popular procedures. It uses bit operation to perform this test. It speeds up the processing of line segments by performing tests that reduce the no. of intersection that must be calculated.

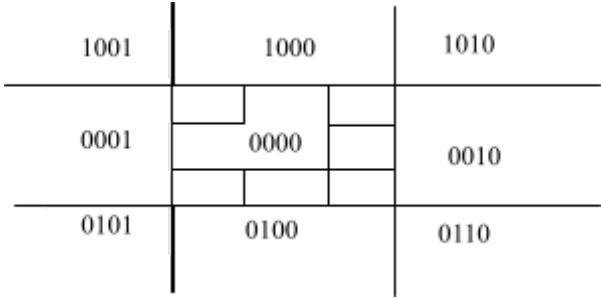
This algorithm divides two dimensional space into nine parts, eight outside regions and one inside region. Each of the window is assigned a four bit code to identify that region as shown in the figure 3.19.

For each end-point of a line, a four digits binary code (called a region code) is assigned to identify the location relative to boundary. Lower order bit (bit1) is set to '1' if end point is at the left side of the window, else set to '0'. Bit 2 is set to '1' if end point lies at right side else set '0'. Bit 3 is set to '1' if end point lies bottom, else set '0'. Bit 4 is set to '1' if end point lies top, else set '0'.

The region code or outcode for endpoints must be recalculated on each iteration after the clipping. This algorithm uses a divide and conquer strategy. The line segment's endpoints are tested if the line can be trivially accepted or rejected. If the line is not trivially accepted or rejected, an intersection point of line



with a window edge is determined and the trivial accept or reject is *repeated*. This process is continued until the line is not accepted.



**Figure 3.19:** *Clip window and region code*

### Algorithm

- Step1: Assign Region Code (TBRL)
- Step2: Establish region code for all line end points.
 

Bit 1 is set to '1' if  $x < x_{\min}$  else set to '0'.

Bit 2 is set to '1' if  $x > x_{\max}$  else set to '0'.

Bit3 is set to '1'  $y < y_{\min}$  else set '0'

Bit 4 is set to '1'  $y > y_{\max}$  else set to '0'
- Step3: Determine whether line is completely inside or outside window using test.
 

a) If both end points have region code ' 0000' line is completely inside.

b) If logical AND of end points of a line not '0000' line is completely outside.
- Step 4. If both condition of step 3 fails. i.e., Logical AND give '0000' we need to find the intersection with window boundary.
 

Here,

$m =$

a) If bit 1 is 1, line intersection with left boundary, so,
 

$y_i = y_1 + m(x - x_l)$  where  $x = x_{\min}$

b) If bit 2 is 1, line intersect with right boundary, so,
 

$y_i = y_1 + m(x - x_l)$  where  $x = x_{\max}$

- c) If bit 3 is 1, line intersect with lower boundary, so,  
 $x_i = x_1 + (y - y_1)$  where  $y = y_{\min}$
- d) If bit 4 is 1, line intersects with upper boundary, so,

$$x_i = x_1 + (y - y_1) \quad \text{where } y = y_{\max}$$

Here,  $x_i$  and  $y_i$  are x, y intercepts for that line, update

Step5: Repeat step 2 and 4 till completely accepted

### 3.10.3.2 Liang-Barsky Line Clipping

---

Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clipping window. It is faster line clipper method than Cohen Sutherland method. The parametric equation of a line segment can be written in the form,

$$x = x_1 + u \Delta x$$

$$y = y_1 + u \Delta y, 0 \leq u \leq 1$$

$$\text{where } \Delta x = x_2 - x_1, \Delta y = y_2 - y_1$$

Using these parametric equations, Cyrus and Beck developed an algorithm that is generally more efficient than the Cohen-Sutherland algorithm. Later, Liang and Barsky independently devised an even faster parametric line-clipping algorithm. Following the Liang-Barsky approach, we first write the point clipping condition in the parametric form,

$$x_{\min} \leq x_1 + u \Delta x \leq x_{\max}$$

$$y_{\min} \leq y_1 + u \Delta y \leq y_{\max}$$

Now, we can write as,

$$-u \Delta x \leq x_1 - x_{\min}$$

$$u \Delta x \leq x_{\max} - x_1$$

$$-u \Delta y \leq y_1 - y_{\min}$$

$$u \Delta y \leq y_{\max} - y_1$$

Each of these four inequalities can be expressed  $u.p_k \leq q_k$ ,  $k = 1, 2, 3, 4$ , where parameters  $p$  and  $q$  are defined as

$k = 1$  (is the line inside the left boundary)  $p_1 = -\Delta x$ ,  $q_1 = x_1 - x_{wmin}$

$k = 2$  (is the line inside the right boundary)  $p_2 = \Delta x$ ,  $q_2 = x_{wmax} - x_1$

$k = 3$  (is the line inside the bottom boundary)  $p_3 = -\Delta y$ ,  $q_3 = y_1 - y_{wmin}$

$k = 4$  (is the line inside the top boundary)  $p_4 = \Delta y$ ,  $q_4 = y_{wmax} - y_1$

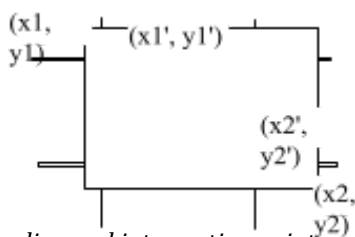
When  $P_k < 0$ , the line proceeds from the outside to inside of clipping boundary. When  $P_k > 0$ , the line proceeds from inside to outside

### Trivial Rejection

The line with  $p_k = 0$  for some  $k$  and one  $q_k < 0$  for these  $k$  lies outside the region, so, the line is rejected. For line with  $p_k = 0$  for some  $k$  and all  $q_k \geq 0$  for those  $k$ , line is parallel to one of the clipping boundary and some portion of the line is inside. The intersection points are found out at parallel boundary line.

For line with  $p_k \neq 0$  to find the intersection with boundary, the parameters  $r_k$  is given calculated by,

$$r_k = \frac{q_k}{p_k}$$



**Figure 3.20:** Clip window, line and intersection points

The value of  $r_k$  becomes candidate for  $u_1$  if  $p_k < 0$ .  $u_1$  is for intersection with the boundaries to which line enters the boundary. The value of  $u_1$  is maximum value between 0 and values of  $r_k$ . The clipped line will be

$$x_1' = x_1 + u_1 \Delta x$$

$$y_1' = y_1 + u_1 \Delta y$$

The value of  $r_k$  becomes candidate for  $u_2$  if  $p_k \geq 0$ .  $u_2$  is for intersection with the boundaries to which line leaves the boundary. It is minimum value between values of  $r_k$  and 1

$$x_2' = x_1 + u_2 \Delta x$$

$$y_2' = y_1 + u_2 \Delta y$$

### Algorithm

- 1) Assign the end points of line  $(x_1, y_1)$  and  $(x_2, y_2)$  and size of the window  $x_{wmin}, y_{wmin}, x_{wmax}, y_{wmax}$ .
- 2) Calculate  $p_k$  and  $q_k$  for  $k = 1, 2, 3, 4$ 

$$p_1 = -\Delta x \quad \text{and} \quad q_1 = x_1 - x_{wmin}$$

$$p_2 = \Delta x \quad \text{and} \quad q_2 = x_{wmax} - x_1$$

$$p_3 = -\Delta y \quad \text{and} \quad q_3 = y_1 - y_{wmin}$$

$$p_4 = \Delta y \quad \text{and} \quad q_4 = y_{wmax} - y_1$$

$p_1$  and  $q_1$  are for left,  $p_2$  and  $q_2$  are for right,  $p_3$  and  $q_3$  are for bottom,  $p_4$  and  $q_4$  are for top
- 3) If any  $p_k = 0$ , then the line is parallel to the clipping boundary and now test the corresponding  $q_k$ .
  - (I) If corresponding  $q_k < 0$  then line is outside the boundary, discard the line and stop
  - (II) If corresponding  $q_k \geq 0$  the line is inside the parallel boundary, calculate the intersection point

If  $p_k \neq 0$  then go to step 4
- 4) Calculate  $r_k = \frac{q_k}{p_k}$  for  $k = 1, 2, 3, 4$ 
  - (I) If  $p_k < 0$ , find initial time  $u_1$  by selecting relative  $r_k$  as  $u_1 = \max \{r_k, 0\}$
  - (II) If  $p_k > 0$ , find final time  $u_2$  by selecting relative  $r_k$  as  $u_2 = \min \{r_k, 1\}$

If  $p_k < 0$  then line proceeds from outside to inside boundary.  
 Calculate  $u_1 = \max(0, \left\{ r_k : r_k = \frac{q_k}{p_k} \right\})$  to determine intersection point with the possible extended clipping boundary k and obtain a new starting point for the line at  $u_1$ .

If  $p_k > 0$  then line proceeds from inside to outside the boundary. Calculate  $u_2 = \min(1, \left\{ r_k : r_k = \frac{q_k}{p_k} \right\})$  to determine intersection point with extended clipping boundary k and obtain a new point at  $u_2$

- 5) If  $u_1 > u_2$ , then the line is totally outside so discard and stop
- 6) If  $u_1 < u_2$ , then calculate end points of clipped line  $I_1'(x_1', y_1')$  and  $I_2'(x_2', y_2')$

$$x_1' = x_1 + u_1 \Delta x$$

$$y_1' = y_1 + u_1 \Delta y$$

$$x_2' = x_1 + u_2 \Delta x$$

$$y_2' = y_1 + u_2 \Delta y$$

Display the segment  $I_1'(x_1', y_1')$  and  $I_2'(x_2', y_2')$

- 
1. *Consider a triangle  $A(0, 0)$ ,  $B(1, 1)$ ,  $C(5, 2)$ . The triangle has to be rotated by an angle  $45^\circ$  about the point  $P(-1, -1)$ . What will be the co-ordinate of new triangle.*

**Solution:**

$$CM = T_{(-1,-1)} R_{45} T_{(1,1)}$$

$$=$$

$$=$$

$$P' = C.M. \cdot P$$

$$=$$

$$=$$

$$A' = (-1, -1)$$

$$B' = (-1, 2-1)$$

$$C' =$$

The co-ordinates of new triangle are A'(-1, -1), B'(-1, 2-1), and C'

2. **Scale an object (4, 4), (3, 2), (5, 2) about a fixed point (4, 3) with scaling factor 2.**

**Solution:**

$$P' = C.M. \cdot P$$

$$C.M. = \begin{bmatrix} 1 & 0 & 4 & 0 & 1 & 3 & 0 & 0 & 1 \\ 1 & 0 & -4 & 0 & 1 & -3 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 4 & 0 & 1 & 3 & 0 & 0 & 1 \\ 2 & 0 & -8 & 0 & 2 & -6 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 0 & -8 & 0 & 2 & -6 & 0 & 0 & 1 \\ 2 & 0 & -4 & 0 & 2 & -3 & 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} 4 & 3 & 5 & 4 & 2 & 2 & 1 & 1 & 1 \end{bmatrix}$$

$$P' = C.M. \times P$$

$$= \begin{bmatrix} 2 & 0 & -4 & 0 & 2 & -3 & 0 & 0 & 1 \\ 4 & 3 & 5 & 4 & 2 & 2 & 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 8 & -4 & 6 & -4 & 10 & -4 & 8 & -3 & 4 & -3 & 1 & 1 & 1 \\ 4 & 2 & 6 & 5 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

So, scaled points are (4, 5), (2, 1) and (6, 1)

3. **Rotate the triangle (5,5), (7,3), (3,3) about fixed points (5,4) in counter clockwise by  $90^\circ$ .**

**Solution:**

$$C.M. = T_{(5,4)} \cdot R_{90} \cdot T_{(-5,-4)}$$

$$= \begin{bmatrix} 1 & 0 & 5 & 0 & 1 & 4 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos 90^\circ & -\sin 90^\circ & 0 & \sin 90^\circ & \cos 90^\circ & 0 & 0 & 0 & 1 \\ 1 & 0 & -5 & 0 & 1 & -4 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & 9 & 1 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Now } P' = C.M. \cdot P$$

$$= \begin{bmatrix} 0 & -1 & 9 & 1 & 0 & -1 & 0 & 0 & 1 \\ -5 & 7 & 3 & 5 & 3 & 3 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -5 & 9 & -3 & 9 & -3 & 9 & 8 & -1 & 7 & -1 & 3 & 1 & 1 & 1 \end{bmatrix}$$

$$= [4 \ 6 \ 6 \ 4 \ 6 \ 2 \ 1 \ 1 \ 1]$$

∴ New co-ordinates are (4,4), (6,6) (6,2)

4. **Reflect an object (2, 3), (4, 3), (4, 5) about line  $y = x + 1$ .**

**Solution:**

Here,  $m = 1$

$c = 1$

$$C.M. = T' R_{\theta} R_{f_x} R_{\theta} T$$

$$T = [1 \ 0 \ 0 \ 0 \ 1 \ -1 \ 0 \ 0 \ 1],$$

$$R_{\theta} = \begin{bmatrix} \cos 45^\circ & \sin 45^\circ & 0 & -\sin 45^\circ & \cos 45^\circ & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{f_y} = [1 \ 0 \ 0 \ 0 \ -1 \ 0 \ 0 \ 0 \ 1]$$

$$R_{\theta}' = \begin{bmatrix} \cos 45^\circ & -\sin 45^\circ & 0 & \sin 45^\circ & \cos 45^\circ & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T' = [1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1]$$

$$P' = C.M. \times P$$

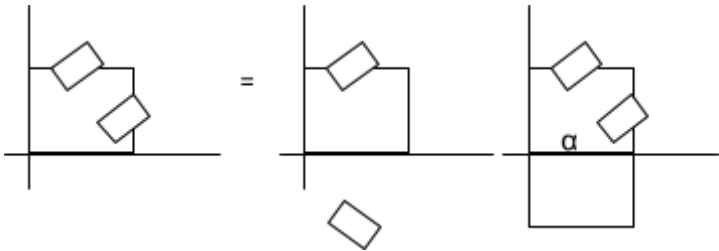
$$= [0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1][2 \ 4 \ 4 \ 3 \ 3 \ 5 \ 1 \ 1 \ 1] =$$

$$[2 \ 2 \ 4 \ 3 \ 5 \ 5 \ 1 \ 1 \ 1]$$

Required points are (4, 3), (4, 5), (6, 5)

5. **The reflection along the line  $y = x$  is equivalent to the reflection along the  $x$  axis followed by counter clockwise rotation by  $\alpha$  degree. Find the angle  $\alpha$ .** [2071 Chaitra]

**Solution:**



**Steps for reflection about line  $y = x$**

- i) Rotate about origin in clockwise direction by  $45^\circ$

- ii) Reflection about x –axis
- iii) Rotate in anticlockwise direction in  $45^\circ$

$$\begin{aligned}
 Rf_{(y=x)} &= R(\theta)^{-1} \cdot Rf_x \cdot R(\theta) \\
 &= [\cos 45 \quad -\sin 45 \quad 0 \quad \sin 45 \quad \cos 45 \quad 0 \quad 0 \quad 0 \quad 1] \\
 &[1 \quad 0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 1] \\
 &[\cos 45 \quad \sin 45 \quad 0 \quad -\sin 45 \quad \cos 45 \quad 0 \quad 0 \quad 0 \quad 1] \\
 &= [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1] \dots\dots\dots(i)
 \end{aligned}$$

Again, the matrix for the reflection along x-axis followed by  $\alpha$  degree

$$\begin{aligned}
 &= [\cos \alpha \quad -\sin \alpha \quad 0 \quad \sin \alpha \quad \cos \alpha \quad 0 \quad 0 \quad 0 \quad 1] \\
 &[1 \quad 0 \quad 0 \quad 0 \quad -1 \quad 0 \quad 0 \quad 0 \quad 1] \\
 &= [\cos \alpha \quad \sin \alpha \quad 0 \quad \sin \alpha \quad -\cos \alpha \quad 0 \quad 0 \quad 0 \quad 1] \dots\dots\dots(ii)
 \end{aligned}$$

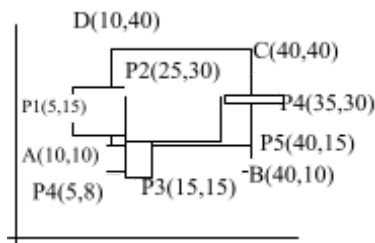
Now equating matrix (1) and (2)

$$\cos \alpha = 0,$$

$$\sin \alpha = 1 \quad -\cos \alpha = 0 \quad \alpha = 90^\circ$$

The angle  $\alpha$  is  $90^\circ$

6. *Given a clipping window A(10, 10), B(40,10), C(40, 40) and D(10, 40). Using Cohen-Sutherland line clipping algorithm find region code of each end points of lines  $P_1P_2$ ,  $P_3P_4$  and  $P_5P_6$  where co-ordinates are  $P_1(5,15)$ ,  $P_2(25, 30)$ ,  $P_3(15, 15)$ ,  $P_4(35, 30)$ ,  $P_5(5, 8)$  and  $P_6(40, 15)$ . Also find clipped lines using above parameters. [2071 Shrawan]*



**Solution:**



### Step 1: Assign the region code.

For  $P_1 (5, 15)$

$5 < 10$  is true, so, bit 1 = 1

$5 > 40$  is false, so, bit 2 = 0

$15 < 10$  is false, so, bit 3 = 0

$15 > 40$  is false, so, bit 4 = 0

Region code for

$P_1 (5, 15) = 0\ 0\ 0\ 1$

For  $P_2 (25, 30)$

$25 < 10$  is false, so, bit 1 = 0

$25 > 40$  is false, so, bit 2 = 0

$30 < 10$  is false, so, bit 3 = 0

$30 > 40$  is false, so, bit 4 = 0

Region code for  $P_2 (25, 30) = 0\ 0\ 0\ 0$

For  $P_3 (15, 15)$

$15 < 10$  is false, so, bit 1 = 0

$15 > 40$  is false, so, bit 2 = 0

$15 < 10$  is false, so, bit 3 = 0

$15 > 40$  is false, so, bit 4 = 0

Region code for  $P_3 (15, 15) = 0\ 0\ 0\ 0$

For  $P_4 (35, 30)$

$35 < 10$  is false, so, bit 1 = 0

$35 > 40$  is false, so, bit 2 = 0

$30 < 10$  is false, so, bit 3 = 0

$30 > 40$  is false, so, bit 4 = 0

Region code for  $P_4 (35, 30) = 0\ 0\ 0\ 0$

For  $P_5 (5, 8)$

$5 < 10$  is true, so, bit 1 = 1

$5 > 40$  is false, so, bit 2 = 0

$8 < 10$  is true, so, bit 3 = 1

$8 > 40$  is false, so, bit 4 = 0

Region code for  $P_5$  (5, 8) = 0 1 0 1

For  $P_6$  (40, 15)

$40 < 10$  is false, so, bit 1 = 0

$40 > 40$  is false, so, bit 2 = 0

$15 < 10$  is false, so, bit 3 = 0

$15 > 40$  is false, so, bit 4 = 0

Region code for  $P_6$  (40, 15) = 0 0 0 0

Now, for line  $P_1 P_2$ ,  $P_1(5, 15)$ ,  $P_2(25, 30)$

Region code for  $P_1$  = 0001

Region code for  $P_2$  = 0000

- i. Both end point have not region code '0000', line is not completely inside.
- ii. The logical AND of end points of the line  $P_1 P_2$  is '0000', so line is not completely outside.

So now,

$m = = = =$

If bit1 is 1, line intersects with left boundary.

So,

$$y_i = y_1 + m(x_{min} - x_1) = 15 + \frac{3}{4}(10 - 5) = 18.75$$

$$y_i = 18.75 = 19$$

$$x_i = 10$$

Region code is,

$10 < 10$  is false, so, bit 1 = 0

$10 > 40$  is false, so, bit 2 = 0

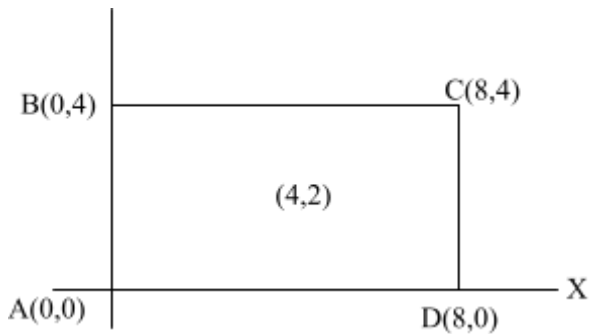
$19 < 10$  is false, so, bit 3 = 0

$19 > 40$  is false, so, bit 4 = 0

So, the required end points of the line are  $P_1'$  (10, 19) and  $P_2'$  (25, 30)

7. Find the transformation matrix that transforms the rectangle ABCD whose center is at (4, 2) is reduced to half of its size, the center will remain same. The co-ordinate of ABCD are A(0, 0), B(0, 4), C(8, 4) and D(8, 0). Find co-ordinate of new square. Also derive the transformation matrix to convert this rectangle to square. [2072 kartik]

**Solution:**



The transformation matrix that transformations the rectangle ABCD whose center is (4, 2) is reduced to half of its size keeping the center same is,

$$P' = \text{C.M.} * P$$

$$\text{C.M.} = [1 \ 0 \ 4 \ 0 \ 1 \ 2 \ 0 \ 0 \ 1] \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & -4 & 0 & 1 & -2 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{C.M} = [1 \ 0 \ 4 \ 0 \ 1 \ 2 \ 0 \ 0 \ 1] \begin{bmatrix} \frac{1}{2} & 0 & -2 & 0 & 1 & -2 & 0 & 0 & 1 \\ \frac{1}{2} & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} =$$

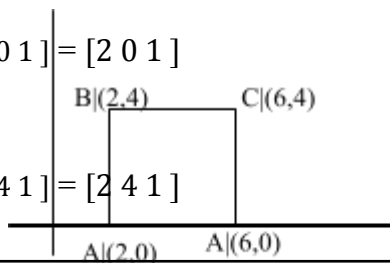
Now to find new coordinate of new square,

$$A' = \text{C.M.} \times A$$

$$\begin{bmatrix} \frac{1}{2} & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$$

$$B' = \text{C.M.} \times B$$

$$\begin{bmatrix} \frac{1}{2} & 0 & 2 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix}$$



$$C' = C.M. \times C$$

$$\left[ \frac{1}{2} \ 0 \ 2 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \right] [8 \ 4 \ 1] = [6 \ 4 \ 1]$$

$$D' = C.M. \times D$$

$$= \left[ \frac{1}{2} \ 0 \ 2 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \right] [8 \ 0 \ 1] = [6 \ 0 \ 1]$$

The coordinates of new square are (2,0), (2,4), (6,4), and (6,0)

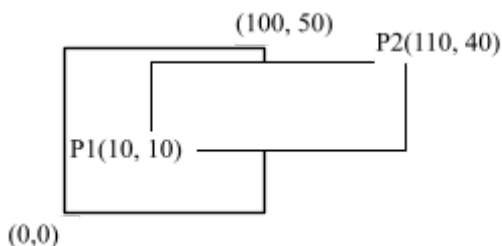
Steps to get the transformation matrix to convert this rectangle to square are;

- i. Translate by (-4, -2)
- ii. Scale by  $S_x=1/2$
- iii. Rotate by (4, 2)

$$\begin{aligned} C.M. &= \begin{bmatrix} 1 & 0 & 4 & 0 & 1 & 2 & 0 & 0 & 1 \\ 1 & 0 & -4 & 0 & 1 & -2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

8. *Use Liang-Barsky clipping method to clip a line starting from P1 (10, 10) and ending at P2(110, 40) against the window having its lower corner at (0, 0) and upper right corner at (100, 50)*

**Solution:**



K	$p_k$	$q_k$	$r_k$
1	$-\Delta x$ $= -(110-10)$ $= -100$	$x_1 - x_{wmin}$ $= 10-0$ $= 10$	$r_1 = 10/-100$ $= -1/10$ $= \text{candidate for } u_1$

K	$p_k$	$q_k$	$r_k$
	i. e., $p_k < 0$		
2	$\Delta x$ = (110-10) = 100 i. e., $p_k > 0$	$x_{wmax} - x_1$ = 100-10 = 90	$r_2 = 90/100$ = 0.9 = candidate for $u_2$
3	$-\Delta y$ = -(40-10) = -30 i. e. $P_k < 0$	$y_1 - y_{wmin}$ = 10-0 = 10	$r_3 = 10/-30$ = -1/3 = candidate for $u_1$
4	$\Delta y$ = (40-10) = 30 i.e. $P_k > 0$	$y_{wmax} - y_1$ = 50-10 = 40	$r_4 = 40/30$ = 4/3 = candidate for $u_2$

$$u_1 = \max \{-1/10, -1/3, 0\}$$

$$u_2 = \min \{0.9, 4/3, 1\}$$

We take  $u_1 = 0$  and  $u_2 = 0.9$

Clipped line

$$x_1' = x_1 + u_1 \Delta x$$

$$y_1' = y_1 + u_1 \Delta y$$

$$x_2' = x_1 + u_2 \Delta x$$

$$y_2' = y_1 + u_2 \Delta y$$

$$x_1' = 10 + 0 \times 100 = 10$$

$$y_1' = 10 + 0 \times 30 = 10$$

$$x_2' = x_1 + u_2 \times 100 = 10 + 0.9 \times 100 = 100$$

$$y_2' = y_1 + u_2 \times 100 = 10 + 0.9 \times 30 = 37$$

9. ***State the condition of point clipping, perform clipping operation for the following using Liang-Barskey line clipping algorithm.*** [2070 Chaitra]

**Solution:**

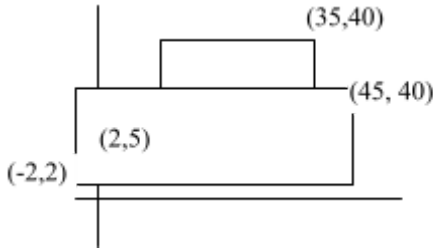
Clipping window:  $(x_{\min}, y_{\min}) = (2, 5)$

And  $(x_{\max}, y_{\max}) = (35, 50)$

Line  $(x_1, y_1) = (-2, 2)$  and  $(x_2, y_2) = (45, 40)$

$$\Delta x = x_2 - x_1 = 45 - (-2) = 47$$

$$\Delta y = y_2 - y_1 = 40 - 2 = 38$$



k	$p_k$	$q_k$	$R_k =$
0	$\Delta x = -47, p_k < 0$	$x_1 - x_{\min} = -4$	$0.0851(u_1)$
1	$\Delta x = 47$	$x_{\max} - x_1 = 37$	$0.787(u_2)$
2	$-\Delta y = -38$	$y_1 - y_{\min} = -3$	$0.0789(u_1)$
3	$\Delta y = 38$	$y_{\max} - y_1 = 48$	$1.263(u_2)$

$$u_1 = \max(0, r_k)$$

$$= \max(0, 0.0851, 0.0789)$$

$$= 0.0851$$

$$u_2 = \min(1, r_k) = \min(1, 0.787, 1.263) = 0.787$$

$$x_1' = x_1 + u_1 \Delta x = -2 + 0.0851 \times 47 = 1.997 \approx 2$$

$$y_1' = y_1 + u_1 \Delta y = 2 + 0.0851 \times 38 = 5$$

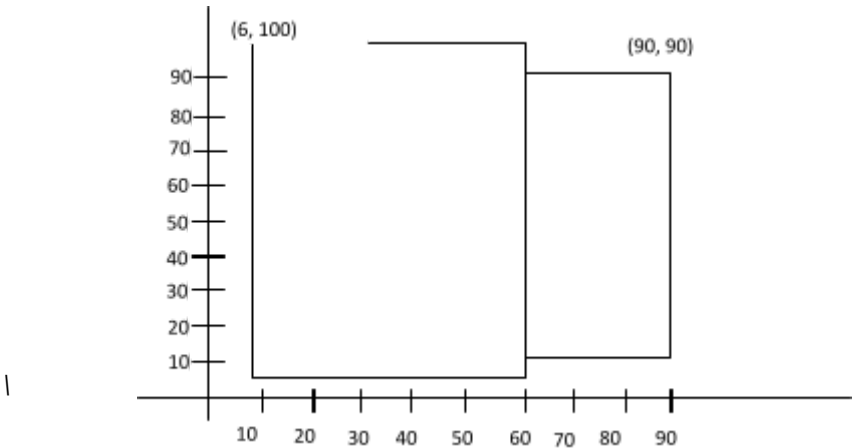
$$x_2' = x_1 + u_2 \Delta x = -2 + 0.787 \times 47 = 35$$

$$y_2' = y_1 + u_2 \Delta y = 2 + 0.787 \times 38 = 32$$

Required points are  $(2, 5)$  and  $(35, 32)$

- 10. Use Liang-Barsky line clipping algorithm to clip a line starting from  $(6, 100)$  and ending at  $(60, 5)$  against the**

*window having its lower left corner at (10, 10) and upper right corner at (90, 90).*
[2075 Ashwin]



**Solution:**

Clipping window  $(x_{min}, y_{min}) = (10, 10)$

$(x_{max}, y_{max}) = (90, 90)$

$(x_1, y_1) = (6, 100)$  and  $(x_2, y_2) = (60, 5)$

$$\Delta x = x_2 - x_1 = 60 - 6 = 54$$

$$\Delta y = y_2 - y_1 = 5 - 100 = -95$$

k	$p_k$	$q_k$	$r_k=q_k/p_k$
0	$-\Delta x$ $= -54 \ (p_k<0)$	$x_1 - x_{min} = 6-10 = -4$	$0.074(u_1)$
1	$\Delta x = 54$	$x_{max} - x_1 = 90-6 = 84$	$1.55(u_2)$
2	$-\Delta y = 95$	$y_1 - y_{min} = 100-10 = 90$	$0.947(u_2)$
3	$\Delta y = -95$	$y_{max} - y_1 = 90-100 = -10$	$0.105(u_1)$

$$u_1= \max(0, r_k) = 0.105$$

$$u_2= \min(0, r_k) =0.947$$

$$x_1' = x_1 + u_1\Delta x = 6 + 0.105\times 54 = 11.67 \approx 12$$

$$y_1' = y_1 + u_1\Delta y = 100 + 0.105\times (-94) = 90.025 \approx 90$$

$$x_2' = x_1 + u_2 \Delta x = 6 + 0.947 \times 54 = 57.118 \approx 57$$

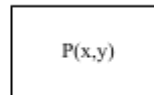
$$y_2' = y_1 + u_2 \Delta y = 100 + 0.947 \times (-94) = 10.035 \approx 10$$

Required points are (12, 90) and (57, 10)

- 11. Write down the condition for point clipping. Find the clipped region in window of diagonal vertex (10, 10) and (100, 100) for line  $P_1(5, 120)$  and  $P_2(80, 7)$  using Liang-Barsky line clipping method.** [2072 kartik]

**Solution:**

Assuming that the clip window is a rectangle in standard position, we save a point  $P(x, y)$  for display, if the following inequalities are satisfied.



- The edges of the clip window can be either the window boundary or viewport boundaries.
- If any one of these four inequalities is not satisfied, the point is clipped i.e., not saved for display.

Here,

$$x_{wmin} = 10$$

$$y_{wmin} = 10$$

$$x_{wmax} = 100$$

$$y_{wmax} = 100$$

$$(x_1, y_1) = (5, 120)$$

$$(x_2, y_2) = (80, 7)$$

Now, we have to find out the value of  $u_1$  and  $u_2$  by calculating  $p_k, q_k, r_k$  from  $k = 1$  to 4.

Then, the clipped line will be

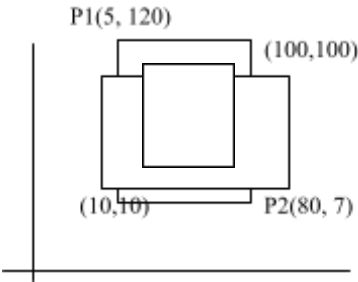
$$x_1' = x_1 + u_1 \Delta x$$



$$y_1' = y_1 + u_1 \Delta y$$

$$x_2' = x_1 + u_2 \Delta x$$

$$y_2' = y_1 + u_2 \Delta y$$



k	$p_k$	$q_k$	$r_k =$
1	$-\Delta x = -(80-5) = -75$ i.e., $p_k < 0$	$x_1 - x_{wmin}$ $(5-10) = -5$	$r_1 =$ $(u_1)$
2	$\Delta x$ $= -(80-5)$ $= 75$ i.e., $p_k > 0$	$x_{wmax} - x_1$ $= 100 - 5$ $= 95$	$r_2 =$ $(u_2)$
3	$\Delta y$ $= -(7-120)$ $= 113$ i.e., $p_k > 0$	$y_1 - y_{wmin}$ $= (120-10)$ $= 110$	$r_3 =$ $(u_2)$
	$\Delta y$ $= -113$ i.e., $p_k < 0$	$y_{wmax} - y_1$ $= 100 - 120$ $= -20$	$r_4 =$ $(u_1)$

Now,

$$u_1 = \max(0, r_k)$$

$$= \max()=$$

$$u_2 = \min(1, r_k)$$

$$= \min()=$$

$$x_1' = x_1 + u_1 \Delta x$$

$$= 5 + \times 75 = 18.27$$

$$y_1' = y_1 + u_1 \Delta y$$

$$= 120 + \times (-113) = 100$$

$$x_2' = x_1 + u_2 \Delta x$$

$$= 5 + \times 75 = 78$$

$$y_2' = y_1 + u_2 \Delta y$$

$$= 120 + \times (-113) = 10$$

$$P_1'(x_1', y_1') = P_1'(18, 27, 100)$$

$$P_2'(x_2', y_2') = P_2'(78, 10)$$

12. Rotate a triangle  $A(5,6)$ ,  $B(6,2)$  and  $C(4,1)$  by 45 degree about an arbitrary point  $(3,3)$ . [2076 Ashwin]

**Solution:**

Composite matrix

$$= T_{(3,3)} R_{(45)} T_{(-3,-3)}$$

=

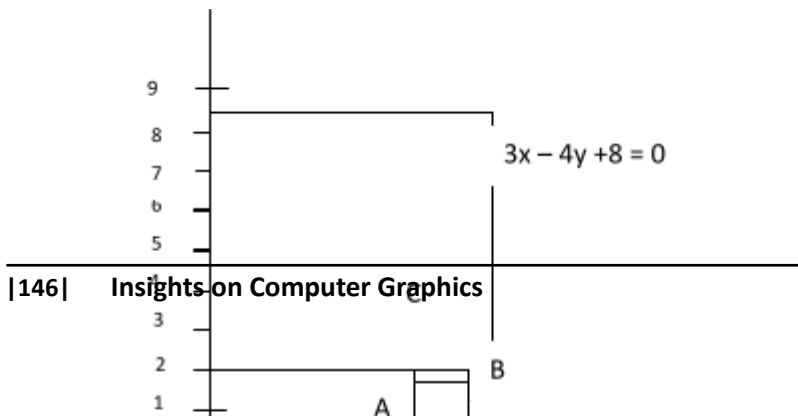
$$\text{Now, } P^1 = \text{C.M.} * P$$

= .

=

13. Reflect the triangle  $ABC$  about the line  $3x - 4y + 8 = 0$  the position vector of coordinate  $ABC$  as  $A(4, 1)$ ,  $B(5, 2)$  and  $C(4, 3)$ . [2078 karik and 2075 Ashwin]

**Solution:**



The arbitrary line about which the triangle ABC has to be reflected  
is  $3x - 4y + 8 = 0$

$$\text{i.e., } y = x + 2$$

Comparing with  $y = mx + c$

$$m =$$

$$c = 2$$

$$\theta = \tan^{-1}(m) = \tan^{-1} = 36.8698^\circ$$

Now, Steps for the transformations are,

Step 1: Transformation of Point P (0, 2) to origin O(0, 0)

Step 2: Rotation of line by an angle of  $\theta = -36.8698^\circ$  (-ve as it is moving in clockwise direction)

Step 3: Reflection of triangle about x-axis

Step 4: Reverse rotation of line to its original angle

Step 5: Inverse Translation of Point P to its original position

Now, composite matrix is given by,

$$\text{C.M.} = T^{-1} R_{\theta}^{-1} R_{fx} R_{\theta} T$$

$$T = \begin{bmatrix} 1 & 0 & t_x & 0 & 1 & t_y & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & -2 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{(-\theta)} =$$

$$\begin{bmatrix} \cos(-36.87) & -\sin(-36.87) & 0 & \sin(-36.87) & \cos(-36.87) & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{fx} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_{(\theta)} = \begin{bmatrix} & \end{bmatrix}$$

$$=$$

$$\begin{bmatrix} \cos(36.87) & -\sin(36.87) & 0 & \sin(36.87) & \cos(36.87) & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & t_x & 0 & 1 & t_y & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned}
C.M. &= \\
&[1\ 0\ 0\ 0\ 1\ 2\ 0\ 0\ 1][0.8\ -0.6\ 0\ 0.6\ 0.8\ 0\ 0\ 0\ 1][1\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 1] \\
&\quad [0.8\ 0.6\ -0.6\ 0.8\ 0\ 0\ 0\ 1][1\ 0\ 0\ 1\ -2\ 0\ 0\ 1] \\
&= \\
&[1\ 0\ 0\ 0\ 1\ 2\ 0\ 0\ 1][0.8\ -0.6\ 0\ 0.6\ 0.8\ 0\ 0\ 0\ 1][1\ 0\ 0\ 0\ -1\ 0\ 0\ 0\ 1] \\
&\quad [0.8\ 0.6\ -1.2\ -0.6\ 0.8\ -1.6\ 0\ 0\ 1] \\
&= \\
&[1\ 0\ 0\ 0\ 1\ 2\ 0\ 0\ 1][0.8\ -0.6\ 0\ 0.6\ 0.8\ 0\ 0\ 0\ 1][0.8\ 0.6\ -1.2\ 0.6\ -0.6\ 0.8\ -1.6\ 0\ 0\ 1] \\
&= \\
&[1\ 0\ 0\ 0\ 1\ 2\ 0\ 0\ 1][0.28\ 0.96\ -1.92\ 0.96\ -0.28\ 0.56\ 0\ 0\ 1] \\
&= [0.28\ 0.96\ -1.92\ 0.96\ -0.28\ 2.56\ 0\ 0\ 1]
\end{aligned}$$

Now, Now, new coordinate of triangle ABC are

$$\begin{aligned}
\begin{bmatrix} x' & y' & 1 \end{bmatrix} &= C.M. \begin{bmatrix} x & y & 1 \end{bmatrix} \\
&= \\
&[0.28\ 0.96\ -1.92\ 0.96\ -0.28\ 2.56\ 0\ 0\ 1] \begin{bmatrix} 4 & 5 & 4 & 1 & 2 & 3 & 1 & 1 & 1 \end{bmatrix} \\
&= [0.16\ 1.4\ 2.08\ 0.12\ 6.8\ 5.56\ 1\ 1\ 1]
\end{aligned}$$

$$A' = (0.16, 6.12)$$

$$B' = (1.4, 6.8)$$

$$C' = (2.08, 5.56)$$

***14. Use Cohen Sutherland clipping method to clip a line starting from A(-1, 5) and ending at B(3, 8) against the window having its lower corner at (-3, 1) and upper right corner at (2, 6).***

[2078 Chaitra]

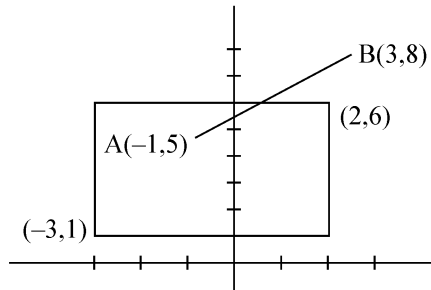
**Solution:**

Here,

The end points of the line are A(-1, 5) and B(3, 8)

The lower corner of the clip window is (-3, 1)

The upper right corner of the clip window is (2, 6)



Now, assign the region code for the end points,

For vertex A(-1, 5)

$x < x_{\min}$ ,  $-1 < -3$  is false, so bit 1 = 0

$x > x_{\max}$ ,  $-1 > 2$  is false, so bit 2 = 0

$y > y_{\min}$ ,  $5 < 1$  is false, so bit 3 = 0

$y > y_{\max}$ ,  $5 < 6$  is false, so bit 4 = 0

So region code for vertex A is, 0000

For vertex B(3, 8),

$x < x_{\min}$ ,  $3 < -3$  is false, so bit 1 = 0

$x > x_{\max}$ ,  $3 > 2$  is true, so bit 2 = 1

$y < y_{\min}$ ,  $8 < 1$  is false, so bit 3 = 0

$y > y_{\max}$ ,  $8 > 6$  is true, so bit 4 = 1

So, region code for vertex B is 1010

Both end point have not region code 0000. So line is not completely inside.

The logical AND of end points of the line AB is 0000. So line is not completely outside.

So, now,

$m = =$

For vertex B, bit 2 is 1. So line intersect with right boundary,

So,

$$y_i = y_1 + m(x_{\max} - x_1)$$

$$= 5 + (2 - (-1))$$

$$= 7.25$$

The x component of the intersection point is x component of right boundary, i.e.  $x_{\max} = 2$

So, the co-ordinate of the intersect point is (2, 7.25)

Now, the region code for new point B'(2, 7.25) is

$2 < -3$  is false so bit 1 = 0

$2 > 2$  is false so bit 2 = 0

$7.25 < 1$  is false so bit 3 = 0

$7.25 > 6$  is true so bit 4 = 1

The region code for B' is 1000

Bit 4 is 1, so, the line intersects with upper boundary.

$$\text{So, } x_i = x_1 + (y_{\max} - y_1)$$

$$= -1 + (6 - 5)$$

$$= -1 +$$

$$= 0.33$$

So, the co-ordinate of new intersection point, i.e. intersection of the line and the upper boundary is B'(0.33, 6)

The region codes for B''(0.33, 6) are,

$0.33 < -3$  is false, so bit 1 = 0

$0.33 > 2$  is false, so bit 2 = 0

$6 < 1$  is false, so bit 3 = 0

$6 > 6$  is false so bit 4 = 0

The required points are A(-1, 5) and B''(0.33, 6)

- 15. Scale the triangle with vertices A(1,1), B(4,4) and C(2,3) to double along horizontal direction and triple of vertical direction about point (2,3) [2078 Chaitra]**

**Solution:**

Here, the vertices of the triangle and A(1,1), B(4,4) and C(2,3).

Scaling factor along horizontal direction ( $S_x$ ) = 2

Scaling factor along vertical direction ( $S_y$ ) = 3

Fixed point about while scaling is done is (2,3)

Now,

$$P' = C.M \times P$$

$$\text{Composite matrix (c.m.)} = T_{(2,3)} S_{(2,3)} T_{(-2,-3)}$$

=

=

=

$$P' =$$

So, the new scaled vertices are  $A'(0,-3)$ ,  $B'(6,6)$  and  $C'(4,3)$ .

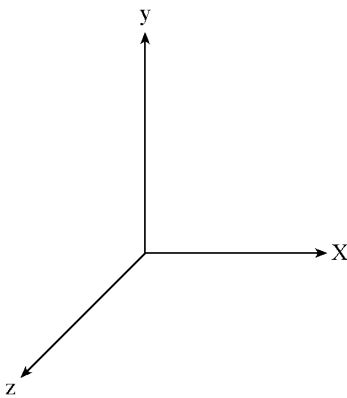
---

## Three-Dimensional Transformations

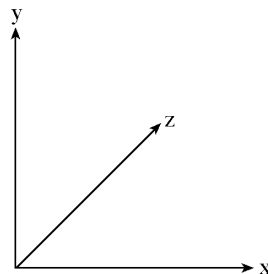
---

## 4.1 Three Dimensional Graphics

Three dimensional graphics uses three dimensional representations of geometric data. Three dimensions add the depth (z) dimension in length (x) and breadth (y) dimension in two dimensions. Three dimension is more complex than two dimension because in three dimension relatively more co-ordinate points are needed, object boundaries can be constructed with various combination of plane and curved surfaces, viewing direction, position in space, orientation, projection consideration, visible surface detections etc. matters in displaying the graphics.



Right hand system



Left hand system

## 4.2 Three-Dimensional Transformations

Three dimensional transformations is the process of manipulating the position of a three- dimensional object with respect to its original position by modifying its physical attributes through various methods of transformation like Translation, Scaling, Rotation, Shear, etc.

- Translation
- Rotation
- Scaling
- Reflection



- Shear

Matrix used in 3D transformation is of order  $4 \times 4$  homogeneous co-ordinate.

#### 4.2.1 Translation/Shifting

In a 3D homogeneous co-ordinate representation, a point is translated from position  $P(x, y, z)$  to position  $P'(x', y', z')$  with the matrix operation

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

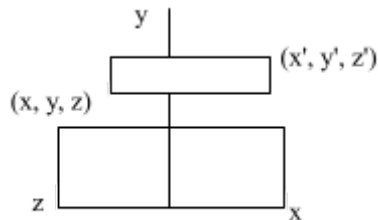
OR

$$P' = T.P$$

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$



**Fig 4.1:** Translation of a point

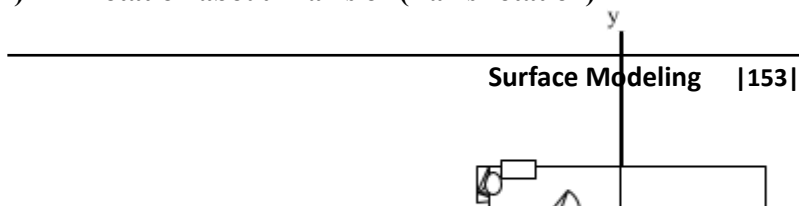
- An object is translated in 3D by transforming each of the defining points of the object.
- For an object represented as set of polygon surfaces, we translate each vertex of each surface and redraw the polygon facts in the new position
- A translation in the opposite direction is obtained by negative the translation distance  $t_x$ ,  $t_y$ , and  $t_z$

#### 4.2.2 Rotation

- We must define an axis of rotation and amount of angular rotation to generate rotation transformation
- Unlike 2D application, where all transformations are carried out in the  $xy$ -plane, a 3D rotation can be specified around any line in space.

#### Co-ordinate Axes Rotations

##### i) Rotation about z- axis or (z axis rotation)



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

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

i.e.  $P' = R_z(\theta).P$

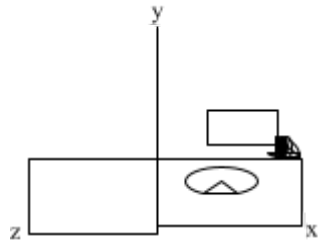
**ii) Rotation about x-axis**

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

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$P' = R_x(\theta).P$



**Fig 4.3 : Rotation about x-axis**

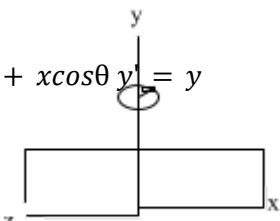
iii) **Rotation about y-axis**

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

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$P' = R_y(\theta).P$$

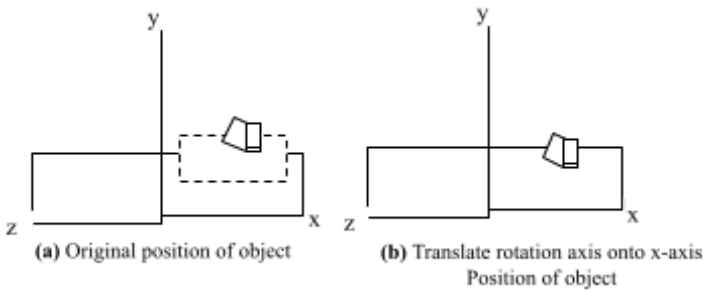


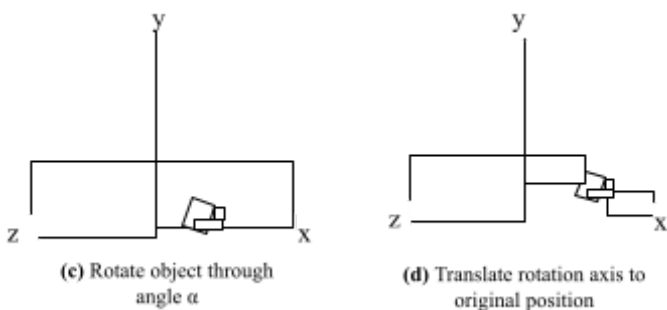
**Fig 4.4 : Rotation about y-axis**

An inverse rotation matrix is formed by replacing the rotation angle  $\theta$  by  $-\theta$ .

**General three dimensional rotations**

- i. **About an axis that is parallel to one of the co-ordinate axes.**
  - a. Translate the object so that the rotation axis coincides with parallel co-ordinate axes.
  - b. Perform the specified rotation about that axis.
  - c. Translate the object so that the window axis is moved back to its original position.





**Figure 4.5:** Rotation about an axis that is parallel to one of the co-ordinate axes.

## ii) Rotation about an arbitrary axis

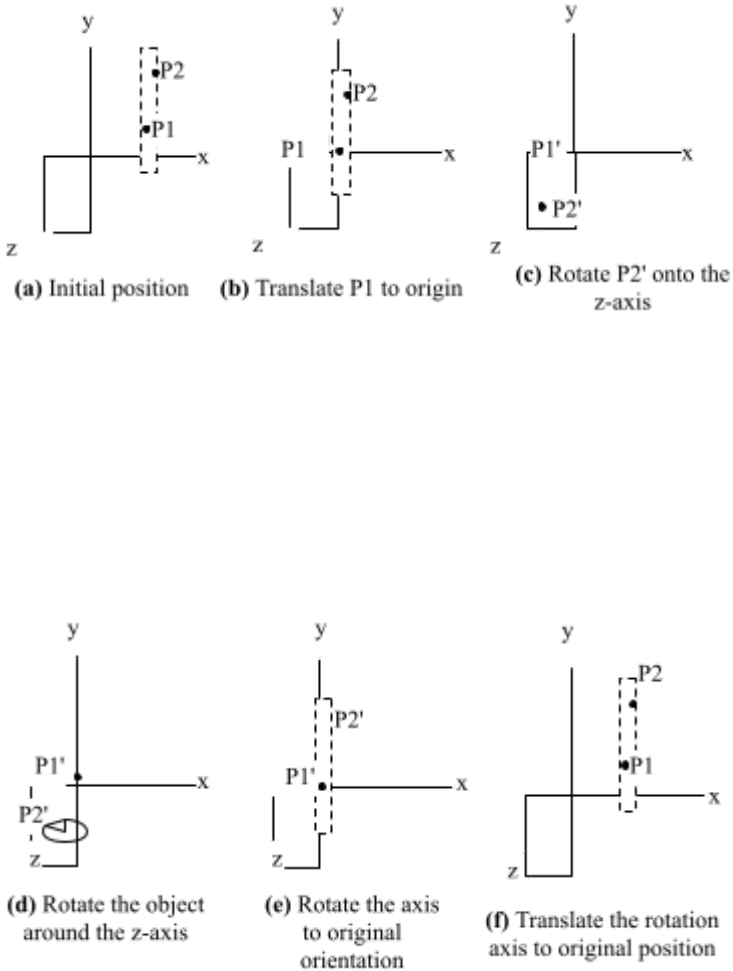
- Translate the object so that the rotation axis passes through the origin
- Rotate the object so that the axis of rotation coincides with one of the co-ordinate axis.
- Perform the specified rotation about that co-ordinate axis.
- Apply inverse rotations to bring the rotation axis back to its original orientation.
- Apply the inverse translation to bring the rotation axis back to its original position.

Rotation of a point in 3 dimensional space by  $\theta$  about an arbitrary axes defined by a line between two points  $P_1 = (x_1, y_1, z_1)$  and  $P_2 = (x_2, y_2, z_2)$  can be achieved by the following steps

- (1) translate space so that the rotation axis passes through the origin
- (2) rotate space about the x axis so that the rotation axis lies in the xz plane
- (3) rotate space about the y axis so that the rotation axis lies along the z axis
- (4) perform the desired rotation by  $\theta$  about the z axis
- (5) apply the inverse of step (3)

(6) apply the inverse of step (2)

(7) apply the inverse of step (1)



**Figure 4.6:** Rotation about an arbitrary axis

A rotation axis can be defined with two co-ordinate positions or with one coordinate point and direction angles (or

direction cosines) between the rotation axis and two of the co-ordinate axes.

- Assume that the rotation axis is defined by two points and that the direction of rotation is to be counterclockwise when looking along the axis from  $P_2$  to  $P_1$ .

- An axis vector is then defined by the two points as

$$\mathbf{V} = \mathbf{P}_2 - \mathbf{P}_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

An unit vector  $\mathbf{u}$  is then defined along the rotation axis as

$$\frac{\mathbf{u}}{|\mathbf{v}|} = (a, b, c)$$

where the components  $a$ ,  $b$  and  $c$  of unit vector  $\mathbf{u}$  are the direction cosines for the rotation axis  $a = \frac{x_2 - x_1}{|\mathbf{V}|}$

$$b = \frac{y_2 - y_1}{|\mathbf{V}|}, \quad c = \frac{z_2 - z_1}{|\mathbf{V}|}$$

### Step 1

Translate space so that the rotation axis passes through the origin.

This is accomplished by translating space by  $-\mathbf{P}_1$  ( $-x_1, -y_1, -z_1$ ). The translation matrix  $\mathbf{T}$  and the inverse  $\mathbf{T}^{-1}$  (required for step 7) are given below

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & -x_1 & 0 & 1 & 0 & -y_1 & 0 & 0 & 1 & -z_1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & x_1 & 0 & 1 & 0 & y_1 & 0 & 0 & 1 & z_1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

### Step 2

Rotate space about the  $x$  axis so that the rotation axis lies in the  $xz$  plane. Let  $\mathbf{U} = (a, b, c)$  be the unit vector along the rotation axis. and define  $d = \sqrt{b^2 + c^2}$  as the length of the projection onto the  $yz$

plane. If  $d = 0$  then the rotation axis is along the  $x$  axis and no additional rotation is necessary. Otherwise rotate the rotation axis so that it lies in the  $xz$  plane. The rotation angle to achieve this is the angle between the projection of rotation axis in the  $yz$  plane and the  $z$  axis. This can be calculated from the dot product of the  $z$  component of the unit vector  $U$  and its  $yz$  projection. The sine of the angle is determined by considering the cross product.

$$\cos(t) = \frac{(0,0,c) \cdot (0,b,c)}{d} = \frac{c}{d} \qquad \sin(t) = \frac{\| (0,0,c) \times (0,b,c) \|}{d} = \frac{b}{d}$$

The rotation matrix  $R_x$  and the inverse  $R_x^{-1}$  (required for step 6) are given below

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & -b/d & 0 \\ 0 & b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_x^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c/d & b/d & 0 \\ 0 & -b/d & c/d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

### Step 3

Rotate space about the  $y$  axis so that the rotation axis lies along the positive  $z$  axis. Using the appropriate dot and cross product relationships as before the cosine of the angle is  $d$ , the sine of the angle is  $a$ . The rotation matrix about the  $y$  axis  $R_y$  and the inverse  $R_y^{-1}$  (required for step 5) are given below.

$$\mathbf{R}_y = \begin{pmatrix} d & 0 & -a & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \mathbf{R}_y^{-1} = \begin{pmatrix} d & 0 & a & 0 \\ 0 & 1 & 0 & 0 \\ -a & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

#### Step 4

Rotation about the z axis by t (theta) is  $\mathbf{R}_z$  and is simply

$$\mathbf{R}_z = \begin{pmatrix} \cos(t) & \sin(t) & 0 & 0 \\ -\sin(t) & \cos(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The complete transformation to rotate a point (x,y,z) about the rotation axis to a new point (x',y',z') is as follows, the forward transforms followed by the reverse transforms.

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \mathbf{T}^{-1} \mathbf{R}_x^{-1} \mathbf{R}_y^{-1} \mathbf{R}_z \mathbf{R}_y \mathbf{R}_x \mathbf{T} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$



If the rotation is to be in the opposite direction (clockwise when viewing from  $P_2$  to  $P_1$ ), then we would reverse axis vector  $v$  and unit vector  $u$  so that they point from  $P_2$  to  $P_1$ .

By moving point  $P_1$  to the origin.

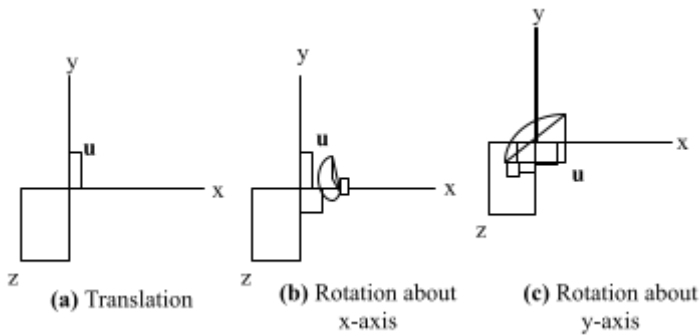
$$T = \begin{bmatrix} 1 & 0 & 0 & -x_1 & 0 & 1 & 0 & -y_1 & 0 & 0 & 1 & -z_1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, we need the transformation that will put the rotation axis on the  $z$  axis.

we can do this in two steps.

First rotate about the  $x$ -axis to transform vector  $u$  into the  $xz$  plane.

Then swing  $u$  around to the  $z$  axis using  $y$ -axis rotation.



**Figure 4.7** Unit vector  $u$  is translation and rotation

Since rotation calculations involve sine and cosine functions, we can use standard vector operations to obtain elements of the two rotation matrices.

$$V_1 \cdot V_2 = |V_1| |V_2| \cos \theta \quad 0 \leq \theta \leq \pi$$

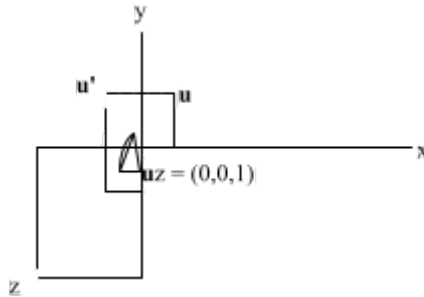
$$V_1 \times V_2 = u |V_1| |V_2| \sin \theta \quad 0 \leq \theta \leq \pi$$

$$V_1 \cdot V_2 = V_{1x} V_{2x} + V_{1y} V_{2y} + V_{1z} V_{2z}$$

$$V_1 \times V_2 = \begin{vmatrix} u_x & u_y & u_z & V_{1x} & V_{1y} & V_{1z} & V_{2x} & V_{2y} & V_{2z} \end{vmatrix}$$

We establish the transformation matrix for rotation around the x axis by determining the values for the sine and cosine of the rotation angle necessary to get  $u$  into the xz plane.

This rotation angle is the angle between the projection by  $u$  in the yz plane and the positive z axis.



**Figure 4.8** Rotation of  $u$  around the x axis into the xz plane is accomplished by rotating  $u'$

If we designate the projection of  $u$  in yz plane as vector  $u' = (0, b, c)$  then cosine of the rotation angle  $\alpha$  can be determined from the dot product of  $u'$  and the unit vector  $u_z$  along z axis

$$\cos \alpha = \frac{u' \cdot u_z}{|u'| |u_z|} = \frac{c}{d} \text{ where } d = \sqrt{b^2 + c^2}$$

Similarly,  $u' \times u_z = u_x |u'| |u_z| \sin \alpha$  and the Cartesian form for the cross product gives us  $u' \times u_z = u_x \cdot b$

$$|u_z| = 1$$

$$|u'| = d$$

$$d \sin \alpha = b$$

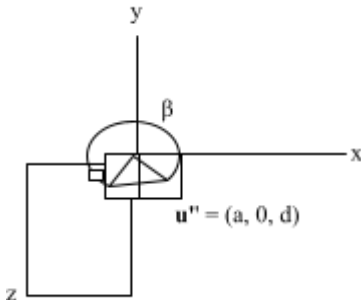
$$\sin \alpha = \frac{b}{d}$$

Now that we have determined the values of  $\sin \alpha$  and  $\cos \alpha$

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & c/d & -b/d & 0 & 0 & b/d & c/d & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix rotates unit vector  $u$  about the  $x$ -axis into  $xz$  plane.

Now, swing the unit vector in  $xz$  plane, counterclockwise around the  $y$  axis onto the positive  $z$  axis



**Figure 4.9** Rotation of unit vector  $u''$  about  $y$  axis

The vector labeled  $u''$  has the value  $a$  for its  $x$  component since rotation about the  $x$ -axis leaves the  $x$ -component unchanged. Its  $z$  component is  $d$  (the magnitude of  $u'$ ). Because vector  $u'$  has been rotated onto the  $z$  axis. And the  $y$  component of  $u''$  is 0 because now it lies in  $xz$  plane.

$$(u')^2 = (\sqrt{a^2 + b^2 + c^2})^2 - (\sqrt{a^2})^2$$

$$(u')^2 = b^2 + c^2$$

$$u' = \sqrt{b^2 + c^2}$$

$$\sqrt{a^2 + b^2 + c^2}$$
$$\sqrt{b^2 + c^2}$$

$\sqrt{a^2}$

$u_z$

$u''$

$$\cos\beta = \frac{u'' \cdot u_z}{|u''||u_z|} = d$$

Since  $|u_z| = |u''| = 1$ , comparing the co-ordinate independent form of the cross product.

$$u'' \times u_z = u_y |u''| |u_z| \sin \beta$$

with the Cartesian form

$$u'' \times u_z = u_y \cdot (-a)$$

We find that  $\sin \beta = -a$

$$R_y(\beta) = \begin{bmatrix} d & 0 & -a & 0 & 0 & 1 & 0 & 0 & a & 0 & d & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The specified rotation angle  $\theta$  can be applied as a rotation about the z axis.

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 & \sin \theta & \cos \theta & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

To complete the required rotation about the given axis, we need to transform the rotation axis back to its original position.

$$R(\theta) = T^{-1} \cdot R_x^{-1}(\alpha) R_y^{-1}(\beta) R_z(\theta) \cdot R_y(\beta) \cdot R_x(\alpha) \cdot T$$

A somewhat quicker but perhaps less intuitive, method for obtaining the composite rotation matrix  $R_y(\beta) \cdot R_x(\alpha)$  is to take advantage of the form of the composite matrix for any sequence of three dimensional rotations.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 & r_{21} & r_{22} & r_{23} & 0 & r_{31} & r_{32} & r_{33} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The upper left  $3 \times 3$  submatrix of this matrix is orthogonal. This means row (or columns) of this sub-matrix form a set of orthogonal unit vectors that are rotated by matrix R onto the x, y, and z axes respectively.

$$\begin{aligned} R \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & 1 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}, \\ R \cdot \begin{bmatrix} r_{21} & r_{12} & r_{23} & 1 \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix}, \\ R \cdot \begin{bmatrix} r_{31} & r_{32} & r_{33} & 1 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix} \end{aligned}$$

We can consider a local co-ordinate system defined by the rotation axis and simply form a matrix whose columns are the local unit co-ordinate vectors. Assume rotation axis is not parallel to any co-ordinate axis.

$$u'_z = u, u'_y = \frac{u \times u_x}{|u \times u_x|}, u'_x = u'_y \times u'_z$$

And if we express element of local unit vectors for rotation axis as

$$u'_x = (u'_{x1}, u'_{x2}, u'_{x3}), u'_y = (u'_{y1}, u'_{y2}, u'_{y3})$$

$u'_z = (u'_{z1}, u'_{z2}, u'_{z3})$  then the required composite matrix equal to the product  $R_y(\beta) \cdot R_x(\alpha)$  is

$$R = \begin{bmatrix} u'_{x1} & u'_{x2} & u'_{x3} & 0 & u'_{y1} & u'_{y2} & u'_{y3} & 0 & u'_{z1} & u'_{z2} & u'_{z3} & 0 & 0 & 0 & 1 \end{bmatrix}$$

**Q. Perform rotation of a line (10, 10, 10), (20,20,15) about Y-axis in clock wise direction by 90 degree. [2071 Shravan]**

**Solution:**

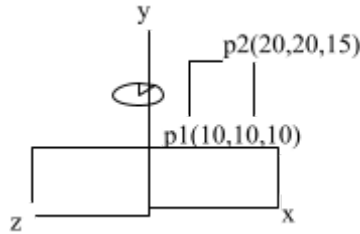
Rotation about y-axis in clock wise direction

$$z' = z \cos\theta + x \sin\theta$$

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

$$y' = y$$

$$\theta = 90^\circ$$



In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos 90^\circ & 0 & \sin 90^\circ \\ 0 & 1 & 0 \\ -\sin 90^\circ & 0 & \cos 90^\circ \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}$$

$$= \begin{bmatrix} 10 \\ 10 \\ -10 \end{bmatrix}$$

$$\text{so, } (x'_1, y'_1, z'_1) = (10, 10, -10)$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 20 & 20 & 15 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -15 & 20 & 20 & 1 \end{bmatrix}$$

$$\text{so, } (x_2', y_2', z_2') = (-15, 20, 20)$$

**Q.** Reflect the object (0, 0, 0), (2, 3, 0) and (5, 0, 4) about the plane  $y = 4$ . [2071 Chaitra]

**Solution:**

**Steps to reflect the object.**

- Translate the plane so that the plane coincides with the  $xz$  plane
- Perform the  $xz$  reflection
- Translate the object so that reflection plane is moved back to its original position.

$$\text{C.M.} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = \text{C.M.} \times p$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

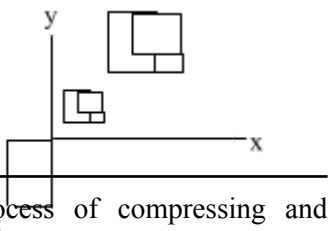
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & -4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

#### 4.2.3 Scaling

Scaling is defined as the process of compressing and expanding of object. It changes the size of an object. If the scaling parameters are not same, then it distorts the shape of the object. The equation for scaling about the origin is,

$$x' = x \times s_x$$

$$y' = y \times s_y$$



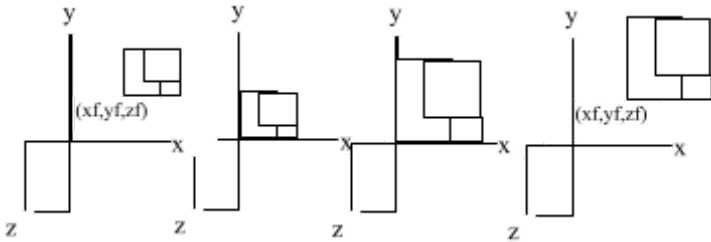
$$Z' = Z \times SZ$$

In matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

### Scaling about fixed point

- Translate the fixed point to the origin.
- Scale the object about the co-ordinate origin.
- Translate the fixed point back to its original position.



$$C.M. = T^{-1}.S.T.$$

$$= T(x_f, y_f, z_f). S(s_x, s_y, s_z). T(-x_f, -y_f, -z_f)$$

$$= \begin{bmatrix} s_x & 0 & 0 & (1 - s_x)x_f & 0 & s_y & 0 & (1 - s_y)y_f & 0 & 0 & s_z & (1 - s_z)z_f & 0 & 0 & 0 & 1 \end{bmatrix}$$

### 4.2.4 Reflection

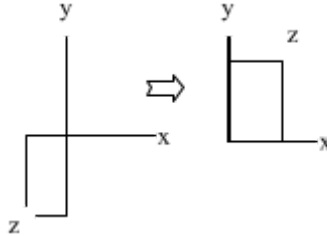
- A 3D reflection can be performed relative to a selected 'reflection axis' or with respect to a selected 'reflected plane'.
- Reflection relative to a given axis is equivalent to 180° rotations about that axis.
- Reflection with respect to a plane is equivalent to 180° rotations in three dimensional space.
- When the reflection plane is a co-ordinate plane (either xy, xz or yz) we can think of the transformation as a conversion between left handed and right handed systems.

**For example,**

If an object is reflected about xy plane, it simply changes the sign of z values keeping the sign of x and y values same. So

### Reflection about z-axis or xy plane

$$P' = R_{f(z)} \cdot P$$



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

### Reflection about x-axis or yz plane

$$P' = R_{f(x)} \cdot P$$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

### Reflection about y-axis or xz plane

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## 4.2.5 Shearing

- Shearing transformation are used to modify shape of the object.
- In space, one can push in z co-ordinate axis direction, keeping the third axis fixed.

**Shearing in x and y direction keeping z co-ordinate same (along z-axis)**

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x + sh_x \cdot z$$

$$y' = y + sh_y \cdot z$$



$$z' = z$$

It alters the x and y co-ordinates values by an amount that is proportional to the z axis while leaving z co-ordinate. same as  $p' = sh_{xy}.p$

**Shearing in yz direction keeping x co-ordinate same (along x-axis)**

$$\begin{bmatrix} 1 & 0 & 0 & sh_y & 1 & 0 & 0 & sh_z & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

$$P' = sh_{yz} . P$$

$$x' = x$$

$$y' = y + sh_y . x$$

$$z' = z + sh_z . x$$

**Shearing in xz direction keeping y co-ordinate same (along y-axis)**

$$\begin{bmatrix} 1 & sh_x & 0 & 0 & 0 & 1 & 0 & 0 & 0 & sh_z & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

$$p' = sh_{xz} . p$$

$$y' = y$$

$$x' = x + sh_x . y$$

$$z' = z + sh_z . y$$

**Q. Find the new coordinates of a unit cube  $90^\circ$  rotated about an axis defined by its end points  $P1(2, 1, 0)$  and  $P2(3, 3, 1)$ .**

**Solution:**

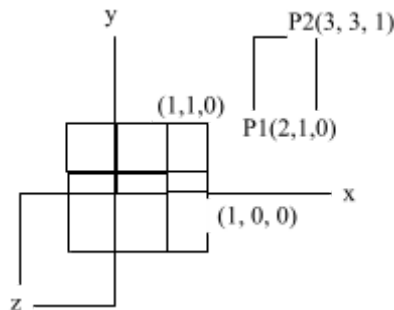
$$v = P_2 - P_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

$$a = \frac{x_2 - x_1}{vV}$$

$$b = \frac{y_2 - y_1}{vV}$$

$$c = \frac{z_2 - z_1}{vV}$$

$$d =$$



$$R(\theta) = T^{-1} R_x^{-1}(\alpha) R_y^{-1}(\beta) . R_z(\theta) . R_y(\beta) . R_x(\alpha) . T$$

$$\begin{aligned}
& \dots \left[ 1 \ 0 \ 0 \ 0 \ 0 \ \frac{c}{d} \ \frac{b}{d} \ 0 \ 0 \ \frac{-b}{d} \ \frac{c}{d} \ 0 \ 0 \ 0 \ 0 \ 1 \right] \\
& [d \ 0 \ a \ 0 \ 0 \ 1 \ 0 \ 1 \ - \ a \ 0 \ d \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [\cos 90 \ - \ \sin 90 \ 0 \ 0 \ \sin 90 \ \cos 90 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [d \ 0 \ - \ a \ 0 \ 0 \ 1 \ 0 \ 1 \ a \ 0 \ d \ 0 \ 0 \ 0 \ 0 \ 1] \\
& \left[ 1 \ 0 \ 0 \ 0 \ 0 \ \frac{c}{d} \ \frac{-b}{d} \ 0 \ 0 \ \frac{b}{d} \ \frac{c}{d} \ 0 \ 0 \ 0 \ 0 \ 1 \right] \\
& [1 \ 0 \ 0 \ - \ 2 \ 0 \ 1 \ 0 \ - \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
= & [1 \ 0 \ 0 \ 2 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ 0.894 \ 0 \ 0 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ - \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0 \ - \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ - \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ - \ 0.894 \ 0 \ 0 \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ - \ 2 \ 0 \ 1 \ 0 \ - \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
- & [1002010100100001] [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ 0.894 \ 0 \ 0 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ - \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0 \ - \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ - \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ - \ 2 \ 0 \ 0.4472 \ - \ 0.894 \ - \ 0.4472 \ 0 \ 0.894 \ 0.4472 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
= & [1 \ 0 \ 0 \ 2 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ 0.894 \ 0 \ 0 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ - \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0 \ - \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ - \ 0.3647 \ - \ 0.1824 \ - \ 1.46 \ 0 \ 0.4472 \ - \ 0.894 \ - \ 0.4472 \ 0 \ 0.894 \ 0.4472 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
= & [1 \ 0 \ 0 \ 2 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ 0.894 \ 0 \ 0 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [0.9128 \ 0 \ 0.408 \ 0 \ 0 \ 1 \ 0 \ 1 \ - \ 0.408 \ 0 \ 0.9128 \ 0 \ 0 \ 0 \ 0 \ 1] \\
[0 \ - \ 0.4472 \ 0.894 \ - \ 0.447 \ 0.9128 \ - \ 0.364 \ - \ 0.1824 \ - \ 0.146 \ 0.4472 \ 0 \ 0.894 \ 0.4472 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
= & [1 \ 0 \ 0 \ 2 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
& [1 \ 0 \ 0 \ 0 \ 0 \ 0.4472 \ 0.894 \ 0 \ 0 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1] \\
[0.1664 \ - \ 0.074 \ 0.983 \ - \ 0.258 \ 0.9128 \ - \ 0.364 \ - \ 0.1824 \ - \ 1.46 \ 0.4472 \ 0 \ 0.894 \ 0.4472 \ - \ 0.894 \ 0.4472 \ 0 \ 0 \ 0 \ 0 \ 1]
\end{aligned}$$

$$\begin{aligned}
&= [1 \ 0 \ 0 \ 2 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1] \\
&\quad [0.1664 \ -0.074 \ 0.983 \ -0.258 \ 1.278 \ 0.666 \ -0.075 \ -2.14] \\
&= \\
&\quad [0.1664 \ -0.074 \ 0.983 \ -0.258 \ 1.278 \ 0.666 \ -0.075 \ -2.14] \\
&p' = c.m. \times p \\
&= \\
&\quad [0.1664 \ -0.074 \ 0.983 \ -0.258 \ 1.278 \ 0.666 \ -0.075 \ -2.14] \\
&\quad 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1
\end{aligned}$$

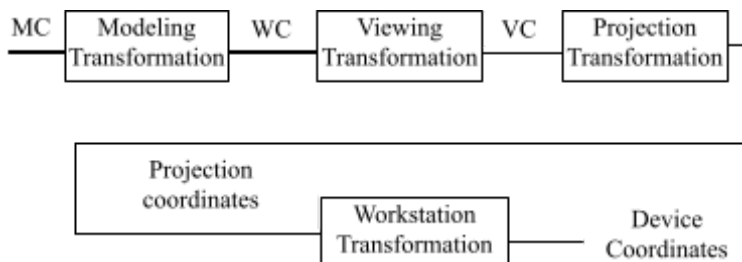
Let the original coordinates are A(0, 1, 1), B(1, 1, 1), C(1, 0, 1), D(0, 0, 1), E(1, 1, 0), F(1, 0, 0), G(0, 1, 0), H(0, 0, 0) then the corresponding new coordinates are A'(1.3718, 0.927, 0.37), B'(1.537, 1.6681, -0.279), C'(1.4308, 0.6381, -1.202), D'(1.2648, -0.103, -0.553), E'(0.92, 2.4721, -0.081), F'(0.813, 1.4421, -1.004), G'(0.647, 0.701, -0.355), H'(0.754, 1.731, 0.568)

### 4.3 3D viewing

In 2D graphics application, viewing operations transfer position from the world coordinate plane to pixel position in the plane of the output device.

- But in 3D graphics application, we have to consider spatial position (i.e., an object can be viewed from the front, from above or from the back) or we could generate a view of what we would see if we were standing in the middle of a group of objects or inside a single object, such as buildings.
- Additionally, 3D descriptions of object must be projected onto the flat viewing surface of the output device.
- And the clipping boundaries now enclose a volume of space, whose shape depends on the type of projection we select.

## Viewing pipeline



**Figure 4.10** General three-dimensional transformation pipeline, from modeling co-ordinate to find device co-ordinates

3D viewing pipeline describes the conversion of 3D object into 2D projection or mapping by using some processes.

- The steps for computer generation of a view of a three dimensional scene are analogous to the process involved in taking a photograph.
- To take a snapshot, we first need to position the camera at a particular point in space, then need to decide on the camera orientation. (i.e., which way do we point the camera and how should we rotate it around the line of sight to set the updirection for the picture). Finally, when we snap the shutter, the scene is cropped to the size of the shutter, the scene is cropped to the size of the 'window' (aperture) of the camera and light from the visible surface is projected.

Each model or object has its own dimension and coordinate system. It is called modeling coordinate. Modeling transformation is to take all the objects in a single scene by using transformation such translation, rotation etc. To set object is called modeling translation.

After modeling translation, the objects come to a scene or a coordinate system is called world coordinate.

To set the camera on some position, angle or orientation is called viewing transformation. From viewing transformation we get viewing coordinate.

Projection transformation is to adjust focus, zoom in, zoom out etc. Projection transformation creates projection coordinates.

Workstation or viewport translation is just like to click the button to save the image in the device.

Once the scene has been modeled, world coordinate positions are converted to viewing co-ordinate.

- The VC system is used in graphics system as a reference for specifying the observer viewing position and the position of the projection plane analogous to camera film plane.
- Projection operations are performed to convert VC description of a scene to co-ordinate positions on the projection plane.
- The projection plane is then mapped to output device.

### **Viewing co-ordinates:**

Views of a scene can be generated by given spatial position (i.e., various distances), angle (i.e., angle with  $z_v$  axis), orientation, aperture (i.e., 'window') size of the camera.

Generating a view of an object in three dimensions is similar to photographing the object

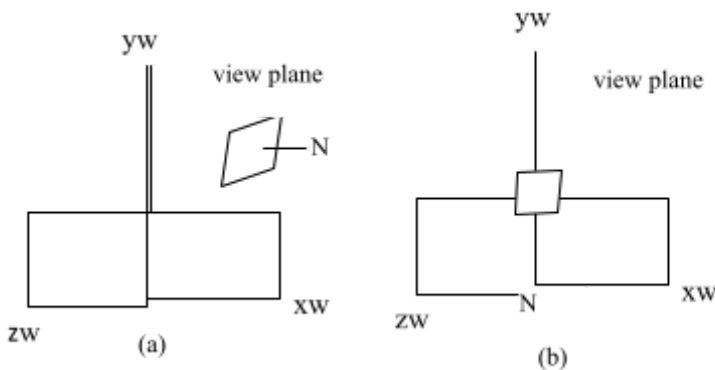
### **Specifying the view plane**

- We choose a particular view for a scene by first establishing the viewing co-ordinate system also called view reference co-ordinate system
- A viewplane or projection plane is then set up perpendicular to the viewing  $z_v$  axis.
- WC positions in the scene are transformed to VC then VC are projected onto the view plane.

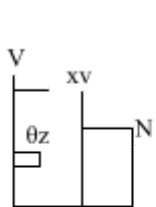
### **Establishing the VC reference frame**

- First, pick a WC position called view reference point, the origin of our viewing co-ordinate system.

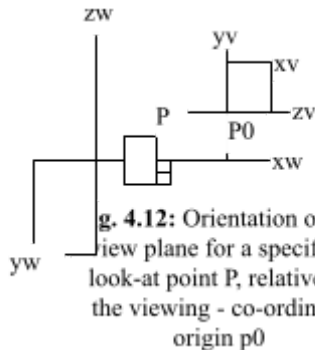
- The view reference point is often chosen to be close to or on the surface of some object in a scene. But it may be center of object or group object.
- Next, the positive direction for the viewing  $z_v$  axis and the orientation of the view plane is selected by specifying the view plane normal vector  $\mathbf{N}$
- We choose a WC position and this point establishes the direction for  $\mathbf{N}$  relative either to the world origin or to the viewing co-ordinate origin. Establish direction of  $\mathbf{N}$  using the selected co-ordinate position as a look at point relative to view reference point (view coordinate origin).
- Finally, we choose the up direction for the view by specifying a vector  $\mathbf{V}$ , called view-up vector.
- This vector is used to establish the true direction for the  $y_v$  axis.



**Figure 4.10:** Orientation of the view plane for specified normal vector co-ordinates relative to the world origin position  $(1,0,0)$  orients the view plane as in (a) and  $(1,0,1)$  gives the orientation in (b)

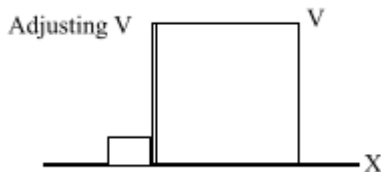


**Fig. 4.11:** specifying the view-up vector with a twist angle  $\theta_t$



**g. 4.12:** Orientation of the view plane for a specified look-at point  $P$ , relative to the viewing - co-ordinate origin  $p_0$

- Vector  $V$  can be defined as a world co-ordinates origin  $P_0$ .
- Vector  $V$  can be defined as a world co-ordinate vector or in some packages it is specified with twist angle  $\theta_t$  about the  $z_v$  axis.
- For general orientation of the normal vector, it can be difficult (or time consuming) to determine the direction for  $V$  that is precisely perpendicular to  $N$ .



**Figure 4.13:** Adjusting the input position of the view-up vector  $V$  to a position perpendicular to the normal vector  $N$

- Viewing procedures typically adjust the user-defined orientation of vector  $V$ ,  
So that  $v$  is projected into a plane that is perpendicular to the normal vector.
- We choose the view up vector  $V$  to be in any convenient direction, as long as it is not parallel to  $N$
- Using vector  $N$  and  $V$ , the graphics package can compute a third vector  $U$  perpendicular to both  $N$  and  $V$ . to define the direction for the  $X_v$  axis.

- Then the direction of  $\mathbf{V}$  can be adjusted so that it is perpendicular to both  $\mathbf{N}$  and  $\mathbf{U}$  to establish the viewing direction
- In transformation from world to viewing co-ordinates, these computations are conveniently carried out with unit axis vector.

## View plane

- The window is defined in this plane.
- The origin of this plane which defines the position of the eye or camera is called the view reference point  

$$\mathbf{e} = (e_x, e_y, e_z)$$
- A unit vector to this plane is the view plane normal  $\mathbf{N}$ .
- Another vector called the view up vector.  $\mathbf{V}_{up}$  is a unit vector perpendicular to  $\mathbf{N}$ .

## View coordinate system

- Usually left-handed system called the uvn system
- $\mathbf{V}$ , the y-axis of the view co-ordinates system is perpendicular projection  $\mathbf{V}_{up}$  of on the view plane.
- $\mathbf{u}$ , the x axis of the view co-ordinate is orthogonal to  $\mathbf{V}$  and  $\mathbf{N}$  i.e.,  $\mathbf{U} = \mathbf{V} \times \mathbf{N}$
- Positive  $\mathbf{u}$  and  $\mathbf{v}$  are to the right and up-direction from eye's point of view.
- $\mathbf{N}$  is the z-axis of the view co-ordinate. It increases in positive direction with depth of a point from the eye.

## Viewing co-ordinate Parameters

We call the viewing co-ordinate frame  $\mathbf{u}\mathbf{v}\mathbf{n}$ , where  $\mathbf{u}$ ,  $\mathbf{v}$  and  $\mathbf{n}$  are three orthogonal vectors.

Let,  $P_0$  be the view co-ordinate origin.

- $P_{ref}$  be the look at point in the scene.
- $\mathbf{N}$  be the vector from  $p_{ref}$  to  $P_0$
- Then  $\mathbf{n}$  is the unit vector in the direction of  $\mathbf{N}$ .



- Let  $\mathbf{v}$  be the unit vector in the view up direction  $\mathbf{N}$ .
- Vector  $\mathbf{u}$  is perpendicular to  $\mathbf{u}$  and  $\mathbf{n}$ , where  $\mathbf{U} = \mathbf{V} \times \mathbf{N}$   
Usually, the user specifies  $P_0$  and  $p_{ref}$  and the view up vector  $\mathbf{v}$
- An eye defined within this system.

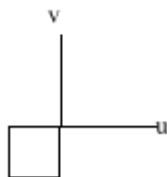
Usually, user doesn't give precise  $\mathbf{V}$  exactly perpendicular to  $\mathbf{N}$ .

Therefore, we use the following method to find  $\mathbf{u}$ ,  $\mathbf{v}$  and  $\mathbf{n}$

$$\mathbf{n} = \frac{\mathbf{N}}{|\mathbf{N}|} \quad \mathbf{u} = \frac{\mathbf{V} \times \mathbf{n}}{|\mathbf{V}|}$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u}$$

Let  $\mathbf{a} = (a_x, a_y, a_z)$  be look-at-point



For perspective views, the view plane normal as a unit vector from eye to a look-at point is given by

$$\mathbf{N} = \frac{1}{|a-e|} (a_x - e_x, a_y - e_y, a_z - e_z) \quad |a - e| = \sqrt{(a_x - e_x)^2 + (a_y - e_y)^2 + (a_z - e_z)^2}$$

The view up vector is the tilt(rotation) of the head or camera.

For parallel views it is convenient to think of the view plane normal as determining the direction of projection.

## 4.4 Transformation from world to viewing coordinates

The transformation from world coordinates to viewing coordinates is a crucial step in computer graphics and involves several stages. This process typically includes translating, rotating, and scaling the world coordinates to align them with the camera's viewpoint. Here's an overview of the steps involved in transforming world coordinates to viewing coordinates:

### 1. Define the Camera (View) Coordinate System

The camera coordinate system (also known as the view coordinate system) is defined relative to the camera's position and orientation in the world. It involves specifying:

- **Eye (Camera) Position E:** The position of the camera in world coordinates.
- **Gaze (Look) Direction G:** The direction in which the camera is looking.
- **Up Vector U:** The upward direction relative to the camera.

## 2. Construct the Camera (View) Coordinate Axes

From the camera parameters, construct the right-handed coordinate system for the camera:

- **Forward (Viewing) Vector N:** This is typically the normalized inverse of the gaze direction.  $N = -G/||G||$
- **Right Vector U:** This is obtained by taking the cross product of the up vector and the viewing vector.  $U = U \times N / ||U \times N||$
- **True Up Vector V:** This is recalculated to ensure orthogonality.  $V = N \times U$

## 3. Translation Matrix

To translate the world coordinates to the camera's position, we use a translation matrix that shifts the world origin to the camera's position:

$$\begin{bmatrix} 1 & 0 & 0 & -E_x & 0 & 1 & 0 & -E_y & 0 & 0 & 1 & -E_z & 0 & 0 & 0 & 1 \end{bmatrix}$$

where  $E=(E_x,E_y,E_z)$  is the eye position.

## 4. Rotation Matrix

To align the world coordinates with the camera's coordinate system, we use a rotation matrix constructed from the camera's right, up, and forward vectors:

$$R = \begin{bmatrix} U_x & U_y & U_z & 0 & V_x & V_y & V_z & 0 & N_x & N_y & N_z & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where  $U=(U_x,U_y,U_z)$ ,  $V=(V_x,V_y,V_z)$ , and  $N=(N_x,N_y,N_z)$ .

## 5. View Transformation Matrix

Combine the translation and rotation matrices to form the view transformation matrix:

$$V=R \cdot T$$

This matrix transforms points from world coordinates to camera (view) coordinates.

## 6. Transformation of a Point

To transform a point  $P=(x,y,z,1)^T$  from world coordinates to viewing coordinates, you multiply it by the view transformation matrix  $V$ :

$$P'=V \cdot P$$

### Summary

1. Define the camera's position and orientation.
2. Construct the camera coordinate system vectors.
3. Create the translation matrix to move the world coordinates to the camera's position.
4. Create the rotation matrix to align the coordinates with the camera's orientation.
5. Combine the translation and rotation matrices to get the view transformation matrix.
6. Transform points by multiplying them with the view transformation matrix.

This process ensures that objects in the world are transformed correctly to the camera's viewpoint, enabling proper rendering from the camera's perspective.

Suppose that the viewing co-ordinates are specified in world co-ordinates. We need to transform each vertex specified in world co-ordinates to view co-ordinates.

1. Translate viewing co-ordinate origin to world co-ordinates origin.
2. Apply rotation to align u, v and n with the world x, y and z axes.

### Transformation matrices:

#### • Translation

If the view reference point is specified at world position ( $x_0$ ,  $y_0$ ,  $z_0$ ) this point is translated to world origin with the matrix transformation.

$$\begin{bmatrix} 1 & 0 & 0 & -x_0 & 0 & 1 & 0 & -y_0 & 0 & 0 & 1 & -z_0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

#### • Rotation

Apply rotations to align the  $X_v$ ,  $Y_v$ , and  $Z_v$  axes with the corresponding world axes.

- Rotate around the world  $X_w$  axis to bring  $Z_v$  into the  $X_wZ_w$  plane
- Rotate around the world  $Y_w$  axis to align the  $Z_w$  and  $Z_v$  axis
- Final rotation is about the  $Z_w$  axis to align the  $Y_w$  and  $Y_v$  axis

The composite rotation matrix for viewing transformation is then

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 & v_1 & v_2 & v_3 & 0 & n_1 & n_2 & n_3 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

### Another approach,

- An object in world co-ordinate space, whose vertices are ( $x$ ,  $y$ ,  $z$ ) can be expressed in term of view co-ordinates ( $u$ ,  $v$ ,  $n$ )
- Translate the view reference point  $e$  to the origin.
- Rotate about the world co-ordinates of axis to bring the view co-ordinate axis into the  $yz$  plane of world co-ordinate
- Rotate about the world co-ordinates  $z$  axis to align the axis with the  $y$ .
- Reflect relative to  $xy$ -plane, revising sign of each  $z$  co-ordinate to change into a left handed coordinate system

The viewing transformation are  $V = T.R_y. R_x. R_z. R_f$

## 4.5 Projection

Projection can be defined as representing an n-dimensional object into an n-1 dimension. It is process of converting a 3D object into 2D object, we represent a 3D object on a 2D plane  $\{(x,y,z) \rightarrow (x,y)\}$ . It is also defined as mapping or transforming of the object in projection plane or view plane which constitutes the display surface. When geometric objects are formed by the intersection of lines with a plane, the plane is called the projection plane and the lines are called projections.

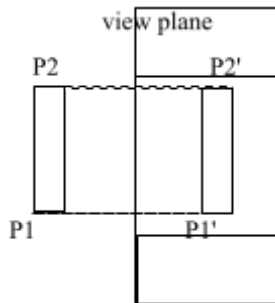
The mapping is determined by a projection line called the projector that passes through p and intersects the view plane. The intersection point is P'.

### Two basic methods

- i. Parallel projection
- ii. Perspective projection

### Parallel projections:

Co-ordinate positions are transformed to the view plane along parallel lines.



**Figure 4.14:** Parallel projection of an object to the view plane

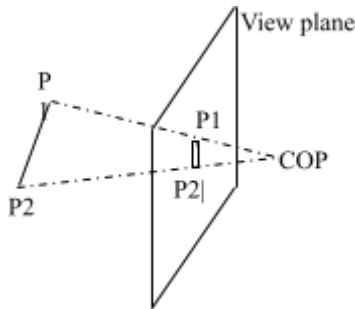
Parallel projection preserves relative proportions of objects, and this method is used in drafting to produce scale drawing of three dimensional objects.

Accurate views of the various sides of an object are obtained with a parallel projection. But this does not give us a realistic representation of the appearance of a three dimensional object.

### **Perspective projection:**

Object positions are transformed to the view plane along lines that converge to a point called projection reference point (or center of projection).

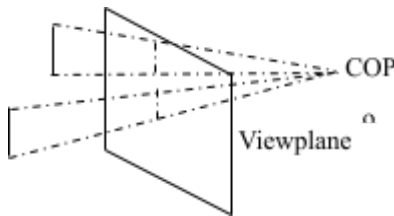
The projected view of an object is determined by calculating the intersection of the projection plane with the view plane.



**Figure 4.15:** *Perspective projection of an object to view plane*

Perspective projection produces realistic views but does not preserve relative proportions.

Projection of distant objects are smaller than the projection of objects of the same size that are closer to the view plane.



**Figure 4.16:** *Perspective projection of equa- sized objects at different distances from the view plane*

## Parallel projections:

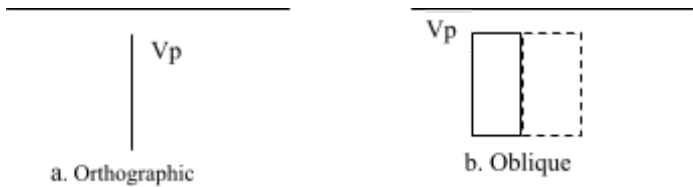
We can specify a parallel projection with a projection vector that defines the direction for the projection line.

### Orthographic parallel projection:

The projection is perpendicular to the view plane.

### Oblique parallel projection:

The projection is not perpendicular to the view plane.



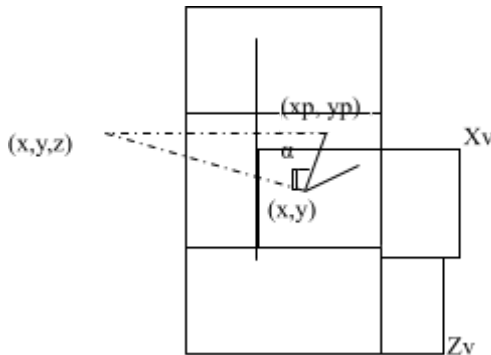
**Figure 4.17:** Orientation of the projection vector  $V_p$  to produce an orthographic projection (a) and an oblique projection (b)

- Orthographic projections are used to produce elevation (front, side, real projections) and plan view (top projection)
- Used in engineering and architectural drawings.
- We can also form orthographic projections that display more than one face of an object such views are called axonometric orthographic projections.
- Most generally used axonometric projection is the isometric projection.
- We generate isometric projection by aligning the projection plane so that it intersects each co-ordinate axis in which the object is defined (principal axes) at the same distance from the origin.
- If view plane is placed at position  $z_{vp}$  along the  $z_v$  axis, then any point  $(x,y,z)$  in viewing co-ordinates is transformed to projection co-ordinate as  $x_p = x, y_p = y$

Where the original z co-ordinate value is preserved for the depth information needed in visible surface detection procedures.

## Oblique projection

- An oblique projection is obtained by projecting point along parallel lines that are not perpendicular to the projection plane.
- Some application packages define an oblique projection vector with two angles  $\alpha$  and  $\phi$ .



**Figure 4.18:** Oblique projection of coordinate position  $(x, y, z)$  to position  $(x_p, y_p)$  on the view plane

- Point  $(x, y, z)$  is projected to position  $(x_p, y_p)$  at the view plane with oblique projection
- $(x, y)$  is the orthographic projection co-ordinates of  $(x, y, z)$
- Oblique projection line from  $(x, y, z)$  to  $(x_p, y_p)$  makes an angle  $\alpha$  with the line on the projection plane that joins  $(x_p, y_p)$  and  $(x, y)$
- This line of length  $L$ , is at an angle  $\phi$  with the horizontal direction in the projection plane

$$\text{then, } x_p = x + L \cos \phi$$

$$y_p = y + L \sin \phi$$

$$\text{But, } \tan \alpha = \frac{z}{L}$$



$$L = \frac{Z}{\tan \alpha} = ZL_1$$

Now,

$$x_p = x + zL_1 \cos \phi$$

$$y_p = y + zL_1 \sin \phi$$

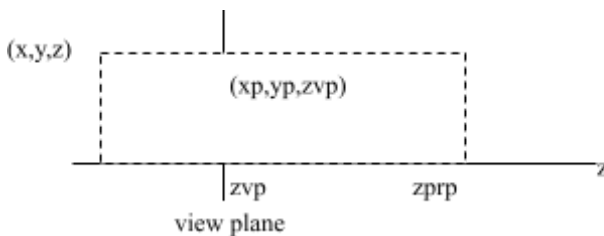
Then, the transformation matrix for producing any parallel projection onto the  $x_v y_v$  Plane becomes

$$M_{\text{parallel}} = \begin{bmatrix} 1 & 0 & L_1 \cos \phi & 0 & 0 & 1 & L_1 \sin \phi & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

- An or the graphic projection is obtained when  $L_1 = 0$  (which occurs at a projection angle  $\alpha = 90$ )
- Oblique projections are generated with non zero values for  $L_1$
- Projection matrix for parallel projection is similar to that of a  $z$  axis shear matrix
- In fact, the effect of this projection matrix is to shear planes of constant  $Z$  and project them onto view plane
- The  $x$  and  $y$  co-ordinate values within each plane of constant  $z$  are shifted by an amount proportional to the  $z$  value of the plane so that angles, distances and parallel lines in the plane are projected accurately.

## Perspective projection

Suppose that cop is at  $z_{\text{prp}}$  along  $z_v$  axis, and view plane is at  $z_{\text{vp}}$

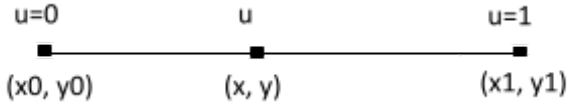


**Figure 4.19:** Perspective projection of a point  $P$  with coordinates  $(x, y, z)$  to position  $(x_p, y_p, z_{vp})$  on the view plane

The parametric equation of line is;

$$x = (1 - u) x_0 + u x_1$$

$$y = (1 - u) y_0 + u y_1$$



We can write equations describing co-ordinate positions along the perspective projection line as

$$x' = x(1 - u) + 0(u)$$

$$y' = y(1 - u) + 0(u)$$

$$z' = z(1 - u) + z_{prp}(u)$$

Now, by simplifying,

$$x' = x - xu \dots\dots\dots (i)$$

$$y' = y - yu \dots\dots\dots(ii)$$

$$z' = z - (z - z_{prp})u \dots\dots\dots(iii)$$

Where  $u$  varies from 0 to 1

When  $u = 0$ ,

$$x' = x$$

$$y' = y$$

$$z' = z, \text{ at original position}$$

When  $u = 1$ ,

$$x' = 0$$

$$y' = 0$$

$$z' = z_{prp} \text{ at the projection reference point}$$

On the view plane  $z' = z_{vp}$ , so equation (iii) becomes  $z_{vp} = z - (z - z_{prp})u$

$$\text{or } u = \frac{z_{vp} - z}{z_{prp} - z} \dots\dots iv$$

also,  $x' = x_p$  and  $y' = y_p$ , so equation(i) and (ii) becomes

$$\begin{aligned} x_p &= x - x \frac{(z_{vp} - z)}{(z_{prp} - z)} \\ &= x \times \frac{(z_{prp} - z - z_{vp} + z)}{(z_{prp} - z)} \\ &= x \times \frac{(z_{prp} - z_{vp})}{(z_{prp} - z)} \\ x_p &= x \times \frac{(d_p)}{(z_{prp} - z)} \dots\dots\dots(v) \end{aligned}$$

Where  $d_p = z_{prp} - z_{vp}$ , is the difference between COP and view plane distances

also,

$$\begin{aligned} y_p &= y - y \frac{(z_{vp} - z)}{(z_{prp} - z)} \\ &= y \times \frac{(z_{prp} - z - z_{vp} + z)}{(z_{prp} - z)} \\ &= y \times \frac{(z_{prp} - z_{vp})}{(z_{prp} - z)} \\ y_p &= y \times \frac{(d_p)}{(z_{prp} - z)} \dots\dots\dots(vi) \end{aligned}$$

And  $z_p = z_{vp} \dots\dots\dots(vii)$

using equation (v), (vi) and (vii), we form a matrix

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\text{where } h = \frac{z_{prp} - z}{d_p}$$

$$x_p = , y_p =$$

and original z co-ordinate value would be retained for visible surface and other depth processing

### Special cases

- i. When  $z_{vp} = 0$ , view plane passes through origin

$$x_p = x \times \frac{\left(\frac{d_p}{z_{prp}}\right)}{\left(\frac{z_{prp}}{z_{prp}} - z\right)} = x \times \frac{1}{\left(1 - \frac{z}{z_{prp}}\right)}$$

$$y_p = y \times \frac{\left(\frac{d_p}{z_{prp}}\right)}{\left(\frac{z_{prp}}{z_{prp}} - z\right)} = y \times \frac{1}{\left(1 - \frac{z}{z_{prp}}\right)}$$

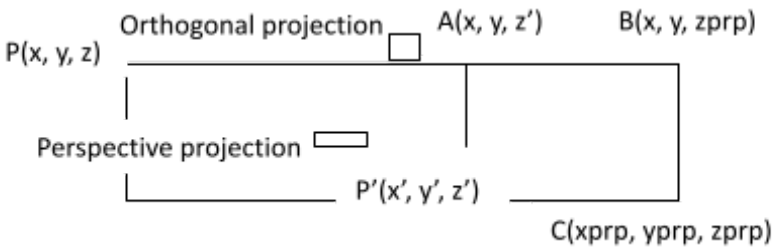
- ii. When  $z_{prp} = 0$ , reference point at origin

$$x_p = x \times \frac{\frac{z_{vp}}{z}}{z} = x \times \frac{1}{\left(\frac{z}{z_{vp}}\right)}$$

$$y_p = y \times \frac{\frac{z_{vp}}{z}}{z} = y \times \frac{1}{\left(\frac{z}{z_{vp}}\right)}$$

### Perspective projection derivation from orthogonal projection

Suppose that  $P(x, y, z)$  is projected orthogonally along  $z$  direction and the viewing axis is along  $z$  axis. The point  $P(x, y, z)$  is projected perspectively as shown in the figure.



$\Delta PAP'$  and  $\Delta PBC$  are similar triangles

$$AP' / BC = PA / PB$$

$$u = AP' / BC = PA / PB \dots\dots\dots(1)$$

$$u = (x - x') / (x - x_{prp}) = (z - z') / (z - z_{prp})$$

$$u = (x - x') / (x - x_{prp})$$

$$x - x' = (x - x_{prp}) u$$

$$x' = x - (x - x_{prp}) u \dots \dots \dots (2)$$

similarly,

$$y' = y - (y - y_{prp}) u \dots \dots \dots (3)$$

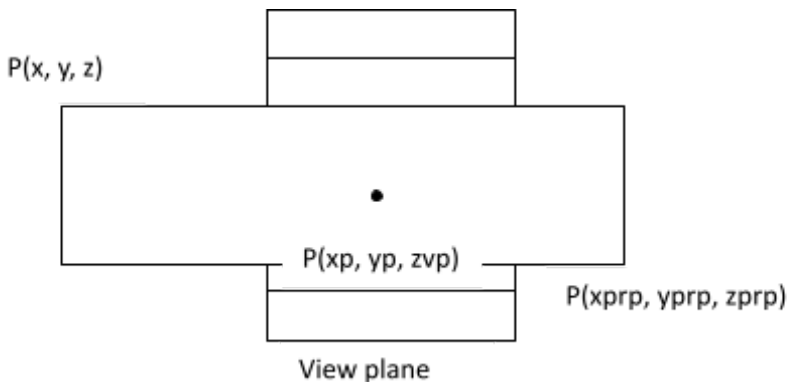
Again,

$$u = PA / PB$$

$$u = (z - z') / (z - z_{prp})$$

$$(z - z') = (z - z_{prp}) u$$

$$z' = z - (z - z_{prp}) u \dots \dots \dots (4)$$



From equation 2, 3, and 4,

When  $u = 0$ ,

$$x' = x$$

$$y' = y$$

$$z' = z$$

When  $u = 1$ ,

$$x' = x_{prp}$$

$$y' = y_{prp}$$

$$z' = z_{prp}$$

At view plan

$$z' = z_{vp}$$

From equation number 4

$$z_{vp} = z - (z - z_{prp}) u$$

$$u = (z_{vp} - z) / (z_{prp} - z)$$

At view plane

$$x' = x_p$$

From equation number 2,

$$x_p = x - (x - x_{prp}) (z_{vp} - z) / (z_{prp} - z)$$

$$x_p = x (z_{prp} - z_{vp}) / (z_{prp} - z) + x_{prp} (z_{vp} - z) / (z_{prp} - z) \dots\dots\dots(5)$$

similarly,

$$y_p = y (z_{prp} - z_{vp}) / (z_{prp} - z) + y_{prp} (z_{vp} - z) / (z_{prp} - z) \dots\dots\dots(6)$$

Special cases,

- (i) Projection reference point on  $Z_{view}$  axis

$$x_{prp} = 0$$

$$y_{prp} = 0$$

From equation number 5 and 6

$$x_p = x (z_{prp} - z_{vp}) / (z_{prp} - z)$$

$$y_p = y (z_{prp} - z_{vp}) / (z_{prp} - z)$$

- (ii) Projection reference point at origin

$$x_{prp} = 0$$

$$y_{prp} = 0$$

$$z_{prp} = 0$$

From equation number 5 and 6,

$$x_p = x (z_{vp} / z)$$

$$y_p = y (z_{vp} / z)$$

- (iii) If the view plane is the uv plane (xy plane)

$$z_{vp} = 0$$

$$x_p = x (z_{prp} / z_{prp} - z) - x_{prp} (z / z_{prp} - z)$$

$$y_p = y (z_{prp} / z_{prp} - z) - y_{prp} (z / z_{prp} - z)$$

- (iv) If the view plane is uv plane and projection reference point on  $Z$  view axis

$$x_{prp} = 0$$

$$y_{prp} = 0$$

$$z_{vp} = 0$$

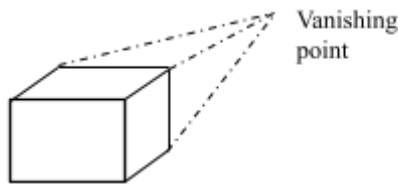
From equation number 5 and 6,

$$x_p = x (z_{prp} / (z/z_{prp} - z))$$

$$y_p = y (z_{prp} / (z/z_{prp} - z))$$

## 4.6 Vanishing Point

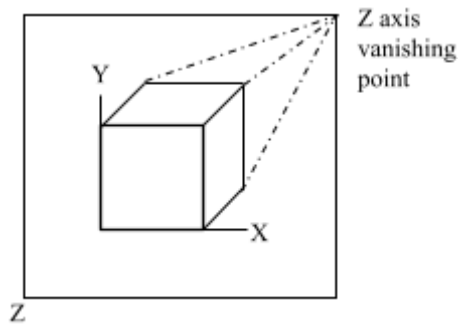
- A set of parallel lines that are not parallel to view plane are projected as converging lines that appear to converge at a point called vanishing point.
- A set of parallel lines that are parallel to view plane are projected as parallel lines.
- More than one set of parallel lines form more than one vanishing point in the scene.
- The use of vanishing point is for realistic representation



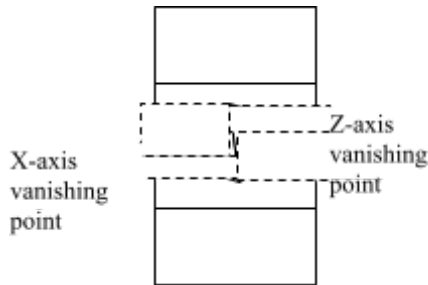
**Figure 4.20 (a):** Perspective views and vanishing point

### Principal vanishing point

- Principal vanishing points are formed by the apparent intersection of lines parallel to one of the three principal x, y, z axes
- The vanishing point for any set of lines that are parallel to one of the principal axes of an object is referred to as a principal vanishing point.
- The number of principal vanishing point is determined by number of principal axes intersected by the view plane.



**Figure 4.20 (b):** Perspective views and z axis vanishing point

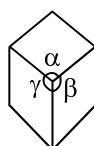
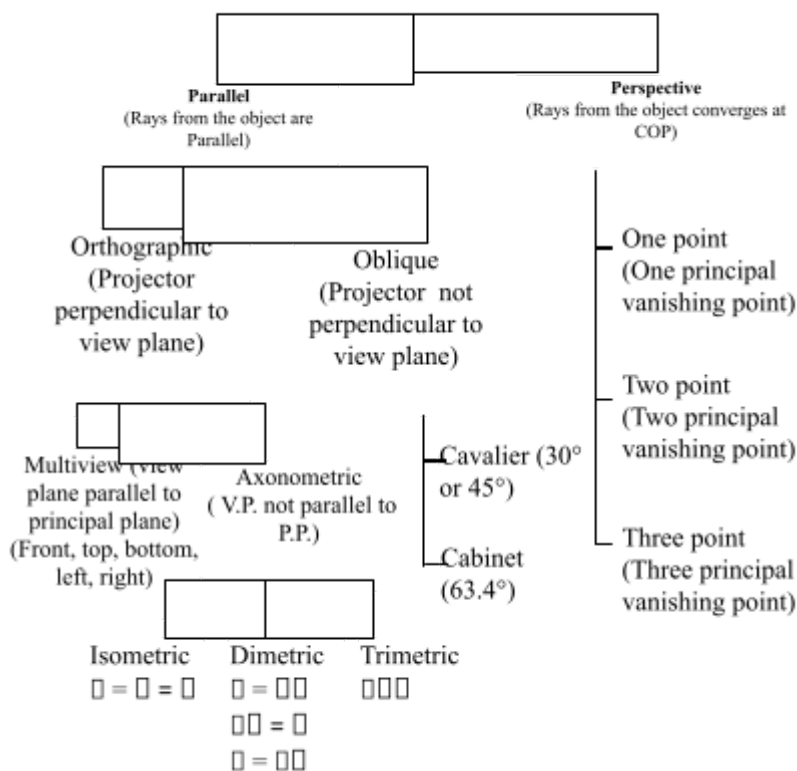


**Figure 4.20 (c):** Two-point perspective views and principal vanishing point

- We can control the number of principal vanishing point to one, two or three with the orientation of projection plane and classify as one, two or three point perspective projections.
- Sets of parallel lines on the same plane lead to collinear vanishing points. The line is called the horizon of the plane.



## Projections



---

**1. Derive the single 3D transformation matrix for the reflection through the plane containing the points with coordinates (5,0,0) and (0,0,5) and being parallel to the y-axis.**

Solution hints,

Recalling, that  $\cos(-45^\circ)=0.7$  and  $\sin(-45^\circ)=-0.7$ , the transformation is as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.7 & 0 & 0.7 & 0 \\ 0 & 1 & 0 & 0 \\ -0.7 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.7 & 0 & -0.7 & 0 \\ 0 & 1 & 0 & 0 \\ 0.7 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \\ = \begin{bmatrix} 0 & 0 & -1 & 5 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**2. Derive a single matrix for the rotation about a vertical axis being parallel to the y-axis followed by the rotation about a horizontal axis being parallel to the x-axis. Both axes pass through the point with coordinates  $(x_0, y_0, z_0)$ .**

Solution hints,

Assume that  $x_0, y_0, z_0$  is a middle point of the object to be drawn.

$$[P] = [Trans_{x_0, y_0, z_0}] [Rot_{ox}] [Rot_{oy}] [Trans_{-x_0, -y_0, -z_0}] [P]$$

**3. Derive the single 3D transformation matrix for the reflection through the point with coordinates (5, 10, -5) followed by the counter-clockwise rotation by 90 about a vertical axis passing through this point and being parallel to the y-axis.**

Solution hints,

$$P' = TP$$

$$T = \begin{matrix} Trans(5,10,-5) & Rot_y(90) & Refl(0,0,0) & Trans(-5,10,5) \end{matrix} \\ = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & -5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -5 \\ 0 & 1 & 0 & -10 \\ 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -1 & 0 \\ 0 & -1 & 0 & 20 \\ 1 & 0 & 0 & -10 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**4. What is the minimum number of points which can define any arbitrary affine transformation?**

Solution hints,

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} a & d & g & l \\ b & e & h & m \\ c & f & i & n \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad \begin{aligned} x &= ax + dy + gz + l \\ y &= bx + ey + hz + m \\ z &= cx + fy + iz + n \end{aligned}$$

There are 12 unknown variables a,b,c,d,e,f,g,h,i,j,k,l.

12/3=4 points.

**5. A unit cube with vertices at points (0,0,0), (0,1,0), (1,1,0), (1,0,0), (0,0,1), (0,1,1), (1,1,1), (1,0,1) is transformed into a prism with the respective vertices at points (-1,0,0), (-1,2,0), (2,0,0), (2,-2,0), (0,0,2), (0,2,2), (3,0,2), (3,-2,2). Find a single transformation matrix performing this transformation.**

*Solution hints,*

*We have to choose 4 pairs of the corresponding points.*

*By selecting the following pair (0,0,0)→(-1,0,0) we get:*

$$l = -1 \qquad m = 0 \qquad n = 0$$

*By selecting (1,0,0) → (2,-2,0) we obtain:*

$$a - 1 = 2 \Rightarrow a = 3 \qquad b = -2 \Rightarrow b = -2 \qquad c = 0 \Rightarrow c = 0$$

*By selecting (0,1,0)→(-1,2,0) we obtain:*

$$d - 1 = -1 \Rightarrow d = 0 \qquad e + 0 = 2 \Rightarrow e = 2 \qquad f + 0 = 0 \Rightarrow f = 0$$

*By selecting (0,0,1) → (0,0,2) we obtain:*

$$g - 1 = 0 \Rightarrow g = 1 \qquad h = 0 \Rightarrow h = 0 \qquad i = 2 \Rightarrow i = 2$$

*Thus, the transformation is defined by the following matrix*

$$\begin{bmatrix} 3 & 0 & 1 & -1 \\ -2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**6. Given a 3D triangle with points (0, 0, 0), (1, 1, 2) and (1, 1, 3). Apply shear parameter 2 on X axis, 2 on Y axis and 3 on Z axis and find out the new coordinates of the object.**

Solution,

Given,

- Old corner coordinates of the triangle = A (0, 0, 0), B(1, 1, 2), C(1, 1, 3)
- Shearing parameter towards X direction ( $Sh_x$ ) = 2
- Shearing parameter towards Y direction ( $Sh_y$ ) = 2
- Shearing parameter towards Y direction ( $Sh_z$ ) = 3

Shearing in X Axis-

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing = ( $X_{new}$ ,  $Y_{new}$ ,  $Z_{new}$ ).

Applying the shearing equations, we have-

- $X_{new} = X_{old} = 0$
- $Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$
- $Z_{new} = Z_{old} + Sh_z \times X_{old} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} = 1$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times X_{\text{old}} = 1 + 2 \times 1 = 3$
- $Z_{\text{new}} = Z_{\text{old}} + Sh_z \times X_{\text{old}} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (1, 3, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} = 1$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times X_{\text{old}} = 1 + 2 \times 1 = 3$
- $Z_{\text{new}} = Z_{\text{old}} + Sh_z \times X_{\text{old}} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (1, 3, 6).

Thus, New coordinates of the triangle after shearing in X axis = A (0, 0, 0), B(1, 3, 5), C(1, 3, 6).

Shearing in Y Axis,

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Y_{\text{old}} = 0 + 2 \times 0 = 0$
- $Y_{\text{new}} = Y_{\text{old}} = 0$
- $Z_{\text{new}} = Z_{\text{old}} + Sh_z \times Y_{\text{old}} = 0 + 3 \times 0 = 0$

Thus, New coordinates of corner A after shearing = (0, 0, 0).

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing = ( $X_{\text{new}}$ ,  $Y_{\text{new}}$ ,  $Z_{\text{new}}$ ).

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Y_{\text{old}} = 1 + 2 \times 1 = 3$
- $Y_{\text{new}} = Y_{\text{old}} = 1$
- $Z_{\text{new}} = Z_{\text{old}} + Sh_z \times Y_{\text{old}} = 2 + 3 \times 1 = 5$

Thus, New coordinates of corner B after shearing = (3, 1, 5).

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing = ( $X_{\text{new}}$ ,  $Y_{\text{new}}$ ,  $Z_{\text{new}}$ ).

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Y_{\text{old}} = 1 + 2 \times 1 = 3$
- $Y_{\text{new}} = Y_{\text{old}} = 1$
- $Z_{\text{new}} = Z_{\text{old}} + Sh_z \times Y_{\text{old}} = 3 + 3 \times 1 = 6$

Thus, New coordinates of corner C after shearing = (3, 1, 6).

Thus, New coordinates of the triangle after shearing in Y axis = A (0, 0, 0), B(3, 1, 5), C(3, 1, 6).

Shearing in Z Axis,

For Coordinates A(0, 0, 0)

Let the new coordinates of corner A after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Z_{\text{old}} = 0 + 2 \times 0 = 0$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times Z_{\text{old}} = 0 + 2 \times 0 = 0$
- $Z_{\text{new}} = Z_{\text{old}} = 0$

Thus, New coordinates of corner A after shearing =  $(0, 0, 0)$ .

For Coordinates B(1, 1, 2)

Let the new coordinates of corner B after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Z_{\text{old}} = 1 + 2 \times 2 = 5$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times Z_{\text{old}} = 1 + 2 \times 2 = 5$
- $Z_{\text{new}} = Z_{\text{old}} = 2$

Thus, New coordinates of corner B after shearing =  $(5, 5, 2)$ .

For Coordinates C(1, 1, 3)

Let the new coordinates of corner C after shearing =  $(X_{\text{new}}, Y_{\text{new}}, Z_{\text{new}})$ .

Applying the shearing equations, we have-

- $X_{\text{new}} = X_{\text{old}} + Sh_x \times Z_{\text{old}} = 1 + 2 \times 3 = 7$
- $Y_{\text{new}} = Y_{\text{old}} + Sh_y \times Z_{\text{old}} = 1 + 2 \times 3 = 7$
- $Z_{\text{new}} = Z_{\text{old}} = 3$

Thus, New coordinates of corner C after shearing =  $(7, 7, 3)$ .

Thus, New coordinates of the triangle after shearing in Z axis = A (0, 0, 0), B(5, 5, 2), C(7, 7, 3).

1. A unit length cube with diagonal passing through (0, 0, 0) and (2, 2, 2) is shared with respect to zx-plane with share constants = 3 in both directions. Obtain the final coordinates of the cube after shearing. [2075 Ashwin]

**Solution:**

Shearing with respect to zx-plane

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & sh_x & 0 & 0 & 0 & 1 & 0 & 0 & 0 & sh_z & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

$$sh_x = sh_y = 3$$

$$A = (2, 0, 0)$$

$$B = (2, 2, 0)$$

$$C = (0, 2, 0)$$

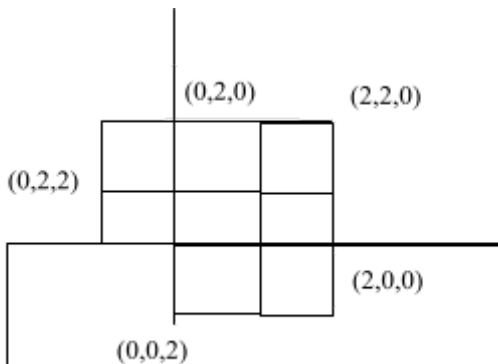
$$D = (0, 2, 2)$$

$$E = (0, 0, 2)$$

$$F = (0, 2, 2)$$

$$G = (2, 2, 2)$$

$$H = (0, 0, 0)$$





$$A' = [1 \ 3 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 3 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1] [2 \ 0 \ 0 \ 1]$$

$$A' = (2, 0, 0)$$

Similarly, we can calculate B', C', D', E', F', G', H'.

$$B' = (8, 2, 6)$$

$$C' = (6, 2, 6)$$

$$D' = (6, 2, 6)$$

$$E' = (0, 0, 0)$$

$$F' = (6, 2, 6)$$

$$G' = (8, 2, 6)$$

$$H' = (0, 0, 0)$$

The final coordinates of the cube after shearing are A'(2, 0, 0), B'(8, 2, 6), C'(6, 2, 6), D'(6, 2, 6), E'(0, 0, 0), F'(6, 2, 6), G'(8, 2, 6), and H'(0, 0, 0)

2. *List down the steps for rotating a 3D object by 90° in counter clockwise direction about an axis joining end points (1, 2, 3) and (10, 20, 30). Also derive the final transformation matrix.* [2076 Ashwin]

**Steps:**

- i. Translate (1, 2, 3) to origin so that the rotation axis passes through the origin.
- ii. Rotate the line so that the line coincides with one of the axes, say 2 axis.
- iii. Rotate the object about that co-ordinate axis by 90° in counter clockwise direction.
- iv. Apply the inverse of step (ii) i.e. inverse rotation to bring the rotation axis back to its original orientation.
- v. Apply the inverse of step (i) i.e. inverse translation to bring the rotation axis back to its original position.

For step (ii) i.e. for coinciding the arbitrary axis with any co-ordinate axis, the rotations are needed about other two axes.

Direction cosines of the given line is

$$[v] = [(x_1 - x_0) (y_1 - y_0) (z_1 - z_0)]$$

$$[c_x \ c_y \ c_z] =$$

$$c_x = = =$$

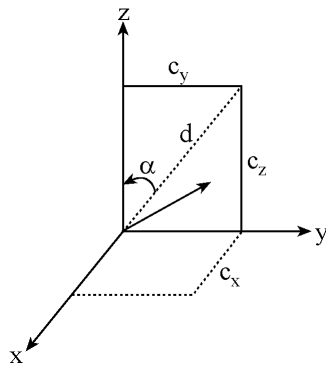
$$c_y = =$$

$$c_z = =$$

To calculate the angles of rotation about x and y axes we use the direction cosines.

To put the line or rotation axis on the z axis we have to follow two steps

- First rotate about the x axis to transform vector u into the x z plane.
- The swing u around to the z axis using 4 axis rotation.



$$d =$$

$$\cos \alpha =$$

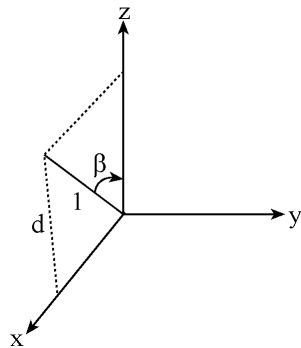
$$\sin \alpha =$$

$$\cos \beta = d$$

$$\sin \beta = -c_x$$

The complete the sequence of operations can be summarized as

$$T = [T_r]^{-1} [R_x(\alpha)]^{-1} [R_y(\beta)]^{-1} [R_z(\theta)] [R_y(\beta)] [R_x(\alpha)] [T_r]$$



$$[T_r] =$$

$$[R_x(\alpha)] =$$

$$[R_y(\beta)] =$$

$$[R_z(\theta)] =$$

$$[R_y(\beta)] =$$

$$[R_x(\alpha)]^{-1} =$$

$$[T_r]^{-1} =$$

3. *Obtain perspective projection co-ordinates for the pyramid with vertices of base (15, 15,10), (20,20,10), (25,15,10), (20,10,10) and apex (20,15,20) given that  $z_{prp}=20$  and  $z_{vp}=0$ .*

**Solution:**

$$z_{prp} = 20, z_{vp} = 0$$

$$x' = x - xu$$

$$y' = y - yu$$

$$z' = z - (z - z_{prp})u$$

On the view plane,  $z' = z_{vp}$ . So,

$$z_{vp} = z - (z - z_{prp})u$$

$$u =$$

For the vertex (15,15,10),

$$X_p = x - x \left( \frac{z_{vp} - z}{z_{prp} - z} \right)$$

$$x \left( \frac{z_{prp} - z - z_{vp} + z}{z_{prp} - z} \right)$$

$$x \left( \frac{z_{prp} - z_{vp}}{z_{prp} - z} \right)$$

$$15 \left( \frac{20-0}{20-10} \right)$$

30

$$y_p = y - y \left( \frac{Z_{vp} - Z}{Z_{pp} - Z} \right)$$

$$10 - 10 \left( \frac{0-10}{20-10} \right)$$

$$= 30$$

$$Z_p = Z_{vp} = 0$$

Projected points is  $(X'_1, Y'_1, Z'_1) = (30, 30, 0)$

Similarly for  $P_2(X_2, Y_2, Z_2) = P_2(20, 20, 10)$

$$X_p = 20 \left( \frac{20-0}{20-10} \right) = 40$$

$$Y_p = 20 \left( \frac{20-0}{20-10} \right) = 40$$

$$Z_p = 0$$

Projected points is  $(40, 40, 0)$

Similarly, for vertex  $(25, 15, 10)$

$$X_p = 50$$

$$Y_p = 30$$

$$Z_p = 0$$

Projected points is  $P_3) = (50, 30, 0)$

for vertex  $(20, 10, 10)$

$$X_p = x - x \left( \frac{Z_{vp} - z}{Z_{pp} - Z} \right)$$

$$20 - 20 \left( \frac{0-10}{20-10} \right)$$

$$Y_p = y - y \left( \frac{Z_{vp} - z}{Z_{pp} - Z} \right)$$

$$10 - 10 \left( \frac{0-10}{20-10} \right)$$

$$20$$

$$Z_p = 0$$

For apex (20, 15, 20)

$$X_p = X\left(\frac{d_p}{Z_{pp} - Z}\right)$$

$$20\left(\frac{20}{20-20}\right)$$

$$= \infty$$

$$Y_p = Y\left(\frac{d_p}{Z_{pp} - Z}\right)$$

$$15\left(\frac{20}{20-20}\right)$$

$$= \infty$$

$$Z_p = 0$$

4. *A unit length cube with diagonal passing through (0, 0, 0) and (1, 1, 1) is shared with respect to yz-plane with share constants = 2 in both directions. Obtain the final coordinates of the cube after shearing.* [2078 Bhadra]

**Solution:**

Shearing with respect to zx-plane

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & sh_x & 1 & 0 & 0 & sh_z & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x & y & z & 1 \end{bmatrix}$$

$$sh_x = sh_y = 2$$

$$A = (1, 0, 0)$$

$$B = (1, 1, 0)$$

$$C = (0, 1, 0)$$

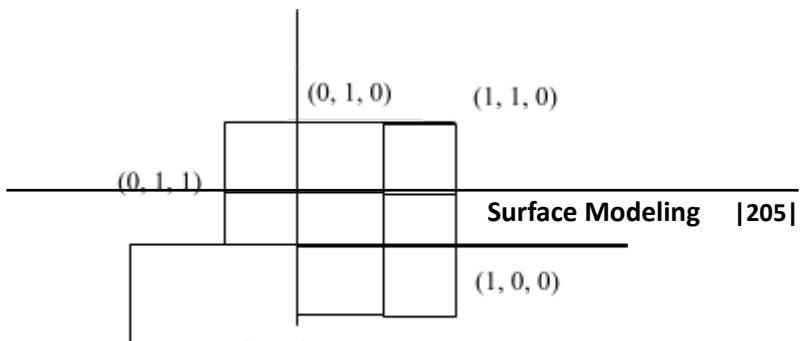
$$D = (0, 1, 1)$$

$$E = (0, 0, 1)$$

$$F = (0, 1, 1)$$

$$G = (1, 1, 1)$$

$$H = (0, 0, 0)$$



$$A' = [1\ 0\ 0\ 0\ 2\ 1\ 0\ 0\ 2\ 0\ 0\ 0\ 0\ 1\ 1] [1\ 0\ 0\ 1]$$

$$A' = (1, 2, 2)$$

Similarly, we can calculate B', C', D', E', F', G', H'

$$B' = (1, 3, 2)$$

$$C' = (0, 1, 0)$$

$$D' = (0, 1, 0)$$

$$E' = (0, 0, 0)$$

$$F' = (0, 1, 0)$$

$$G' = (1, 3, 2)$$

$$H' = (0, 0, 0)$$

The final coordinates of the cube after shearing are A'(1, 2, 2), B'(1, 3, 2), C'(0, 1, 0), D'(0, 1, 0), E'(0, 0, 0), F'(0, 1, 0), G'(1, 3, 2), and H'(0, 0, 0)



---

## Curve Modeling

---

Spline is a specific type of curve made up of piecewise polynomial segments. Polynomials are mathematical expressions involving a sum of powers in one or more variables multiplied by coefficients. Curve is a broad term for any smooth, continuous path.

Splines are mathematical functions used to create smooth curves through a series of points.

A spline is a piecewise-defined polynomial function used to approximate or interpolate a set of data points. Splines ensure smoothness at the points where polynomial pieces join, providing continuity in the first and often higher derivatives. Common types of splines include linear splines, quadratic splines, and cubic splines, with cubic splines being the most widely used.

A curve is a continuous and smooth flowing line without any sharp turns. Curves can be described mathematically using equations in parametric or implicit forms. Examples include circles, ellipses, parabolas, and other higher-order curves.

## 5.1 Spline Representations

### 5.1.1 Curve, Spline and Surface

#### Curve

A **curve** is a general term that refers to any smooth, continuous line or path that can be defined mathematically or drawn graphically. Curves can be represented in various ways, including:

- **Implicitly:** By an equation  $F(x,y)=0$
- **Explicitly:** By a function  $y=f(x)$
- **Parametrically:** By functions  $x=g(t)$  and  $y=h(t)$ , where  $t$  is a parameter

Curves can be simple, such as a straight line or a circle, or more complex, like a parabola or an ellipse. They are used extensively in geometry, calculus, and computer graphics.

#### Spline

A **spline** is a specific type of curve that is constructed using piecewise polynomial functions. The term originates from the flexible spline devices used by draftsmen to draw smooth curves before the advent of computer graphics. There are several types of splines, including:

- **Linear Splines:** Connect data points with straight line segments.
- **Quadratic and Cubic Splines:** Use second or third-degree polynomials to ensure smoothness at the data points.
- **B-Splines (Basis Splines):** Offer greater control and flexibility in shape modeling.
- **NURBS (Non-Uniform Rational B-Splines):** Extend B-splines by adding weights, allowing for the representation of more complex shapes, including circles and ellipses.



Splines are particularly useful in computer graphics, data fitting, and numerical analysis because they provide a smooth approximation to a set of data points or a desired shape.

### **Key Differences**

#### **1. General vs. Specific:**

- **Curve:** A broad term for any smooth, continuous path.
- **Spline:** A specific type of curve made up of piecewise polynomial segments.

#### **2. Mathematical Representation:**

- **Curve:** Can be represented in various forms (implicit, explicit, parametric).
- **Spline:** Specifically represented by piecewise polynomials, often with additional constraints for smoothness.

#### **3. Applications:**

- **Curve:** Used in general mathematical contexts and graphical representations.
- **Spline:** Used extensively in computer-aided design (CAD), computer graphics, and data interpolation for its smoothness and flexibility.

#### **4. Control and Flexibility:**

- **Curve:** The shape is generally defined by its equation or parameterization.
- **Spline:** Provides localized control over the shape, allowing adjustments at specific intervals or knots without affecting the entire curve.

Spline curve or simply spline is continuous curve that are formed with polynomial pieces with certain boundary conditions. A spline surface is combination of spline curves. A spline surface can be described with two sets of orthogonal spline curves.

In drafting terminology, a spline is a flexible strip used to produce a smooth curve through a designated set of points. Several small weights are distributed along the length of the strip to hold the position of spline curve as required. We can mathematically describe such a curve with a piecewise cubic polynomial function whose first and second derivatives are continuous across the various curve sections. The general shape of a spline curve is indicated by a set of coordinate positions called **control points**. A spline curve is defined, modified, and manipulated with operations on the control points.

Splines are used in graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation path for the objects or image. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

Cubic splines are piecewise-defined polynomials of degree three.

Polynomials are mathematical expressions involving a sum of powers in one or more variables multiplied by coefficients.

A polynomial in a single variable  $x$  of degree  $n$  can be written as:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

where:

- $a_n, a_{n-1}, \dots, a_1, a_0$  (with  $a_n \neq 0$  if  $n > 0$ ).
- $n$  is a non-negative integer called the degree of the polynomial.
- $x$  is the variable.

**The different types of spline curve are:**

1. Piecewise cubic spline
  - i. Hermite spline
  - ii. Cardinal spline

- iii. Kochanek-Bartels spline
- 2. Bezier spline
- 3. B-spline
- 4. Beta spline

### 5.1.2 Interpolation and Approximation Splines

---

We specify a spline curve by giving a set of coordinate positions, called control points, which indicates the general shape of curve. The control points are fitted with piecewise continuous parametric polynomial function in one of the two ways.



**Fig 5.1(a):** A set of control points interpolated with piecewise continuous polynomial sections

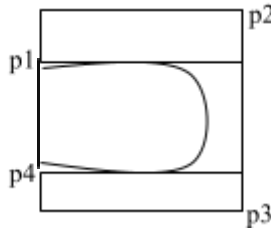
**Fig 5.1(b):** A set of six points control points approximated with piecewise continuous polynomial sections

When polynomial sections are fitted so that the curve passes through each control point as in **Figure a**, the resulting curve is said to interpolate the set of control points. Interpolation curves are commonly used to digitize drawings or to specify animation paths, and graphs of data trends of discrete set of data points.

When the polynomials are fitted to the general control point path without necessarily passing through any control point, the resulting curve is said to approximate the set of control points (**Figure b**). Approximation curves are primary used as design tools to structure objects surfaces. It is used in drawing contour lines for GIS (Geographical Information System) applications.

## Convex hull

The convex polygon boundary that encloses a set of control points is called the convex hull. Convex hulls provide a measure for the deviation of a curve or surface from the region bounding the control points. Some splines are bounded by the convex hull that ensures the polynomials smoothly follow the control points without erratic oscillations. Also, the polygon region inside the convex hull is useful in some algorithms as a clipping region.



*Fig 5.2: Convex hull*

## Curve Continuity or Smoothness of Curve

There are two approaches which determine the smoothness of curve or curve continuity. They are as follows:

- i. Parametric continuity conditions(C)
- ii. Geometric continuity conditions (G)

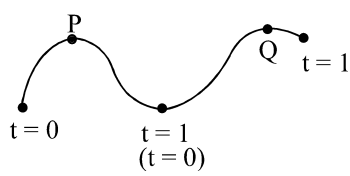
### 5.1.3 Parametric Continuity Conditions

---

Parametric continuity deals in parametric equations associated to piecewise parametric polynomial curve not the shape or appearance of the curve. We set parametric continuity by matching the parametric derivatives of adjoining curve sections at their common boundary.

#### i. Zero order parametric continuity ( $C^0$ ):

$C^0$  continuity means that two piece of curves are joined or meet at same point. The two pieces of curve P and Q are in zero order parametric continuity if  $P(t=1) = Q(t=0)$



**ii. First order parametric continuity ( $C^1$ ):**

Two successive curve sections are in first order parametric continuity if first parametric derivative of the coordinate function are equal at the joining point.

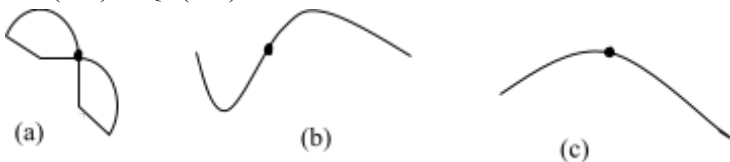
$$P'(t=1) = Q'(t=0)$$

Where  $P'$  and  $Q'$  are first order derivative.

**iii. Second order parametric continuity ( $C^2$ ):**

Two curve are in second order parametric continuity if both first and second derivatives of the two curve sections are same at the intersection point.

$$P''(t=1) = Q''(t=0)$$



**Figure 5.3:** Piecewise construction of a curve by joining two curve segments using different order of continuity (a) zero-order continuity only (b) first-order continuity (c) second-order continuity.

In summary we can conclude that,

- $C_0$ : curves are joined
- $C_1$ : first derivatives equal
- $C_2$ : first and second derivatives are equal and If  $t$  is taken to be time, this implies that the acceleration is continuous.
- $C_n$ :  $n^{\text{th}}$  derivatives are equal

### 5.1.4 Geometric Continuity Conditions

---

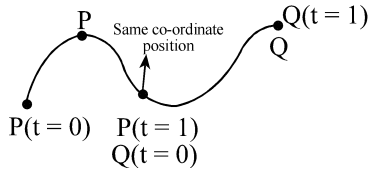
Another method for joining two successive curve sections is to specify conditions for geometric continuity. Geometric continuity refers to the way that a curve or surface looks. In this case, we only require parametric derivatives of the two sections to be proportional to each other at their common boundary instead of equal to each other.

**i. Zero order geometric continuity ( $G^0$ ):**

Zero-order geometric continuity ( $G^0$  continuity) is the same as zero-order parametric continuity. That is, the two curves sections must have the same coordinate position at the boundary point.

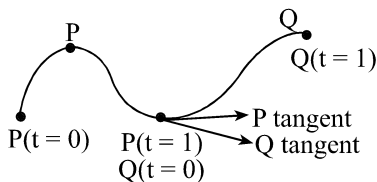
$$P(t=1) = Q(t=0)$$

P and Q are two segments of curves.



**ii. First order geometric continuity ( $G^1$ ):**

First-order geometric continuity ( $G^1$  continuity) means that the parametric first derivatives are proportional at the intersection of two successive sections. If P and Q are two piece of curves, then  $P'(t)$  and  $Q'(t)$  must have same direction of tangent vector but not necessary to be the same magnitude.

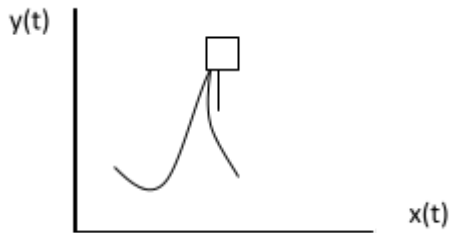


**iii. Second order geometric continuity ( $G^2$ ):**

Second-order geometric continuity ( $G^2$  continuity) means that both the first and second parametric derivatives of the two curve sections are proportional at their boundary. Under  $G^2$  continuity, curvatures of two curve sections will match at the joining position.

In general  $C1$  continuity implies  $G1$  continuity but  $G1$  continuity doesn't imply  $C1$  continuity.

C1 continuity doesn't imply G1 continuity when segments tangent vector are  $[0 \ 0 \ 0]$  at join point. In this case, the tangent vectors are equal but their directions are different.



In summary we can conclude that,

- G0: curves are joined
- G1: first derivatives are proportional at the join point and the curve tangents thus have the same direction, but not necessarily the same magnitude. i.e.,  $C1'(1) = (a,b,c)$  and  $C2'(0) = (k*a, k*b, k*c)$ .
- G2: first and second derivatives are proportional at join point

As their names imply, geometric continuity requires the geometry to be continuous, while parametric continuity requires that the underlying parameterization be continuous as well.

## Splines

- splines are cubic curves which maintain C2 continuity.
- natural spline
  - ◆ interpolates all of its control points.
  - ◆ equivalent to a thin strip of metal forced to pass through control points
  - ◆ no local control
- B-spline
  - ◆ local control
  - ◆ does not interpolate control points



## Three types of Curve

There are three types of curve. They are:

i. Open curve:



ii. Closed curve:



iii. Crossing curve:



## Representation of Curve

All objects are not flat but may have many bends and deviations. We have to compute all curves. We can represent curve by three mathematical function:

- i. Explicit function
- ii. Implicit function
- iii. Parametric function

### i. **Explicit representation of curve:**

In this method the dependent variable is given explicitly in terms of the independent variable as;

$$y=f(x)$$

e.g.,  $y= mx + c$

$$y = 5x^2 + 2x + 1$$

In explicit representation, for each single value of  $x$ , only a single value of  $y$  is computed

### ii. **Implicit representation of curve:**

In this method, dependent variable is not expressed in terms of some independent variables as;

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

e.g.;

$$x^2 + y^2 - 1 = 0$$

In implicit representation, for each single value of  $x$ , multiple values of  $y$  is computed.

If we convert implicit function to explicit function it will be more complex and will give different values.

$$\text{e.g. } y = \pm \sqrt{1 - x^2}$$

### iii. Parametric representation of curve:

We cannot represent all curves in single equation in terms of only  $x$  and  $y$ . Instead of defining  $y$  in terms of  $x$  (i.e.  $y = f(x)$ ) or  $x$  in terms of  $y$  (i.e.  $x = h(y)$ ); we define both  $x$  and  $y$  in terms of a third variable in parametric form.

Curves having parametric form are called parametric curves.

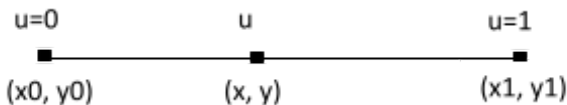
$$x = f_x(u)$$

$$y = f_y(u) \text{ where } u \text{ is parameter}$$

Similarly, parametric equation of line is;

$$x = (1 - u) x_0 + u x_1$$

$$y = (1 - u) y_0 + u y_1$$



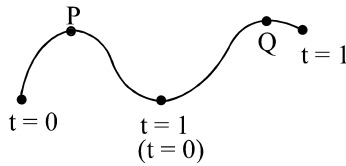
### Parametric Curve

The parametric representation for curve is as follows:

$$x = x(t)$$

$$y = y(t)$$

$$z = z(t)$$



A curve is approximated by a piecewise polynomial curve instead of piecewise linear curve. Piecewise linear curve is represented by linear equation and polylines. Piecewise polynomial curve is represented by polynomial equation.

Cubic polynomial means the polynomials which represent the curve with degree three.

The cubic polynomial that define a curve can be represented as

$$Q(t) = [x(t) \quad y(t) \quad z(t)]$$

A parametric cubic curve is defined as

$$Q(t) = \sum_{i=0}^3 a_i t^i \quad 0 \leq t \leq 1$$

Cubic polynomial equation can be defined as,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

$$Q(t) = [t^3 \ t^2 \ t \ 1]$$

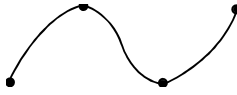
$$Q(t) = T.C.$$

$$C = M.G.$$

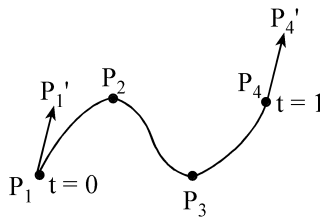
Where M is 4 X 4 basis matrix and G is a four element column vector of geometric constants called geometric vector.

## 5.2 Hermite Cubic Spline

- i. Hermite spline is interpolation spline (curve passes through control point)



- ii. It uses cubic polynomial function (to make Hermite function four point is necessary)
- iii. To make Hermite function it uses four point  $P_1, P_4, P_1', P_4'$ . Where  $P_1$  and  $P_4$  are position vector and  $P_1'$  and  $P_4'$  are tangent vectors (first order derivative) which show direction of the curve.



Let  $Q(t)$  is the curve  $t \in [0, 1]$

$Q(t) = [x(t) \ y(t) \ z(t)]$ ,  $t \in [0, 1]$  where all points satisfy cubic parametricity.

The general curve equation is,

$$p(t) = at^3 + bt^2 + ct + d \text{ where } 0 \leq t \leq 1$$

So,

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

$$Q(t) = [t^3 \quad t^2 \quad t \quad 1]$$

$$Q(t) = T \cdot C$$

$$C = M_H \cdot G$$

$M$  = basis matrix that provides blending function.

$G$  = Geometric vector

$$Q(t) = T \cdot M_H \cdot G_H$$

$$= [t^3 \quad t^2 \quad t \quad 1] \cdot M_H \cdot G_H$$

$$Q_x(t) = P_1(t) = [0 \ 0 \ 0 \ 1] M_H \cdot G_H$$

$$t = 0$$

$$Q_x(t) = P_4(t) [1 \ 1 \ 1 \ 1] M_H \cdot G_H$$

$$t = 1$$

$$Q'_x(t) = R_1(t) = [3t^2 \ 2t \ 1 \ 0] M_H \cdot G_H$$

$$t = 0$$

$$= [0 \ 0 \ 1 \ 0] M_H \cdot G_H$$

$$Q'_H(t) = R_a(t) = [3 \ 2 \ 1 \ 0] M_H \cdot G_H$$

$$t = 1$$

$$= M_H G_H$$

$$M_H \cdot G_H =$$

$$M_H =$$

$$G_{HX} =$$

$$Q(t) = T M_H G_H$$

$$Q(t) = [t^3 \ t^2 \ t \ 1]$$

$$= (2t^3 - 3t^2 + 1)P_1 + (-2t^2 + 3t^2)P_2 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4$$

$$= P_1 H_0(t) + P_4 H_1(t) + R_1 H_2(t) + R_4 H_3(t)$$

Where  $H_0(t)$ ,  $H_1(t)$ ,  $H_2(t)$ ,  $H_3(t)$  are Hermite blending function.

### Alternative way to derive Hermite curve

Parametric cubic curves are defined by parametric equations that describe the coordinates of points on the curve as a function of a single parameter, usually denoted as  $t$ . Cubic curves are particularly popular because they provide a good balance between flexibility and computational efficiency.

A parametric cubic curve can be described by a pair of parametric equations:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

There are several common types of parametric cubic curves:

1. **Bezier Curves:** Defined by four control points,  $P_0$ ,  $P_1$ ,  $P_2$  and  $P_3$ , the cubic Bezier curve is given by:

$$P(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3 \quad 0 \leq t \leq 1$$

**2.B-Spline Curves:** Defined using basis functions and a set of control points, B-splines provide more control over the curve shape. A cubic B-spline curve typically uses four control points at a time.

where  $t$  typically ranges from 0 to 1, and  $a_x, b_x, c_x, d_x$  and  $a_y, b_y, c_y, d_y$  are coefficients that determine the shape of the curve in the  $x$  and  $y$  directions.

A parametric cubic curve is defined as

$$P(t)=\sum_{i=0}^3 a_i t^i \quad 0\leq t\leq 1 \qquad \dots\dots\dots(i)$$

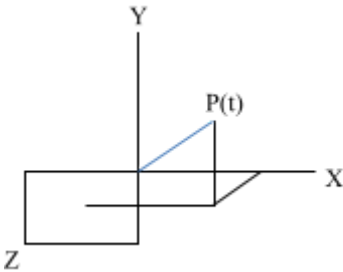
Where,  $P(t)$  is a point on the curve.

Expanding equation (i) yields,

$$P(t) = a_3t^3 + a_2t^2 + a_1t + a_0 \qquad \dots\dots\dots(ii)$$

Separating three components of  $P(t)$ , we have

$$\begin{aligned} x(t) &= a_{3x}t^3 + a_{2x}t^2 + a_{1x}t + a_{0x} \\ y(t) &= a_{3y}t^3 + a_{2y}t^2 + a_{1y}t + a_{0y} \\ z(t) &= a_{3z}t^3 + a_{2z}t^2 + a_{1z}t + a_{0z} \dots\dots\dots(iii) \end{aligned}$$



In order to solve equation (iii), twelve unknown coefficients must be specified.From the known end point coordinates of each

segment, six of the twelve needed equations are obtained. The other six are found by using tangent vectors at the two ends of each segment.

The direction of the tangent vectors establishes the slopes (direction cosines) of the curve at the end points. This procedure for defining a cubic curve using end points and tangent vector is one form of Hermite interpolation (cubic spline).

Each cubic curve segment is parameterized from 0 to 1 so that known end points correspond to the limit values of the parametric variable  $t$ , that is,  $P(0)$  and  $P(1)$ .

Substituting  $t = 0$  and  $t = 1$  in equation (ii), we have

$$P(0) = a_0 \qquad \qquad \qquad \dots\dots\dots (iv)$$

$$P(1) = a_3 + a_2 + a_1 + a_0 \qquad \dots\dots\dots (v)$$

To find the tangent vectors, equations (ii) must be differentiated with respect to  $t$ ,

$$P'(t) = 3a_3t^2 + 2a_2t + a_1$$

The tangent vectors at the two end points are found by substituting  $t = 0$  and  $t = 1$  in the above equation.

$$P'(0) = a_1 \qquad \qquad \qquad \dots\dots\dots (vi)$$

$$P'(1) = 3a_3 + 2a_2 + a_1 \qquad \dots\dots\dots (vii)$$

The values of  $a_2$  and  $a_3$  can be determined by solving equations (iv), (v), (vi), and (vii)

$$a_0 = P(0)$$

$$a_1 = P'(0)$$

$$a_2 = -3P(0) + 3P(1) - 2P'(0) - P'(1)$$

$$a_3 = 2P(0) - 2P(1) + P'(0) + P'(1)$$

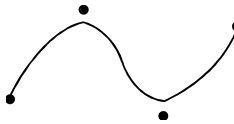
Substituting these values of  $a_i$  in equation (ii) and rearranging the terms yields

$$P(t) = (2t^3 - 3t^2 + 1)P(0) + (-2t^3 + 3t^2)P(1) + (t^3 - 2t^2 + t)P'(0) + (t^3 - t^2)P'(1)$$

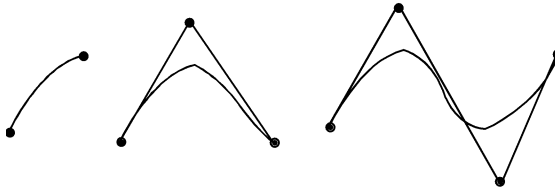
The values of  $P(0)$ ,  $P(1)$ ,  $P'(0)$ ,  $P'(1)$  are called geometric coefficients and represent the known vector quantities. The polynomial coefficients of these vector quantities are commonly known as blending functions. By varying parameter  $t$  in these blending functions from 0 to 1, several points on curve segments can be found.

### 5.3 Bezier Curves

- i. Bezier curve is approximation curve



- ii. Instead of end points and tangents, we use number of control points in Bezier curve. We use three control points to create quadratic curve, four control points to create cubic curve, five control points to create biquadratic curve in Bezier curve.
- iii. It has Bernstein polynomial function (Its own polynomial function to provide continuity)
- iv. All control points uses to make Bezier curve.



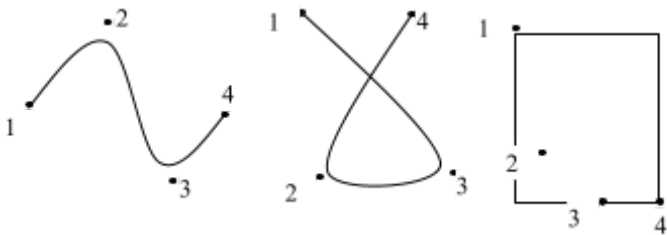
Curve doesn't go out of polygon boundary (Convex hull)

- Bezier curves are widely used in various CAD system COREL DRAW packages and many more Graphic packages.
- As with splines, a Bezier curve can be specified with boundary conditions with a characterizing matrix or with blending function. For general Bezier curves, the blending function specification is most convenient.



The Bezier curve was developed by the French engineer Pierre Bezier for use in the design of Renault automobile bodies. Bezier spline is highly useful and convenient for curve and surface design. They are also easy to implement. For these reasons, Bezier splines are widely available in various CAD systems, in general graphics packages, and in assorted drawing and painting packages.

The control points are blended using Bernstein polynomials to compute a set of position vectors  $\mathbf{P}(u)$  which are then joined by straight line segments to get the curve.



**Figure 5.4:** Bezier curves generated four control points

Suppose we are given  $n+1$  control-point positions:  $\mathbf{p}_k = (x_k, y_k, z_k)$  where  $k$  varies from 0 to  $n$ . These points can be blended to produce the position vector  $\mathbf{P}(u)$ , which describes the path of an approximating Bezier polynomial function between  $P_0$  and  $P_n$ .

$$\mathbf{P}(u) = \sum_{k=0}^n p_k BEZ_{k,n}(u) \dots\dots\dots (i) \quad \text{where } 0 \leq u \leq 1$$

The Bezier blending functions  $BEZ_{k,n}(u)$  are the Bernstein polynomials

$$BEZ_{k,n}(u) = C(n,k) u^k (1-u)^{n-k} \dots\dots\dots (ii)$$

where  $C(n,k)$  are the binomial coefficients and is given as

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

Equation (i) represents a set of three parametric equations for the individual curve coordinates,

$$P_x(u) = \sum_{k=0}^n x_k BEZ_{k,n}(u)$$

$$P_y(u) = \sum_{k=0}^n y_k BEZ_{k,n}(u)$$

$$P_z(u) = \sum_{k=0}^n z_k BEZ_{k,n}(u)$$

If we take  $n = 3$ , then number of control points =  $n+1 = 4$ .  
Then the above equations become

$$P_x(u) = x_0 BEZ_{0,3}(u) + x_1 BEZ_{1,3}(u) + x_2 BEZ_{2,3}(u) + x_3 BEZ_{3,3}(u)$$

$$P_y(u) = y_0 BEZ_{0,3}(u) + y_1 BEZ_{1,3}(u) + y_2 BEZ_{2,3}(u) + y_3 BEZ_{3,3}(u)$$

$$P_z(u) = z_0 BEZ_{0,3}(u) + z_1 BEZ_{1,3}(u) + z_2 BEZ_{2,3}(u) + z_3 BEZ_{3,3}(u)$$

where

$$BEZ_{0,3}(u) = C(3,0) u^0(1-u)^3 = (1-u)^3$$

$$BEZ_{1,3}(u) = C(3,1) u^1(1-u)^2 = 3u(1-u)^2$$

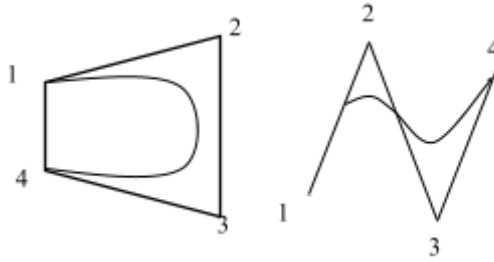
$$BEZ_{2,3}(u) = C(3,2) u^2(1-u) = 3u^2(1-u)$$

$$BEZ_{3,3}(u) = C(3,3) u^3(1-u)^0 = u^3$$

### Properties of Bezier curve:

1. Starting and ending control points lie on the curve.  
 $P(0) = P_0$   
 $P(1) = P_n$
2. They generally follow the shape of the control polygon which consists of the segments joining the control points.
3. Multiple control points of same coordinate values give more weight to that point and as a result, the curve is pulled nearer to that point.
4. The curve lies within the convex hull of the region created

by joining the control point. This is because  $\sum_{k=0}^n BEZ_{k,n} = 1$ .



**Figure 5.5:** Two dimensional Bezier curve

5. The slope at the beginning of the curve is along the line joining the first two control points, and the slope at the end of the curve is along the line joining the last two endpoints.
6. Collinear control points with same coordinate values produce a point.
7. Control point does not have local control over the shape of the curve.
8. The degree of the polynomial defining the curve segment is one less than the number of defining control polygon points. For 4 control points the degree of polynomial is 3. i.e. cubic Bezier curve.

#### **Drawback:**

- The degree of Bezier curve depends on number of control points.
- Bezier curve exhibit global control property means moving a control point alters the shape of the whole curve.

### **Bezier surfaces**

Two sets of orthogonal Bezier curves can be used to design an object surface by specifying by an input mesh of control points. The parametric vector functions for the Bezier surface is formed as the Cartesian product of Bezier blending functions

$$P(u, v) = \sum_{k=0}^n \sum_{j=0}^n p_{j,k} BEZ_{k,n}(u)$$

with  $p_{j,k}$  specifying the location of the  $(m+1)$  by  $(n+1)$  control points.

## 5.4 B-spline Curve

B-spline curve was developed to overcome the limitation or demerits of bezier curve. Demerits of Bezier curve are as follows:

1. Polynomial degree is decided by control points  
If C.P. = 5 then  
P.D. =  $5-1 = 4$
2. Blending function is non-zero for all parameter value over the entire curve. Due to this change in one vertex, changes the entire curve and this eliminates the ability to produce a local change within a curve.

### Properties of B-spline curve

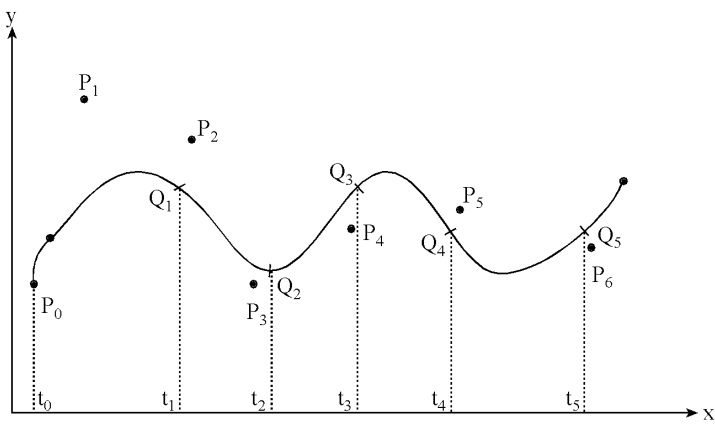
1. B-spline approximate spline curve with local effect. In this curve, each control point affects the shape of the curve only over range of parameter values where its associated basis function is non-zero.
2. B-spline curve made up of  $n+1$  control point.
3. B-spline curve let us specify the order of basis ( $k$ ) function and the degree of the resulting curve is independent on the no. of vertices.
4. It is possible to change the degree of the resulting curve without changing the no. of control points.
5. B-spline can be used to define both open & close curves.
6. Curve generally follows the shape of defining polygon. If we have order  $k = 4$  then degree will be 3  $P(k) = x^3$ .
7. The curve line within the convex hull of its defining polygon.

In B-spline we segment out the whole curve which is decided by the order ( $k$ ). By formula ' $n-k+2$ '

### For example:

If we have 7 control points and order of curve  $k=3$  then  $n = 6$

and this B-spline curve has segments  $6 - 3 + 2 = 5$



Five segments  $Q_1, Q_2, Q_3, Q_4, Q_5$

Segment	Control points	Parameter
$Q_1$	$P_0 P_1 P_2$	$t_0 = 0, t_1 = 1$
$Q_2$	$P_1 P_2 P_3$	$t_1 = 1, t_2 = 2$
$Q_3$	$P_2 P_3 P_4$	$t_2 = 2, t_3 = 3$
$Q_4$	$P_3 P_4 P_5$	$t_3 = 3, t_4 = 4$
$Q_5$	$P_4 P_5 P_6$	$t_4 = 4, t_5 = 5$

There will be a join point or knot between  $Q_{i-1}$  &  $Q_i$  for  $i \geq 3$  at the parameter value  $t_i$  know as KNOT VALUE [X].

If  $P(u)$  be the position vectors along the curve as a function of the parameter  $u$ , a B-spline curve is given by

$$P(u) = \sum P_i N_{i,k}(u) \quad 0 \leq u \leq n-k+2$$

$N_{i,k}(u)$  is B-spline basis function

$$N_{i,k}(u) = +$$

The values of  $X_i$  are the elements of a knot vector satisfying the relation  $X_i \leq X_{i+1}$ .

The parameter  $u$  varies from 0 to  $n-k+2$  along the  $P(u)$ .

So there are some conditions for finding the KNOT VALUES [X]

$$X_i \quad (0 \leq i \leq n+k): \text{Knot values}$$

$$X_i = 0 \text{ if } i < k$$

$$X_i = i - k + 1 \text{ if } k \leq i \leq n$$

$$X_i = n - k + 2 \text{ if } i > n$$

So as B-spline curve has Recursive Equation. So we stop at

$$N_i, K(u) = 1 \text{ if } X_i \leq u < X_{i+1} \\ = 0 \text{ otherwise}$$

**Example:**

$$n = 5, k = 3$$

then  $X_i (0 \leq i \leq 8)$  knot values

$$X_i \{0, 0, 0, 1, 2, 3, 4, 4, 4, 4, \}$$

$$N_{0,3}(u) = (1-u)^2 \cdot N_{2,1}(u)$$

When  $i = 0, k = 3$  so  $i < k$  is true

$$X_0 = 0$$

$$i = 1, k = 3 \quad X_1 = 0$$

$$i = 2, k = 3 \quad X_2 = 0$$

$$i = 3, k = 3 \quad X_3 = i - k + 1 = 3 - 3 + 1$$

$$X_3 = 1$$

$$i = 4, k = 3 \quad X_4 = i - k + 1 = 4 - 3 + 1$$

$$X_4 = 2$$

$$i = 8, k = 3 \quad X_8 = i - n + k + 2 = 5 - 3 + 2 = 4$$

In this way we can calculate

- 
1. A parametric cubic curve passes through the point  $(0, 0), (2, 4), (4, 3)$  and  $(5, -2)$  which are parameterized at  $t = 0, \frac{1}{4}, \frac{3}{4}$  and  $1$  respectively. Determine the geometric coefficient matrix and the slope of the curve when  $t = 0.5$

**Solution:**

The points on the curve are

$$(0, 0) \text{ at } t = 0$$

$$(2, 4) \text{ at } t = 1/4$$

(4, 3) at  $t = 3/4$

(5, -2) at  $t = 1$

$$P(t) = [2t^3 - 3t^2 + 1]P(0) + [(-2t^3 + 3t^2)]P(1) + [(t^3 - 2t^2 + t)]P'(0) + [(t^3 - t^2)]P'(1)$$

In matrix form the equation can be written as.

$$\begin{bmatrix} P(t) \\ 2 \\ -2 \\ 1 \\ 1 \\ -3 \\ 3 \\ -2 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 2 & 4 & 4 & 3 & 5 \\ -2 & -2 \\ 1 & 1 \\ 1 & -3 \\ 3 & -2 \\ -1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 & 0.0156 & 0.0625 & 0.25 & 1 & 0.4218 & 0.5625 & 0.75 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 5 & -2 & 10.33 & 22 & 4.99 & -26 \end{bmatrix} \begin{bmatrix} P(0) \\ P(1) \\ P'(0) \\ P'(1) \end{bmatrix}$$

The geometric coefficients are:

$$p(0) = (0, 0)$$

$$p(1) = (5, -2)$$

$$p'(0) = (10.33, 22)$$

$$p'(1) = (4.99, -26)$$

The slope at  $t = 0.5$  is found by taking the first derivative of the above equation as follows.

$$p'(t) = [3t^2 \ 2t \ 1 \ 0]$$

$$\begin{bmatrix} 2 \\ -2 \\ 1 \\ 1 \\ -3 \\ 3 \\ -2 \\ -1 \\ 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} P'(0) \\ P'(1) \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 5 & -2 & 10.33 & 22 & 4.95 & -26 \end{bmatrix} \begin{bmatrix} P'(0) \\ P'(1) \end{bmatrix}$$

Therefore

$$P'(0.5) = [3.67 \ -2.0]$$

$$\text{Slope} = -0.545$$

2. Find equation of Bezier curve whose control points are  $P_0(2, 6)$ ,  $P_1(6, 8)$  and  $P_2(9, 12)$ . Also find co-ordinate of point at  $u = 0.8$ . [2071 Chaitra]

**Solution:**

Control points are  $P_0(2, 6)$ ,  $P_1(6, 8)$  and  $P_2(9, 12)$

Number of control points = 3

A Bezier curve is a polynomial of degree one less than the number of control points.

Degree of polynomial =  $3-1 = 2$

$u = 0.8$

$$P(u) = \sum_{k=0}^n p_k \quad BEZ_{k,n}(u)$$

$n = 2$ , so,

$$P(u) = \sum_{k=0}^2 p_k \quad BEZ_{k,2}(u)$$

$$BEZ_{k,2}(u) = C(2,k)u^k(1-u)^{2-k}$$

$$C(2,k) = \frac{2!}{k!(2-k)!}$$

$$P_x(u) = \sum_{k=0}^2 x_k BEZ_{k,2}(u)$$

$$BEZ_{0,2}(u) = C(2,0)u^0(1-u)^2 = (1-0.8)^2$$

$$BEZ_{1,2}(u) = C(2,1)u^1(1-u)^1 = 2 * (0.8)^1 * (1-0.8)^1$$

$$BEZ_{2,2}(u) = C(2,2)u^2(1-u)^0 = (0.8)^2 * (1-0.8)^0$$

$$\begin{aligned} P_x(u) &= x_0 BEZ_{0,2}(u) + x_1 BEZ_{1,2}(u) + x_2 BEZ_{2,2}(u) \\ &= 2 * (1-0.8)^2 + 6 * 2 * (0.8)^1 * (1-0.8)^1 + 9 * (0.8)^2 * (1-0.8)^0 \end{aligned}$$



$$= 7.76$$

$$\begin{aligned} P_y(u) &= y_0 \text{BEZ}_{0,2}(u) + y_1 \text{BEZ}_{1,2}(u) + y_2 \text{BEZ}_{2,2}(u) \\ &= 6 \times (1 - 0.8)^2 + 8 \times 2 \times (0.8)^1 \times (1 - 0.8)^1 + 12 \times \\ &\quad (0.8)^2 \times (1 - 0.8)^0 \\ &= 10.48 \end{aligned}$$

The coordinate point is (7.76, 10.48)

3. **Find the Bezier curve which passes through (0, 0, 0) and (-2, 1, 1) and is controlled by (7, 5, 2) and (2, 0, 1).**

[2076 Ashwin Back]

**Solution:**

In Bezier curve the starting and ending control points lie on the curve so (0, 0, 0) and (-2, 1, 1) are starting and ending control points and (7, 5, 2) and (2, 0, 1) are intermediate control points.

We know,

Position vector  $P(u) = P_k \text{BEZ}_{k,n}(u)$  for  $0 \leq u \leq 1$

Where,

$P_k$  is control point position

We have 4 control points so  $P_k$  varies  $k = 0$  to 3

The Bezier blending function  $\text{BEZ}_{k,n}(u)$  are the Bernstein polynomials,

$$\text{BEZ}_{k,n}(u) = C(n, k) u^k (1 - u)^{n-k}$$

where,  $C(n, k)$  are binomial coefficient

$$C(n, k) =$$

So,

$$P_x(u) = x_0 \text{BEZ}_{0,3}(u) + x_1 \text{BEZ}_{1,3}(u) + x_2 \text{BEZ}_{2,3}(u) + x_3 \text{BEZ}_{3,3}(u)$$

$$P_y(u) = y_0 \text{BEZ}_{0,3}(u) + y_1 \text{BEZ}_{1,3}(u) + y_2 \text{BEZ}_{2,3}(u) + y_3 \text{BEZ}_{3,3}(u)$$

$$P_z(u) = z_0 \text{BEZ}_{0,3}(u) + z_1 \text{BEZ}_{1,3}(u) + z_2 \text{BEZ}_{2,3}(u) + z_3 \text{BEZ}_{3,3}(u)$$

Where,

$$\text{BEZ}_{0,3}(u) = C(3,0)u^0 (1 - u)^3 = (1 - u)^3$$

$$\text{BEZ}_{1,3}(u) = C(3,1)u^1 (1 - u)^2 = 3u(1 - u)^2$$

$$\text{BEZ}_{2,3}(u) = C(3,2)u^2 (1 - u) = 3u^2(1 - u)$$

$$\text{BEZ}_{3,3}(u) = C(3,3)u^3(1-u)^0 = u^3$$

Now,

$$P_x(u) = x_0\text{BEZ}_{0,3}(u) + x_1\text{BEZ}_{1,3}(u) + x_2\text{BEZ}_{2,3}(u) + x_3\text{BEZ}_{3,3}(u)$$

$$= 0 \times (1-u) + 7 \times 3u(1-u)^2 + 2 \times 3u^2(1-u) + (-2) \times u^3$$

$$P_y(u) = y_0\text{BEZ}_{0,3}(u) + y_1\text{BEZ}_{1,3}(u) + y_2\text{BEZ}_{2,3}(u) + y_3\text{BEZ}_{3,3}(u)$$

$$= 0 \times (1-u) + 5 \times 3u(1-u)^2 + 0 \times 3u^2(1-u) + 1 \times u^3$$

$$P_z(u) = z_0\text{BEZ}_{0,3}(u) + z_1\text{BEZ}_{1,3}(u) + z_2\text{BEZ}_{2,3}(u) + z_3\text{BEZ}_{3,3}(u)$$

$$= 0 \times (1-u) + 2 \times 3u(1-u)^2 + 1 \times 3u^2(1-u) + 1 \times u^3$$

4. *A cubic Bezier curve is described by the four control points. (0,0) , (2,1) , (5,2) and (6,1) Find the tangent to the curve at t = 0.5*

**Solution:**

Here, we know the Bezier cubic polynomial equation.

$$P(t) = (1-t)^3P_0 + 3t(1-t)^2P_1 + 3t^2(1-t)P_2 + t^3P_3$$

The tangent is given by the derivative of the general equation above,

$$P'(t) = -3 \times (1-t)^2P_0 + 6t(1-t)P_1 - 6t^2P_1 + 3t(1-t)^2P_2 - 3t^2P_2 + 6t(1-t)P_3 + 3t^2P_3$$

$$x'(t) = -3 \times (1-t)^2x_0 - 6t(1-t)x_1 + 3(1-t)^2x_1 + 6t(1-t)x_2 - 3t^2x_2 + 3t^2x_3$$

$$= -3 \times (1-0.5)^2(0) - 6(0.5)(1-0.5)(2) + 3(1-0.5)^2(2) + 6(0.5)(1-0.5)(5) - 3(0.5)^2(5) + 3(0.5)^2(6)$$

$$= 6.75$$

$$y'(t) = 3 \times (1-t)^2y_0 + 6t(1-t)y_1 - 6t^2P_1 + 3t(1-t)^2y_1 - 3t^2y_2 + 6t(1-t)y_2 + 3t^2y_3$$

$$= 1.5$$

Or we can solve it by alternative method,

In matrix form this equation is written as

$$P(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \\ -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Where,

$$V_0 = (0,0)$$

$$V_1 = (2,1)$$

$$V_3 = (5,2)$$

$$V_4 = (6,1)$$

The tangent is given by the derivative of the general equation above,

$$P'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \\ -2 & 3 & -2 & 0 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

At  $t = 0.5$ , we get

$$P'(t) = \begin{bmatrix} 3(0.5)^2 & 2(0.5) & 1 & 0 \\ -2 & 3 & -2 & 0 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.75 & 1 & 1 & 0 \\ -2 & 3 & -2 & 0 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.75 & -0.75 & 0.75 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} -0.75 & -0.75 & 0.75 & 0.75 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$= [6.75 \quad 1.5]$$

$$= 1.5/6.75$$

$$= 0.222$$

$$\text{Tangent} = 0.222$$

$$\text{Tangent angle} = 12.53^\circ$$

5. *Design a Bezier curve controlled by points A(1,1), B(2, 3), C(4, 3) and D(6, 4)*

**Solution:**

Control point = 4

Degree of polynomial =  $n - 1 = 3$

$$A(1, 1) = p_0$$

$$B(2, 3) = p_1$$

$$C(4, 3) = p_2$$

$$D(6, 4) = p_3$$

Equation of Bezier curve

$$p(u) = p_k.B_{k,n}(u)$$

where  $p(u)$  = position vector

$p_k$  = control point

$$B_{k,n} = u^k(1 - u)^{n-k}$$

$$\begin{aligned} B_{0,3}(u) &= (3, 0) u^0 \cdot (1 - u)^{3-0} \\ &= (1 - u)^3 \\ &= (1 - u)^3 \end{aligned}$$

$$\begin{aligned} B_{1,3}(u) &= C(3,1) u^1 (1 - u)^{3-1} \\ &= 3u \cdot (1 - u)^2 \end{aligned}$$

$$\begin{aligned} B_{2,3}(u) &= C(3,2) \cdot u^2 (1 - u)^{3-2} \\ &= 3u^2 (1 - u) \end{aligned}$$

$$B_{3,3}(u) = C(3,3) u^3 (1 - u)^{3-3} = u^3$$

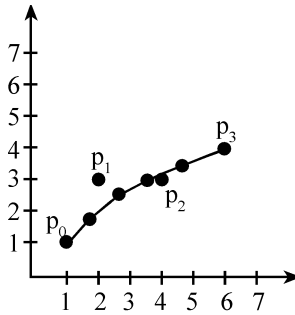
$$P(u) = p_0(1 - u)^3 + p_1 \cdot 3u(1 - u)^2 + p_2 3u^2(1 - u) + p_3 u^3$$

To find  $x(u)$  and  $y(u)$

$$x(u) = x_0(1 - u)^3 + x_1 \cdot 3u(1 - u)^2 + x_2 3u^2(1 - u) + x_3 u^3$$

$$y(u) = y_0(1 - u)^3 + y_1 \cdot 3u(1 - u)^2 + y_2 3u^2(1 - u) + y_3 u^3$$

u	x(u)	y(u)
0	1	1
0.2	1.712	1.984
0.4	2.616	2.632
0.6	3.664	3.088
0.8	4.808	3.496
1	6	4



6. Write the equation of Bezier curve with four control points  $p_1 (2,2,0)$ ,  $p_2 (2,3,0)$ ,  $p_3 (3,3,0)$ , and  $p_4 (3,2,0)$ . Find the coordinate pixel of the curve for  $u = 0, 1/4, 1/2, 3/4$ .

[2078 Chaitra]

**Solution:**

Here, given,

Number of points i.e.  $k = 4$ ,

Hence, we know that the degree of the Bezier curve is

$$n = k - 1 = 4 - 1 = 3$$

Now, to obtain the equation of the Bezier curve in parametric format with parameter 'u', we know that

$$P(u) = P_i B_{i,n}(u)$$

where,  $n = 3$  and  $P_i$  is the  $i^{\text{th}}$  control and  $B_{i,n}$  is defined as,

$$B_{i,n} = {}^nC_r \times u^i (1-u)^{n-i}$$

Hence,

$$P_0(2, 2, 0) \text{ and } B_{0,3} = (1-u)^3,$$

$$P_1(2, 3, 0) \text{ and } B_{1,3} = 3u(1-u)^2,$$

$$P_2(3, 3, 0) \text{ and } B_{2,3} = 3u^2(1-u) \text{ and}$$

$$P_3(3, 2, 0) \text{ and } B_{3,3} = u^3$$

Thus,

$$P(u) = P_0 \times B_{0,3}(u) + P_1 \times B_{1,3}(u) + P_2 \times B_{2,3}(u) + P_3 \times B_{3,3}(u)$$

$$P(u) = P_0 \times (1-u)^3 + P_1 \times 3u(1-u)^2 + P_2 \times 3u^2(1-u) + P_3 \times u^3$$

which is the equation of the Bezier curve.

It can be further be simplified and written in matrix form as,

$$P(u) = [u^3 \ u^2 \ u \ 1]$$

$$P(u) = [u^3 \ u^2 \ u \ 1]$$

$$P(u) = [u^3 \ u^2 \ u \ 1]$$

Now, to obtain the points on the curve for  $u=0, ,$  and  $1$

$$P(u = 0) = [0 \ 0 \ 0 \ 1] = [2 \ 2 \ 0]$$

$$P(u = 1/4) = [1/64 \ 1/16 \ 1/4 \ 1] \\ = [71/32 \ 21/8 \ 0]$$

$$P(u = 1/2) = [1/8 \ 1/4 \ 1/2 \ 1] = [3 \ 13/4 \ 0]$$

$$P(u = 3/4) = [27/64 \ 9/16 \ 3/4 \ 1] \\ = [145/32 \ 17/4 \ 0]$$

$$P(u = 1) = [1 \ 1 \ 1 \ 1] = [7 \ 6 \ 0]$$

The coordinate pixel of the curve for  $u = 0, 1/4, 1/2, 3/4$  are  $(2 \ 2 \ 0), (71/32 \ 21/8 \ 0), (3 \ 13/4 \ 0),$  and  $(145/32 \ 17/4 \ 0)$  respectively.

7. ***Find the coordinates of Bezier curve at  $u = 0.25, 0.5$  and  $0.75$  with respect to the control points  $(10, 15), (15, 20), (20, 35), (25, 10)$  using Bezier function. [2079 Baishakh]***

***Solution:***

Here, given,

Total number of control point = 4

Then degree of polynomial =  $n - 1 = 3$

$$A(10, 15) = p_0$$

$$B(15, 20) = p_1$$

$$C(20, 35) = p_2$$

$$D(25, 10) = p_3$$

Equation of Bezier curve

$$p(u) = p_k B_{k,n}(u)$$

where  $p(u)$  = position vector

$p_k$  = control point

$$B_{k,n} = u^k (1 - u)^{n-k}$$

$$B_{0,3}(u) = (3, 0) u^0 \cdot (1 - u)^{3-0} \\ = (1 - u)^3$$

$$\begin{aligned}
&= (1 - u)^3 \\
B_{1,3}(u) &= C(3,1) u^1 (1 - u)^{3-1} \\
&= 3u(1 - u)^2 \\
B_{2,3}(u) &= C(3,2) \cdot u^2 (1 - u)^{3-2} \\
&= 3u^2(1 - u) \\
B_{3,3}(u) &= C(3,3) u^3 (1 - u)^{3-3} = u^3 \\
P(u) &= p_0(1 - u)^3 + p_1 \cdot 3u(1 - u)^2 + p_2 3u^2(1 - u) + p_3 u^3 \\
\text{To find } x(u) \text{ and } y(u) \\
X(u) &= x_0(1 - u)^3 + x_1 \cdot 3u(1 - u)^2 + x_2 3u^2(1 - u) + x_3 u^3 \\
y(u) &= y_0(1 - u)^3 + y_1 \cdot 3u(1 - u)^2 + y_2 3u^2(1 - u) + y_3 u^3
\end{aligned}$$

u	x(u)	y(u)
0	10	15
0.25	13.75	19.84375
0.5	17.5	23.75
0.75	21.25	22.03125
1	25	10

The coordinates of Bezier curve at  $u = 0.25, 0.5$  and  $0.75$  are  $(13.75, 19.84375)$ ,  $(17.5, 23.75)$  and  $(21.25, 22.03125)$  respectively.

## 8. Derive the blending function for parametric cubic curve

To derive the blending functions for a parametric cubic curve, we need to start with the general form of a cubic polynomial. In parametric form, a cubic curve can be expressed as:

$$P(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$$

Here,  $P(t)$  represents a point on the curve at parameter  $t$ , and  $a_0, a_1, a_2, a_3$  are vector coefficients that define the shape of the curve. These coefficients will be determined based on the given control points and curve properties.

### Blending Functions for Cubic Bezier Curve

A common representation of a parametric cubic curve is the cubic Bezier curve, defined by four control points  $P_0, P_1, P_2, P_3$ . The parametric equation for a cubic Bezier curve is:

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$

This can be expanded to:

$$P(t) = P_0(1-3t+3t^2-t^3) + P_1(3t-6t^2+3t^3) + P_2(3t^2-3t^3) + P_3 t^3$$

Here, the blending functions  $B_i(t)$  are the coefficients of the control points  $P_i$ :

$$B_0(t) = (1-t)^3$$

$$B_1(t) = 3t(1-t)^2$$

$$B_2(t) = 3t^2(1-t)$$

$$B_3(t) = t^3$$

These blending functions have the property that they sum to 1 for any  $t$  in the interval  $[0,1]$ .

### Blending Functions for Hermite Curve

Another common representation is the cubic Hermite curve, defined by two endpoints  $P_0$  and  $P_1$  and the tangents at these endpoints  $R_0$  and  $R_1$ . The parametric equation for a cubic Hermite curve is:

$$P(t) = (2t^3 - 3t^2 + 1)P_0 + (t^3 - 2t^2 + t)R_0 + (-2t^3 + 3t^2)P_1 + (t^3 - t^2)R_1$$

Here, the blending functions  $H_i(t)$  are the coefficients of the control points and tangents:

$$H_0(t) = 2t^3 - 3t^2 + 1$$

$$H_1(t) = -2t^3 + 3t^2$$

$$H_2(t) = t^3 - 2t^2 + t$$

$$H_3(t) = t^3 - t^2$$

These functions are designed so that  $H_0(t)$  and  $H_1(t)$  blend the endpoints  $P_0$  and  $P_1$ , and  $H_2(t)$  and  $H_3(t)$  blend the tangents  $R_0$  and  $R_1$ .

### General Parametric Cubic Curve

For a general parametric cubic curve expressed with four control points  $P_0, P_1, P_2, P_3$  we can write the equation as:



$$P(t)=a_0+a_1t+a_2t^2+a_3t^3$$

To express this in terms of the control points, we solve for the coefficients  $a_0$ ,  $a_1$ ,  $a_2$ ,  $a_3$  using the conditions provided by the control points and the parametric form. This typically involves setting up and solving a system of equations based on the curve's specific form (Bezier, Hermite, B-spline, etc.).

### Summary

The blending functions are the polynomial coefficients that determine how the control points influence the final curve. For a cubic Bezier curve, these functions are:

$$B_0(t)=(1-t)^3$$

$$B_1(t)=3t(1-t)^2$$

$$B_2(t)=3t^2(1-t)$$

$$B_3(t)=t^3$$

For a cubic Hermite curve, the blending functions are:

$$H_0(t)=2t^3-3t^2+1$$

$$H_1(t)=-2t^3+3t^2$$

$$H_2(t)=t^3-2t^2+t$$

$$H_3(t)=t^3-t^2$$

These functions ensure the curve passes through the control points and interpolates the tangents (for Hermite) or smoothly blends between control points (for Bezier).

---



# Surface Modeling

---

## 6.1 Three-Dimensional Object Representations

---

Graphics scenes can contain many different kinds of objects: trees, flowers, clouds, rocks, water, bricks, rubber, paper, marble, steel, glass, plastic, and so on. So, there is no single method that can be used to describe objects that will include all characteristics of these different materials. To provide realistic displays of scenes, we need to use representations that accurately model object characteristics. The representation of complex objects such as aircraft bodies, casting and dies, automotive bodies can not be achieved by wireframe modeling. For this, another geometric modeling called *surface modeling* is necessary.

Polygon and quadric surfaces provide precise descriptions for simple objects like polyhedrons and ellipsoids. Spline surfaces are useful for designing aircraft wings, gears, and other engineering structure with curved surfaces. Procedural methods such as fractal construction and particle systems are useful for accurately representing clouds, clumps of grass, and other natural objects. Octree encodings are used to represent internal features of object, such as those obtained from medical CT images.

Representation schemes for solid objects are divided into two broad categories: Boundary representations and space-partitioning representations; although not all representations fall neatly into one or other these categories.

### 1) Boundary representations

Boundary representation method is used to describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. E.g., polygon surfaces, curved surfaces.

### 2) Space-partitioning representations

Space partitioning representation method is used to describe interior properties by partitioning the region containing an object into a set of small, nonoverlapping contiguous solids (usually cubes). E.g., octree representation

## **6.2 Polygon Surfaces**

The most commonly used boundary representation for a three-dimensional object is a set of surface polygons that enclose the object interior. Many graphics systems store all object descriptions as sets of surface polygons. This makes surface rendering and display of objects simple and fast because all surfaces are described with linear equations. Almost all graphics packages provide this type of object representations. This is why polygon description is referred to as “standard graphics objects”.

Polygon surface representation of an object is created by dividing the boundary surface into a number of interconnected polygons. But for some objects, surfaces are simply tiled to produce the polygon mesh approximation. Such representations are common in design and modeling applications, since the wireframe outline can be displayed quickly to give a general indication of the surface structure. Realistic renderings are produced by interpolating shading patterns across the polygon surfaces to eliminate or reduce the presence of polygon edge boundaries.

## **6.3 Polygon Table**

A polygon surface is specified with a set of vertex coordinates and associated attribute parameters. Information of each polygon is stored in polygon data tables that are used to process, display, and manipulate the objects in a scene.

Polygon data tables store the coordinate description and parameters that specify the spatial orientation of polygon surfaces as well as the attribute parameters that specify the surface characteristics such as surface reflectivity, degree of transparency, surface texture, etc.

Polygon data tables are organized into two groups: geometric tables and attribute tables.

1. **Geometric table**

This table contains vertex coordinates and parameters that specify the spatial orientation of the polygon surfaces.

Geometric data tables are usually organized into three lists:

i. **Vertex table**

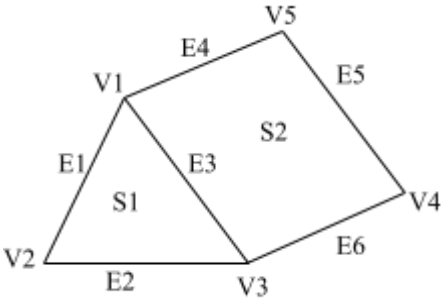
It stores coordinate values for each vertex in the object.

ii. **Edge table**

It contains pointers back into the vertex table to identify the vertices for each polygon edge.

iii. **Polygon-surface table**

It contains pointers back into the edge table to identify the edges for each polygon.



Vertex Table	Edge Table	Polygon- Surface Table
V <sub>1</sub> :x <sub>1</sub> ,y <sub>1</sub> ,z <sub>1</sub>	E <sub>1</sub> :V <sub>1</sub> ,V <sub>2</sub>	S <sub>1</sub> :E <sub>1</sub> ,E <sub>2</sub> ,E <sub>3</sub>
V <sub>2</sub> : x <sub>2</sub> ,y <sub>2</sub> ,z <sub>2</sub>	E <sub>2</sub> :V <sub>2</sub> ,V <sub>3</sub>	S <sub>2</sub> :E <sub>3</sub> ,E <sub>4</sub> ,E <sub>5</sub> ,E <sub>6</sub>
V <sub>3</sub> :x <sub>3</sub> ,y <sub>3</sub> ,z <sub>3</sub>	E <sub>3</sub> :V <sub>3</sub> ,V <sub>1</sub>	
V <sub>4</sub> :x <sub>4</sub> ,y <sub>4</sub> ,z <sub>4</sub>	E <sub>4</sub> :V <sub>3</sub> ,V <sub>4</sub>	
V <sub>5</sub> :x <sub>5</sub> ,y <sub>5</sub> ,z <sub>5</sub>	E <sub>5</sub> :V <sub>4</sub> ,V <sub>5</sub>	
	E <sub>6</sub> :V <sub>5</sub> ,V <sub>1</sub>	

**Figure 6.1:** Geometric data table representation for two adjacent polygon surfaces formed with six edges and five vertices.

Listing the geometric data in three tables provides a convenient reference to the individual components (vertices, edges, and polygons) of each object.

With the help of these three tables we can model conical or cylindrical objects like dharahara. We divide objects's shape into facets and represent them in polygon tables.

**2) Attribute table**

This table contains attribute information that includes parameters that specify the degree of transparency of the object, its surface reflectivity, and texture characteristics.

**Some tests (guidelines) to generate error free table:**

- i) Check if every vertex is listed as an endpoint for at least two edges.
- ii) Check is every edge is part of at least one polygon.
- iii) Check if each polygon is closed.
- iv) Check if each polygon has at least one shared edge and
- v) Check if the edge table contains pointers to polygons, every edge referenced by a polygon pointer has a reciprocal pointer back to be the polygon.

**6.4 Plane Equations**

To produce a display of three-dimensional object, we must process the input data representation for the object through several procedures. These processing steps include:

- i. Transformation of the modeling and world coordinate descriptions to viewing coordinates, then to device coordinates.
- ii. Identification of visible surfaces, and
- iii. The application of surface rendering procedures.

For some of these processes, we need information about the spatial orientation of the individual surface components of the object.

This information is obtained from the vertex co-ordinate values and the equations that describe the polygon planes.

Equation for a plane surface can be expressed as

$Ax + By + Cz + D = 0..... (i)$

Where  $(x, y, z)$  is a point on the plane, and coefficients  $A, B, C$ , and  $D$  are constant that describe the spatial properties of the plane.

We can obtain the values for  $A, B, C$  and  $D$  by solving a set of three plane equations using the coordinate values for three non-collinear points. We select 3 successive polygon vertices  $(x_1, y_1, z_1)$ ,  $(x_2, y_2, z_2)$  and  $(x_3, y_3, z_3)$  and solve the following set of simultaneous linear plane equations for the ratios  $\frac{A}{D}$ ,  $\frac{B}{D}$ , and  $\frac{C}{D}$ .

$$\frac{A}{D}x_1 + \frac{B}{D}y_1 + \frac{C}{D}z_1 = -1$$

$$\frac{A}{D}x_2 + \frac{B}{D}y_2 + \frac{C}{D}z_2 = -1$$

$$\frac{A}{D}x_3 + \frac{B}{D}y_3 + \frac{C}{D}z_3 = -1$$

Using Cramer's rule, we get

$$A = \frac{\begin{vmatrix} y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}}{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}} \quad B = \frac{\begin{vmatrix} x_1 & x_2 & x_3 \\ z_1 & z_2 & z_3 \end{vmatrix}}{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}}$$

$$C = \frac{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}}{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}} \quad D = \frac{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{vmatrix}}{\begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{vmatrix}}$$

After expanding the determinants, we get,

$$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$$

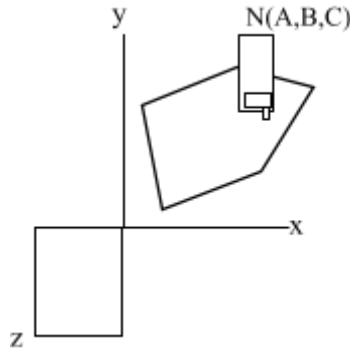
$$B = x_1(x_2 - x_3) + x_2(x_3 - x_1) + x_3(x_1 - x_2)$$

$$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$$

$$D = -x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$$

As vertex values and other information are entered into the polygon structure, values for  $A, B, C, D$  are computed for each polygon and stored with the other polygon data.

Orientation of a plane surface in space can be described with the normal vector to the plane. The normal vector has Cartesian components  $(A, B, C)$  where  $A, B, C$  are the plane coefficients calculated above.

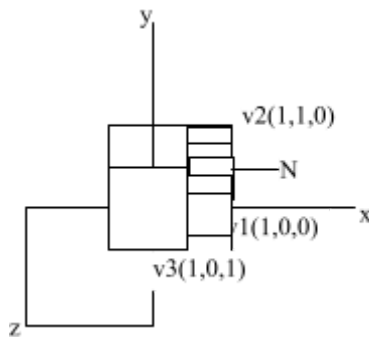


**Figure 6.2:** The vector  $N$ , normal to the surface of a plane described by the equation  $Ax + By + Cz + D = 0$ , has Cartesian components  $(A, B, C)$

Usually we deal with polygon surface that encloses the object interior. In that case, we need to distinguish between the "inside" and "outside" faces.

If polygon vertices are specified in a counter clockwise direction when viewing from the outer side of the plane, in a right-handed coordinate system, the direction of the normal vector will be from inside to outside.

To determine the components of the normal vector for the shaded surface, we select three of the four vertices along the boundary of the polygon, while viewing from outside in counter clockwise direction.



**Figure 6.3:** The shaded polygon surface of the unit cube has plane equation  $x - 1 = 0$  and normal vector  $N = (1, 0, 0)$

Now using the co-ordinate values for these vertices, we solve the equation for A, B, C and D are given.

$$A = 1, B = 0, C = 0, D = -1$$

This shows the normal vector is in positive x-direction. we can obtain the element of normal vector by calculating the vector cross product.

We again select any three vertices in counter clockwise direction while viewing from outside and calculate the normal vector as

$$N = (V_2 - V_1) \times (V_3 - V_1)$$

This will give us the direction of N.

For points not on the polygon surface,

$$Ax + By + Cz + D = 0$$

So,

If  $Ax + By + Cz + D < 0$ , the point (x, y, z) is inside the surface.

If  $Ax + By + Cz + D > 0$ , the points (x, y, z) is outside the surface.

$$N = (V_2 - V_1) \times (V_3 - V_1)$$

Plane can also be represented as  $N \cdot P = -D$ , where P is a position vector of any given point on the plane.

## Polygon meshes

A polygon mesh is a type of computer graphics technique used for creating 3D models. It is a collection of vertices, edges and faces that define the shape and surface of a 3D object.

A polygon mesh is a mathematical representation of a 3D object. It is composed of a set of vertices, edges, and faces that define the shape of the object. A vertex is a single point in 3D space, an edge is a line connecting two vertices, and a face is a closed loop of edges. The faces are usually triangles or



quadrilaterals, and can be used to represent curved surfaces or other complex shapes.

Some graphics packages provide several polygon functions for modeling objects. But when object surfaces are to be tiled, it is more convenient to specify the surface facets with a mesh functions.

A polygon mesh is a sequence of connected polygon. The components of polygon mesh are vertices, edges and polygons that define the shape of polyhedral object. The polygons are connected in such a manner that at least two polygon share an edge and hence bounded the planner surface. Vertices are points in three dimension that stores x, y, and z components. Edge joins two vertices. Faces are closed set of edges. Polygons are set of faces. Surface means group of connected polygons. Surface defines the different element of the object.

### **Representation ways of Polygon Mesh**

Polygon Mesh method basically used to represent a broad class of surfaces or solids in computer graphics. It is possible to render a polygonal mesh by applying the hidden surface removal methods as per requirement. So, the polygon mesh can be represented by **three ways** which are:

- Explicit representation
- Pointers to a vertex list
- Pointers to an edge list

### **Uses of Polygon Mesh**

- Polygon mesh has a wide range of applications in computer graphics. It can be used to represent a variety of shapes, from simple cubes to complex characters. The mesh can be manipulated to create realistic images, animations, and simulations.
- Polygon mesh can be used to create 3D models for games and films. The mesh can be modified to create realistic characters and environments. It is also used in virtual

reality applications, as it allows users to interact with a 3D environment.

- Polygon mesh can also be used to create 3D printing models. The mesh can be manipulated to create intricate designs for 3D printing.

### **Advantages**

- Polygon mesh has several advantages over other 3D modeling techniques. It is relatively easy to use and understand. It is also highly efficient, as it requires fewer polygons to create a 3D object than other techniques.
- Polygon mesh is also versatile. It can be used to create a wide range of shapes, from simple cubes to complex characters. It can also be used to create 3D printing models, as well as realistic animations and simulations.
- Polygon mesh is highly scalable. It can be used to create 3D models of any size, from tiny objects to large structures.

### **Disadvantages**

- In Polygon Mesh method, simulating some types of objects, such as liquid or hair, is very much challenging.
- The description phase of Polygon Mesh method, it is much difficult to simulate some type of objects like liquid or hair.

### **Applications**

- It is used in computer games and animation. Due to its lightweight nature, it is well-suited for real-time applications. It is also used in virtual reality, where it is used to create virtual environments.
- It is used in CAD applications. It is used to create 3D models for engineering and manufacturing projects.
- It is used in medical imaging. It is used to create 3D models of the human body for medical research and surgery planning.

## Conclusion

Polygon mesh is a versatile technique for creating 3D models. It is composed of vertices, edges, and faces that define the shape of an object. It is lightweight, efficient, and widely supported, making it ideal for real-time applications such as computer games and animation. It has some drawbacks, such as its limited ability to represent curved surfaces, but these can be overcome with careful modeling. Polygon mesh is used in a variety of applications, including computer games, animation, virtual reality, CAD, and medical imaging.

## How a Polygon Mesh Represents a Surface

### 1. Surface Approximation:

- The mesh approximates a continuous surface by dividing it into small, flat polygonal faces.
- The more vertices and faces in the mesh, the closer the approximation to the actual surface.

### 2. Connectivity:

- The vertices, edges, and faces are connected in a specific way to form a coherent surface.
- The connectivity information is crucial for defining the topology of the mesh.

### 3. Manifold and Non-Manifold Surfaces:

- A manifold mesh has a well-defined surface with no ambiguities; each edge belongs to exactly two faces (except at boundaries).
- Non-manifold meshes have edges or vertices shared by more than two faces, which can create complex surfaces.

## Example: A Simple Triangle Mesh

Consider a mesh composed of two triangles:

- **Vertices:**  $V1=(0,0,0)$ ,  $V2=(1,0,0)$ ,  $V3=(0,1,0)$ ,  $V4=(1,1,0)$
- **Edges:**  $E1=(V1,V2)$ ,  $E2=(V2,V4)$ ,  $E3=(V4,V3)$ ,  
 $E4=(V3,V1)$ ,  $E5=(V2,V3)$
- **Faces:**
  - F1 with vertices  $V1,V2,V3$
  - F2 with vertices  $V2,V4,V3$

This simple configuration approximates a rectangular surface by dividing it into two triangles.

### Representation in Data Structures

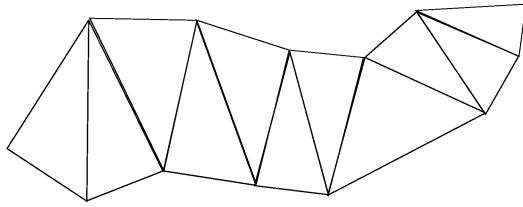
A polygon mesh can be represented in various data structures, such as:

1. **Vertex List:** A list of all vertices with their coordinates.
2. Example:  $[(0,0,0),(1,0,0),(0,1,0),(1,1,0)]$
3. **Face List:** A list of faces, where each face is represented by indices of its vertices.
4. Example:  $[(0,1,2),(1,3,2)]$
5. **Edge List (Optional):** Sometimes, an explicit list of edges is maintained for easier edge manipulation.
  - Example:  $[(0,1),(1,3),(3,2),(2,0),(1,2)]$

### Common types of polygon mesh

#### i. Triangular mesh

With  $n$  vertices, produce  $n-2$  triangles



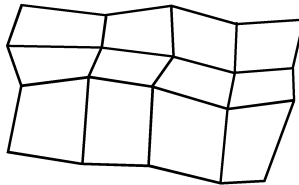
**Figure 6.4:** A triangle strip formed with 11 triangles 13 vertices

**ii. Quadrilateral mesh**

$$m = 5$$

$$n = 4$$

$$\text{Quadrilaterals} = (m-1)(n-1) = 12$$

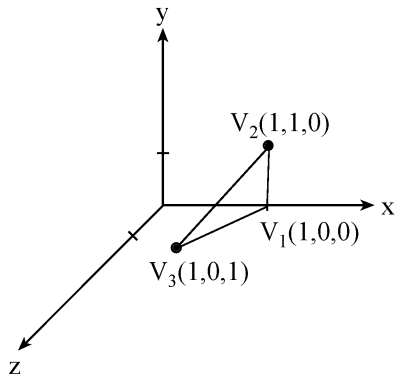


**Figure 6.3:** A quadrilateral mesh containing 12 quadrilaterals constructed from a 5 by 4 input vertex array

- 
- 1. If three vertices of a polygon surfaces in anticlockwise direction are  $V_1(1,0,0)$ ,  $V_2(1, 1, 0)$  and  $V_3(1, 0, 1)$ . Calculate normal vector of that surface. [2078 Chaitra]**

**Solution:**

The vertices of a polygon surfaces in anticlockwise direction are  $V_1(1,0,0)$ ,  $V_2(1,1,0)$  and  $V_3(1,0,1)$ .



To calculate the normal vector, we select the three vectors in anticlockwise direction.

$$\begin{aligned}
 \mathbf{N} &= (\mathbf{V}_2 - \mathbf{V}_1) \times (\mathbf{V}_3 - \mathbf{V}_1) \text{ (since } \mathbf{j} \times \mathbf{k} = \mathbf{i} \text{)} \\
 &= ((1-1)\mathbf{i} + (1-0)\mathbf{j} + (0-0)\mathbf{k}) \times ((1-1)\mathbf{i} + (0-0)\mathbf{j} + (1-0)\mathbf{k}) \\
 &= (\mathbf{j}) \times (\mathbf{k}) \\
 &= \mathbf{i}
 \end{aligned}$$

The normal vector of that surface =  $\mathbf{i}$

---



---

# Visible Surface Determination

---

## 7.1 Visible Surface Determination (Hidden Surface Elimination)

---

Visible surface determination is a process of identifying those parts of a scene that are visible from a chosen viewing position. There are numerous algorithms: some require more memory, some involve more processing time, and some apply only to special types of objects. The choice of a particular algorithm depends on factors like the complexity of the scene, type of objects to be displayed, available equipment, and whether static or animated displays are to be generated.

Visible-surface detection algorithms are broadly classified into two categories:

**i. Object-space method**

This method deals with object definitions directly. It compares objects and parts of objects to each other within the scene definition to determine which surfaces, as a whole, we should label as visible.

**ii. Image-space method**

This method deals with the projected images of the objects. In this method, visibility is decided point by point at each pixel position on the projection plane. Most visible-surface algorithms use image-space methods.

## 7.2 Back-Face Detection

---

It is a fast and simple object-space method for identifying the backfaces of a polyhedron (a solid in three dimensions with flat polygonal faces, straight edges and sharp corners or vertices) and is based on the “inside-outside” tests.

A point  $(x, y, z)$  is “inside” a polygon surface with plane parameters  $A, B, C$ , and  $D$  if

$$Ax + By + Cz + D < 0$$

When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

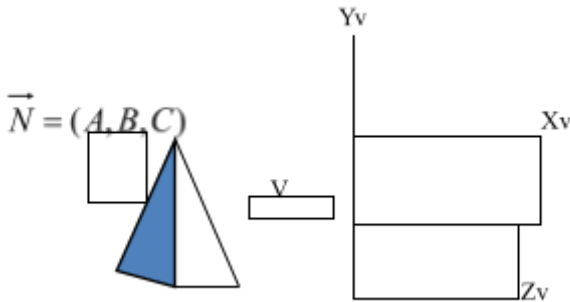
To simplify this test, consider the normal vector  $N$  to a polygon surface. If  $V$  is a vector in the viewing direction from the eye (or camera) position, then this polygon is a back face if

$$V \cdot N > 0$$

If object description have been converted to projection coordinates and our viewing direction is parallel to the viewing  $z_v$  axis, then  $V = (0, 0, V_z)$  and

$$V \cdot N = V_z C$$

So that we only need to consider the sign of  $C$ , the  $z$  component of the normal vector  $N$ .

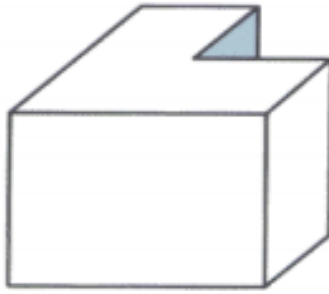


**Figure 7.1:** A polygon surface with plane parameter  $C < 0$  in a right-handed viewing coordinate system is identified as a back face when the viewing direction is along the negative  $z_v$  axis.

In a right-handed viewing system with viewing direction along the negative  $z_v$  axis, the polygon is a backface if  $C < 0$ . Also, we cannot see any face whose normal has  $z$  component  $C = 0$ , since our viewing direction is grazing that polygon. Thus, in general, we can label any polygon as a backface if its normal vector has a  $z$  component value

$$C \leq 0$$

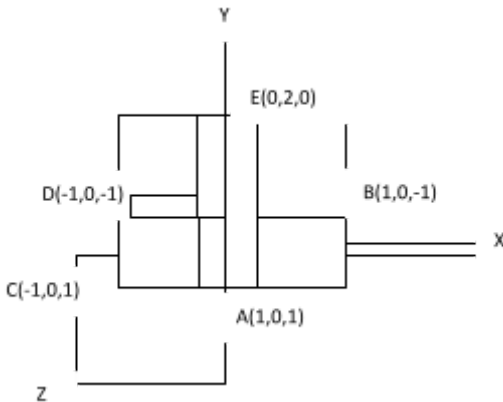




**Figure 7.2:** View of concave polyhedron with one face partially hidden by other faces.

It is not useful for concave polyhedron or overlapping surface. It does not deal whether the object is transparent or opaque.

**Q.** Find the visibility for the surfaces  $BED$ ,  $ABCD$ ,  $ABE$ , and  $ACE$  where observer at  $P(5,5,5)$



For surface  $BED$ ,

Normal vector of the surface  $BED$ ,  $N_{BED} = \vec{BD} \times \vec{BE}$   
(since  $-\vec{i} \times \vec{j} = -\vec{k}$ )

$$=((-1-1)\vec{i} + (0-0)\vec{j} + (-1+1)\vec{k}) \times ((0-1)\vec{i} + (2-0)\vec{j} + (0-1)\vec{k})$$

$$\begin{aligned}
 &= (-2i) \times (-i + 2j + k) \\
 &= 2j - 4k
 \end{aligned}$$

Now,

$$VPE = -5i - 3j - 5k$$

$$VPB = -4i - 5j - 6k$$

$$VPD = -6i - 5j - 6k$$

Now, For Visibility test for vertex E,

$$\begin{aligned}
 VPE \cdot NBED &= (-5i - 3j - 5k) \cdot (2j - 4k) \\
 &= -6 + 20 \\
 &= 14 > 0
 \end{aligned}$$

So, the vertex E is invisible

For Visibility test for vertex B,

$$\begin{aligned}
 VPB \cdot NBED &= (-4i - 5j - 6k) \times (2j - 4k) \\
 &= -10 + 24 \\
 &= 14 > 0
 \end{aligned}$$

So, the vertex B is invisible

For Visibility test for vertex D,

$$\begin{aligned}
 VPD \cdot NBED &= (-6i - 5j - 6k) \times (2j - 4k) \\
 &= -10 + 24 \\
 &= 14 > 0
 \end{aligned}$$

So, the vertex D is invisible

For surface ABCD,

$$\begin{aligned}
 \text{Normal vector of the surface ABCD, } N_{ABCD} &= BD \times BA \\
 &= (-2i) \times (-2k) \\
 &= -4j
 \end{aligned}$$

Now,

$$\text{VPB} = -4i - 5j - 6k$$

$$\text{VPD} = -6i - 5j - 6k$$

$$\text{VPA} = -4i - 5j - 4k$$

$$\text{VPC} = -6i - 5j - 4k$$

For Visibility test for vertex B,

$$\text{VPB} \cdot \text{NACBD} = (-4i - 5j - 6k) \times (-4j)$$

$$= 20$$

$$= 20 > 0$$

So, the vertex B is invisible

For Visibility test for vertex D,

$$\text{VPD} \cdot \text{NACBD} = (-6i - 5j - 6k) \times (-4j)$$

$$= 20$$

$$= 20 > 0$$

So, the vertex D is invisible

Now, For Visibility test for vertex A,

$$\text{VPA} \cdot \text{NBED} = (-4i - 5j - 4k) \times (-4j)$$

$$= 20$$

$$= 20 > 0$$

So, the vertex A is invisible

Now, For Visibility test for vertex C,

$$\text{VPC} \cdot \text{NBED} = (-6i - 5j - 4k) \times (-4j)$$

$$= 20$$

$$= 20 > 0$$

So, the vertex C is invisible

For surface ABE,

Normal vector of the surface ABE,  $\text{NABE} = \text{AB} \times \text{AE}$

$$\begin{aligned}
 &= (-2k) \times (-i + 2j - k) \\
 &= 4i + 2j
 \end{aligned}$$

Now,

$$VPE = -5i - 3j - 5k$$

$$VPA = -4i - 5j - 4k$$

$$VPB = -4i - 5j - 6k$$

For Visibility test for vertex E,

$$\begin{aligned}
 VPE \cdot NABE &= (-5i - 3j - 5k) \times (4i + 2j) \\
 &= -20 - 6 \\
 &= -26 < 0
 \end{aligned}$$

So, the vertex E is visible

For Visibility test for vertex B,

$$\begin{aligned}
 VPB \cdot NABE &= (-4i - 5j - 6k) \times (4i + 2j) \\
 &= -16 - 20 \\
 &= -36 < 0
 \end{aligned}$$

So, the vertex B is visible

Now, For Visibility test for vertex A,

$$\begin{aligned}
 VPA \cdot NABE &= (-4i - 5j - 4k) \times (4i + 2j) \\
 &= -16 - 20 \\
 &= -36 < 0
 \end{aligned}$$

So, the vertex A is visible

For surface ACE,

$$\begin{aligned}
 \text{Normal vector of the surface ACE, } NACE &= AE \times AC \\
 &= (-i + 2j - k) \times (-2i) \\
 &= 2j + 4k
 \end{aligned}$$

Now,

$$\text{VPA} = -4\mathbf{i} - 5\mathbf{j} - 4\mathbf{k}$$

$$\text{VPC} = -6\mathbf{i} - 5\mathbf{j} - 4\mathbf{k}$$

$$\text{VPE} = -5\mathbf{i} - 3\mathbf{j} - 5\mathbf{k}$$

For Visibility test for vertex A,

$$\text{VPA} \cdot \text{NACE} = (-4\mathbf{i} - 5\mathbf{j} - 4\mathbf{k}) \times (3\mathbf{j} + 4\mathbf{k})$$

$$= -15 - 16$$

$$= -31 < 0$$

So, the vertex A is visible

For Visibility test for vertex C,

$$\text{VPC} \cdot \text{NACE} = (-6\mathbf{i} - 5\mathbf{j} - 4\mathbf{k}) \times (3\mathbf{j} + 4\mathbf{k})$$

$$= -15 - 16$$

$$= -31 < 0$$

So, the vertex C is visible

For Visibility test for vertex E,

$$\text{VPE} \cdot \text{NACE} = (-5\mathbf{i} - 3\mathbf{j} - 5\mathbf{k}) \times (3\mathbf{j} + 4\mathbf{k})$$

$$= -9 - 20$$

$$= -29 < 0$$

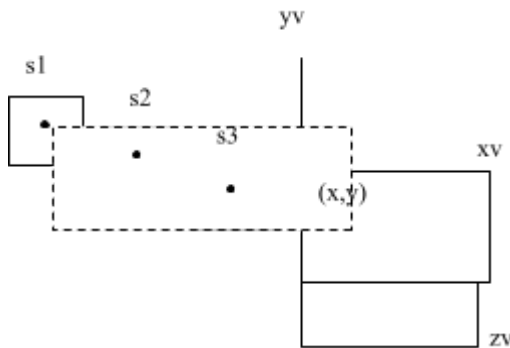
So, the vertex E is visible

### 7.3 Depth-Buffer Method (Z-Buffer)

Depth-buffer method is a commonly used image-space method for detecting visible surfaces. This method compares surface depth at each pixel position on the projection plane. It deals with overlapping surface but does not deal with transparent surface.

This procedure is also called z-buffer method, as object depth is usually measured from the view plane along the z axis of a viewing system. Each surface of a scene is processed separately, one point at a time across the surface. This method is suitable for scenes containing only polygon surfaces.

With object descriptions converted to projection coordinates, each  $(x, y, z)$  position on a polygon surface corresponds to the orthographic projection point  $(x, y)$  on the view plane. For each pixel position  $(x, y)$  on the view plane, object depths can be compared by comparing  $z$  values. Along the projection line from the position  $(x, y)$  in view plane taken as  $x_v y_v$  plane. Surface  $S_1$  is closest at this position, so its surface intensity value at  $(x, y)$  is saved.



**Figure 7.2:** At view-plane position  $(x, y)$ , surface  $S_1$ , has the smallest depth from the view plane and so is visible at that position

Depth-buffer method requires two buffer areas:

- i. *Depth buffer* that stores depth values for each  $(x, y)$  position.
- ii. *Refresh buffer* that stores the intensity values for each position.

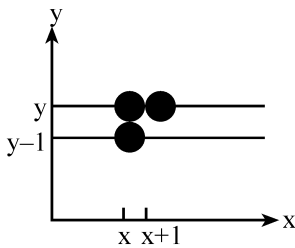
Initially, all positions in the depth buffer are set to 0 (minimum depth) and the refresh buffer is initialized to background intensity. Each surface listed in the polygon table is then processed, one scan line at a time, calculating the depth ( $z$  value) at each  $(x, y)$  fixed position. The calculated depth is compared to the value

previously stored in the depth buffer at that position. If the calculated depth is greater, then surface intensity at that position is determined and placed in the same (x, y) location in the refresh buffer.

### Depth calculation

Depth values for a surface position (x, y) can be calculated from the plane equation for each surface as

$$z = \dots\dots\dots(i)$$



**Figure 7.3:** From position (x, y) on a scan line, the next position across the line has coordinates (x+1, y), and position immediately below on the next line has coordinates (x, y-1)

For any scan line, adjacent x and y values differs by 1. If the depth of position (x, y) has been determined to be z, then the depth z' of the next position (x+1, y) along the scan line is obtained from equation (i) as

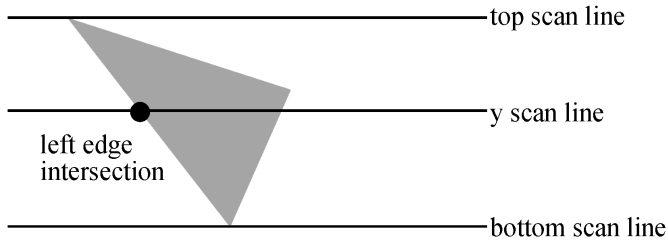
$$z' = \dots\dots\dots(ii)$$

$$z' = z - \dots\dots\dots(iii)$$

The ratio -A/C is constant for each surface, so succeeding depth values across a scan line are obtained from proceeding values with a single addition.

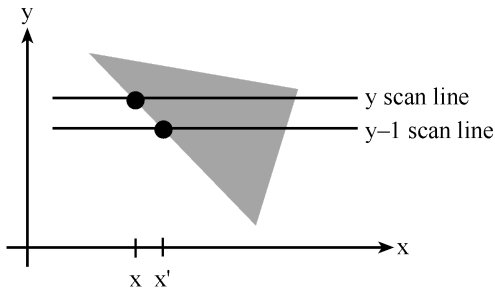
On each scan line, we start by calculating the depth on a left edge of the polygon that intersects that scan line. Depth values at each successive position across the scan line are then calculated by equation (iii).

We first determine the y-coordinate extents of each polygon, and process the surface from the topmost scan line to the bottom scan line as shown in figure.



**Figure 7.4:** Scan lines intersecting a Polygon surfaces

Starting at a top vertex, we can recursively calculate x positions down a left edge of the polygon as  $x' = x - \frac{y-y_1}{m}$ , where m is the slope of the edge (see Figure 13.7)



**Figure 7.5:** Intersection positions on successive scan lines along a left polygon edge.

Depth values down the edge are then obtained as

$$z' = \frac{-Ax' - By' - D}{C} = \frac{-A(x - \frac{y-y_1}{m}) - B(y-1) - D}{C} = \frac{-Ax - By - D}{C} + \frac{A(m+B)}{C} = Z$$

For vertical edge, the slope is infinite, so  $z' = z +$

For polygon surfaces, depth-buffer method is very easy to implement and it requires no sorting of the surfaces in a scene. However, it requires second buffer (depth buffer) in addition to the refresh buffer. Another drawback is that depth-buffer method can only find one visible surface at each pixel position. That is, it deals with only opaque surfaces and cannot accumulate intensity values



for more than one surface, as is necessary if transparent surfaces are to be displayed.

## 7.4 A-Buffer Method

This method is an extension of depth-buffer method. The A-buffer method represents an antialiased, area-averaged, accumulation-buffer method. It deals with transparent surfaces.

A-buffer method expands the depth-buffer so that each position in the buffer can reference a linked list of surfaces. Thus, more than one surface intensity can be taken into consideration at each pixel position, and object edges can be antialiased.

Each position in the A-buffer has two fields:

- **Depth field**– stores a positive or negative real number
- **Intensity field or Surface data field**– stores surface-intensity information or a pointer value.

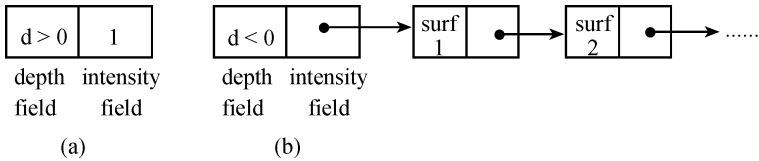
The first field is depth field which works as flag whether it is single surface or multiple surfaces.

If the depth field is positive ( $d \geq 0$ ), it indicates a single surface i.e., no overlapping. The number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point.

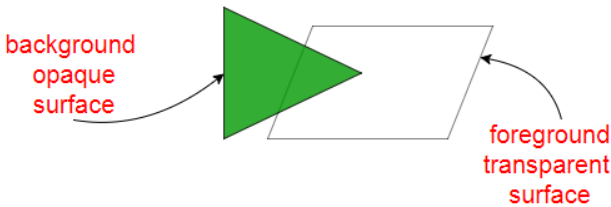
If the depth field is negative ( $d < 0$ ), this indicates multiple-surface contributions to the pixel intensity, i.e., surface is overlapped. The intensity field then stores a pointer to a linked list of surface data. Data for each surface in the linked list includes

- RGB intensity components
- Opacity parameter (percent of transparency)
- Depth
- Percent of area coverage

- Surface identifier
- Other surface-rendering parameters
- Pointer to next surface



**Figure 7.6:** Organization of an A-buffer pixel position: (a) single-surface overlap of the corresponding pixel area (b) multiple-surface overlap.



**Figure 7.5:** Viewing an opaque surface through a transparent surface requires multiple surface intensity contributions for pixel position

The demerit of A-buffer method is depth must be calculated and compared for each pixel. So, it consumes more resources.

## 7.5 Scan-Line Method

Scan-line method is the image-space method for removing multiple hidden surfaces. It is fast than A-buffer method as scan line method does not compare each pixel in non overlapping surface. It uses active edge list to set the intensity of the surface.

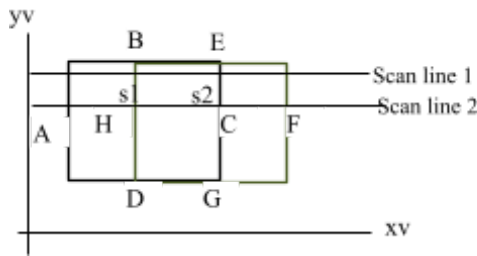
As each scan line is processed, all polygon surfaces intersecting that line are examined to determine which are visible. Across each scan line, depth calculations are made for each overlapping surface to determine which is nearest to the view

plane. When the visible surface is determined, the intensity value of that position is entered into the refresh buffer.

Different tables are set up for different surfaces: edge table and polygon table.

- i. **Edge table:** It contains coordinate endpoints of each line in the scene, inverse slope of each line, and pointers into the polygon table to identify the surfaces bounded by each line.
- ii. **Polygon table:** It contains coefficient of the plane equation for each surface, surface intensity information, and pointers to the edge table.

In order to search for surfaces crossing a given scan line, active edge lists are set up. It contains only edges that cross the current scan line, sorted in order of increasing x. Additionally, surface flag can be defined for each surface to indicate whether a position along a scan line is inside or outside the surface. At the leftmost boundary of a surface, the surface flag is turned on and at the rightmost boundary, surface flag is turned off.



**Figure 7.7:** Scan line crossing the projection of two surfaces,  $S_1$  and  $S_2$ , in the view plane

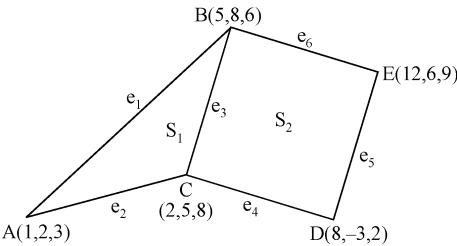
Active list for scan line 1 contains information from the edge table for edges AB, BC, EH and EF. For positions along this scan line between edges AB and BC only the flag for surface  $S_1$  is on. So, on depth calculations are necessary and the intensity information for surface  $S_1$  is entered from the polygon table into the refresh butter. Between edges EH and EF, only the flag for surface  $S_2$  is entered into the refresh buffer, while for all other positions the intensity values are set to the background intensity.

Similarly, active list for scan line 2 contains the edges AB, EH, BC and EF. Between edges AB and EH, only the flag for surface  $S_1$  is set on and intensity value for  $S_1$  is stored into refresh buffer.

Between edges EH and BC, the flags for both the surfaces  $S_1$  and  $S_2$  are set on. For this interval depth calculation must be done (using the plane coefficients) and depending upon which surface is closer to the view plane, its intensity value is stored into the refresh buffer. Between edges BC and EF, the flag for surface  $S_1$  goes off and the intensity value for surface  $S_2$  is stored into the refresh buffer.



1.    *Represent the following surfaces by polygon table method. Find the normal of surface  $S_1$ .*



[2076 Ashwin]

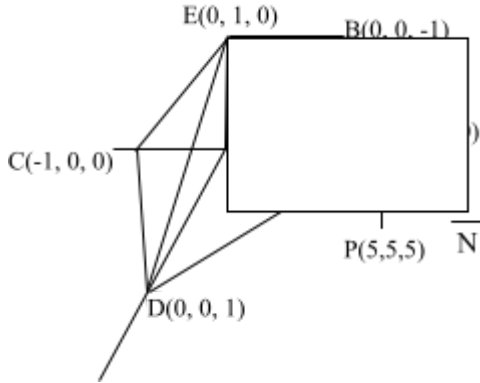
**Solution:**

Vertex table	Edge table	Polygon -surface table
A: 1,2,3	$e_1$ : A, B	$S_1$ : $e_1, e_2, e_3$
B: 5,8,6	$e_2$ : A, C	$S_2$ : $e_3, e_4, e_5, e_6$
C: 2,5,8	$e_3$ : C, B	
D: 8,-3,2	$e_4$ : C, D	
E: 12,6,9	$e_5$ : D, E	
	$e_6$ : E, B	

We can find the normal of surface  $S_1$  by taking the coordinates in consideration and viewig from positive z axis. As j component is maximum in AB and k component is maximum in AC, we do  $j \times k$ ,

$$\begin{aligned}
 N &= AB \times AC \\
 &= [(5-1)i + (8-2)j + (6-3)k] \times [(2-1)i + (5-2)j + (8-3)k] \\
 &= (4i + 6j + 3k) \times (i + 3j + 5k) \\
 &= 27i + 17j + 6k
 \end{aligned}$$

2. Find the visibility for the surface AED where observer at  $P(5,5,5)$ .



**Solution:**

**Step 1:**

Find the normal vector  $N$  for AED surface (always take anti-clockwise direction convention)

i.e.,  $AE \times AD$  not  $AD \times AE$

$$AE = OE - OA$$

$$= (0 - 1)i + (1 - 0)j + (0 - 0)k$$

$$= -i + j$$

$$N = AE \times AD$$

$$= (-i + j) \times (-i + k)$$

$$= i(1 - 0) - j(-1 + 0) + k(0 + 1)$$

$$= i + j + k$$

**Step 2:**

The observer is at  $P(5,5,5)$  so we can construct the view vector  $V$  from surface to view point  $A(1,0,0)$  as

$$\begin{aligned}V &= PA = (1 - 5)i + (0 - 5)j + (0 - 5)k \\&= -4i - 5j - 5k\end{aligned}$$

**Steps 3:**

To find the visibility of the object, we use dot product of view vector and normal vector  $N$  as

$$\begin{aligned}V \cdot N &= (-4i - 5j - 5k) \cdot (i + j + k) \\&= -4 - 5 - 5 \\&= -14 < 0\end{aligned}$$

This shows that the surface is visible for observer.



---

# Illumination and Surface Rendering Method

---



## 8.1 Illumination Models and Surface Rendering Technique

### 8.1.1 Illumination Model/Lighting Model/Shading Model

---

Illumination model is used to calculate the intensity of light at a given point on the surface of an object. An illumination model (equation) expresses the components of light reflected from or transmitted (refracted) through a surface. There are three basic light components: ambient light, diffuse, and specular reflections that are used to calculate the light intensity.

### 8.1.2 Surface Rendering Algorithms

---

Rendering is a method to calculate the total light intensity in the whole polygon surface. Rendering method uses the light intensity calculated from an illumination model to determine the light intensity for all projected pixel positions for the various surfaces in a scene. Rendering can be performed by applying the illumination model to every visible surface point, or the rendering can be accomplished by interpolating intensities across the surfaces from a small set of illumination model calculations.

## **Illumination models take account into several factors which are as follows:**

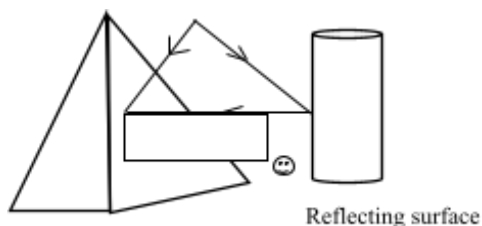
- i. Optical properties of the surfaces (transparency, reflectivity, surface texture)
- ii. Relative positions of the surfaces in a scene.
- iii. Color and position of the light sources
- iv. Position and orientation of the viewing plane.

### **8.2 Light Source**

Light source is an object that emits radiant light energy. Light sources can be of different types like, light emitting sources, reflecting sources, point source, distributed source. Total reflected light from an opaque non luminous object is the sum of the contributions from light source and other reflecting surfaces in the scene. So, a surface that is not directly exposed to a light source may still be visible if nearby objects are illuminated. Light emitting sources are light bulbs, sun etc. Light reflecting sources are walls of a room, other reflecting surfaces etc. A luminous object, in general, can be both a light source and a light reflector, e.g. a plastic globe with a light bulb, etc.

#### **1. Point Source**

Point source is a simplest model for a light emitter. It shines light equally in all directions. The dimension of point source is small compared to the size of objects in the scene. E.g., Sun

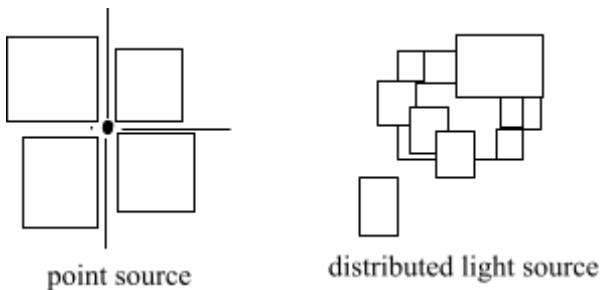


**Figure 8.1:** Light viewed from an opaque nonluminous surface is in general a combination of reflected light from a light source and reflections from other surfaces



2.     **Distributed Light Source**

The area of the distributed light source is not small compared to the surface in the scene. A nearby source, such as the long fluorescent light is an example of a distributed light source.

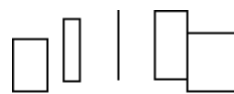


**8.3 Light Reflection**

When light is incident on an opaque surface, part of it is reflected and part is absorbed. Amount of incident light reflected by a surface depends on the type of nature. Shining materials reflect more of the incident light, and dull surfaces absorb more of the incident light. Similarly, for an illuminated transparent surface some of the incident light will be reflected and some will be transmitted through the material.

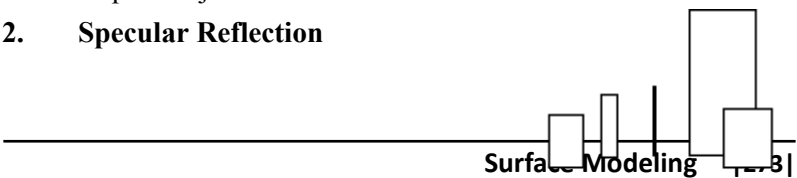
1.     **Diffuse Reflection**

Diffuse reflection of light is the reflection of incident ray on the surface in many angles rather than at just one angle as in the case of specular reflection. An ideal diffuse reflecting surface is said to exhibit Lambertian reflection, i. e., there is equal luminance when viewed from all directions lying in the half-space adjacent to the surface.



**Fig 8.2:** Diffuse reflection  
superimposed on diffuse reflection

2.     **Specular Reflection**



**Fig 8.3:** Specular reflection superimposed on diffuse reflection

Specular or regular reflection is the mirror-like reflection of light, i. e., each incident ray is reflected at the same angle of incident ray, but on the opposite side of the surface with respect to the normal at that point. The result is that an image reflected by the surface is reproduced in mirror-like (specular) fashion. The law of reflection states that for each incident ray the angle of incidence equals the angle of reflection, and the incident, normal, and reflected directions are coplanar.

## 8.4 Basic Illumination Models

Illumination model is method for calculating light intensities. Light calculations are based on the optical properties of the surface, the back ground lighting conditions and the light-source specifications. Optical parameters are used to set surface properties such as transparency, opacity etc, and these control the amount of reflection and absorption of incident light. All light sources are considered to be point source specified with a co-ordinate position an intensity value.

### 1. Ambient Light

Ambient light is the light reflected from various surfaces to produce a uniform illumination. It is also called background light. Ambient light has no spatial or directional characteristics. A surface that is not directly exposed to a light source will still be visible because of ambient light.

Amount of ambient light incident on each object is constant for all surfaces and over all directions, but the intensity of the reflected light for each surface depends on the optical properties of the surface. So if  $I_a$  is the amount of ambient light incident on any surface, the ambient light reflection is given by ambient illumination equation,

$$I = K_a I_a$$

We have  $K_a$  is ambient reflectivity or ambient reflection coefficient which ranges from 0 to 1. It is a property of material.

### 2. Diffuse Reflection

Ambient light reflection is an approximation of global diffuse lighting effects. Diffuse reflections are constant over each

surface in a scene, independent of viewing direction. Amount of incident light that is diffusively reflected is defined with a surface parameter  $K_d$  called diffuse-reflection coefficient or diffuse-reflectivity.

$K_d$  is assigned as a constant value in the interval 0 to 1, according to the reflecting properties of the surface to have. For highly reflecting surface, the value of  $K_d$  is tends to 1 and for a very dull surface the value of  $K_d$  is tends to 0. If a surface is exposed only to ambient light, the intensity of diffuse reflection at any point the surface is

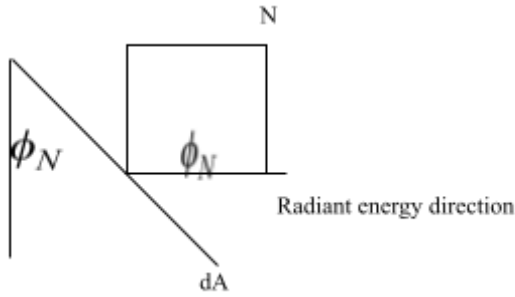
$$I_{\text{ambdiff}} = K_d I_a$$

Ambient light produces flat uninteresting shading for each surface. So, scenes are rarely rendered with ambient light alone. At least one light source is included in a scene, often as a point source at the viewing position.

### **Ideal Diffuse Reflector**

It scatters diffuse reflections from the surface with equal intensity in all directions. Since radiant light energy from any point on the surface is governed by Lambert's cosine law, ideal diffuse reflector is often known as Lambertian reflector. Lambert's cosine law states that the radial energy from any small surface area  $A$  in any direction  $N$  relative to the surface normal is proportional to  $\cos\phi_N$  and the light intensity depends on the radial energy per projected area perpendicular to direction  $\phi_N$ .

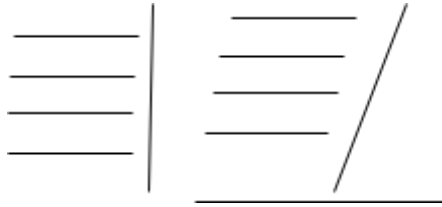
$$\text{i.e. } I = dA \cos\phi_N$$



**Fig 8.4:** Radiant energy from a surface area  $dA$  in direction  $\phi_N$  relative to the surface normal direction

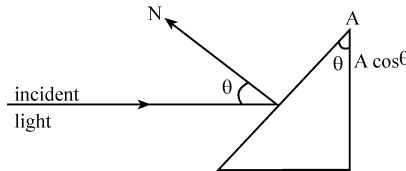
Thus, for Lambertian reflection, the intensity of light is the same over all viewing direction. Even though there is equal light scattering in all directions from a perfect diffuse reflector, the brightness of the surface does depend on the orientation of the surface relative to the light source.

Surface perpendicular to the direction of incident light appears brighter than the one with some oblique angle to the direction of the incoming light.



**Fig 8.5:** Specular reflection superimposed on diffuse reflections

If  $\theta$  is angle of incidence between the incoming light direction and the surface normal then the projected area of a surface path perpendicular to light direction is proportional to  $\cos\theta$



**Fig 8.6:** *An illuminated area projected perpendicular to the path of the incoming light rays*

Thus, the amount of illumination (or the number of incident light rays cutting the projected surface path) depends on  $\cos\theta$ .

If the incoming light from the source is perpendicular to the surface at a particular point, that point is fully illuminated.

As the angle of illumination moves away from the surface normal the brightness of the point drops off.

If  $I_l$  is the intensity of the point light source, then the diffuse reflection equation for a point on the surface is,

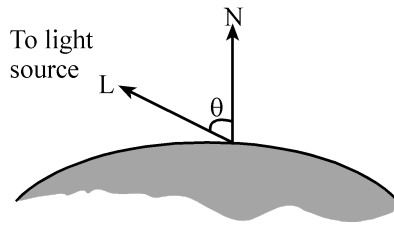
$$I_{l,\text{diff}} = K_d I_l \cos\theta.$$

A surface is illuminated by a point source only if the angle of incidence is in the range  $0^\circ$  to  $90^\circ$  for which  $\cos\theta$  is in the range 0 to 1. When  $\cos\theta$  is negative, the light source is behind the surface.

If  $N$  is the unit normal vector to a surface and  $L$  is the direction vector to the point source from a position on the surface then,

$$\cos\theta = NLNL$$

$$I_{l,\text{diff}} = K_d I_l (N.L)$$



**Figure 8.7:** *Angle of incident  $\theta$  between the unit light-source direction vector  $\vec{L}$  and the unit surface normal  $\vec{N}$ .*

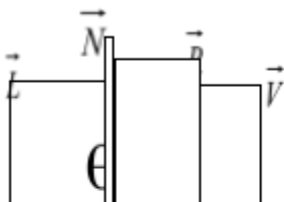
We can combine the ambient and point source intensity calculations to obtain an expression for the total diffuse reflection

$$I_{\text{diff}} = K_a I_a + K_d I_l (N.L)$$

Whose both  $k_a$  and  $K_d$  depends upon surface material properties and are assigned values in the range from 0 to 1.

### 3. Specular Reflection and the Phong Model

Specular reflection is the highlight or bright spots seen on shiny surfaces. It is a result of total or near total reflection of the incident light in a concentrated region around the specular reflection angle.



**Figure 8.8:** Specular-reflection angle equals angle of incident  $\theta$ .

$L$  = unit vector directed towards the point light source

$V$  = unit vector pointing to the viewer from the surface position

$R$  = unit vector in the direction of ideal specular reflection

$N$  = unit normal vector of the surface

$\phi$  = angle between vector  $R$  and vector  $V$  (also called a viewing angle)

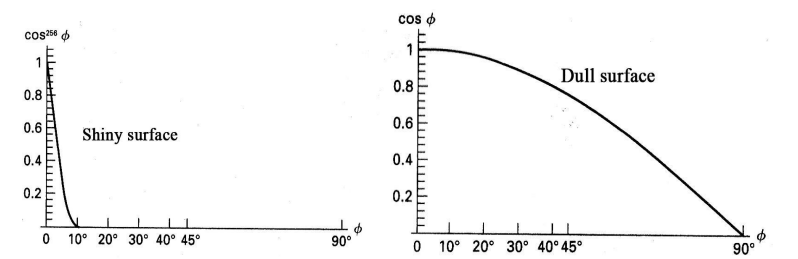
For an ideal reflector (perfect mirror), incident light is reflected only in the specular-reflection direction, i.e.  $V$  and  $R$  vectors coincide and  $\phi = 0$ . Objects other than ideal reflectors exhibit specular reflections over a finite range of viewing positions around vector  $R$ . Shiny surfaces have a narrow specular-reflection range, where as dull surfaces have a wider reflection range. An empirical model for calculating the specular reflection range was invented by

the Phong Bui Tuong. It is also known as **Phong specular reflection model**. This model sets the intensity of specular reflection directly proportional to  $\cos^n\phi$ . Where  $\phi$  can be assigned values in the range  $0^\circ$  to  $90^\circ$  ( $\cos\phi = 0$  to  $1$ ) and  $n_s$  is a **specular reflection parameter** whose value is determined by the type of surface to be displayed. The value of  $n_s$  for brighter (shiny) surfaces could be 100 or more whereas for dull surfaces its value is 1 or less than 1 and for a perfect reflector,  $n_s$  is infinite.

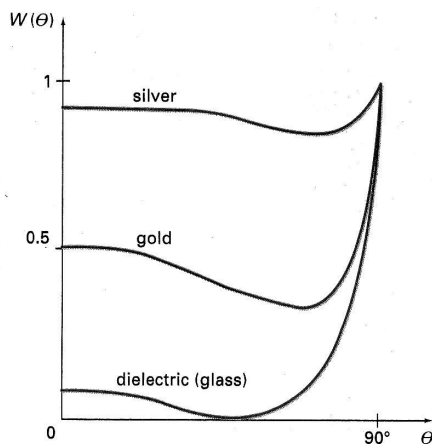
The intensity of specular reflection depends on the object (material) properties of the surface & the angle of light incidence, as well as other factors such as the polarization and color of the light incident. We can model the intensity variation of the light through, **specular-reflection**, using **spectral-reflection function**  $W(\theta)$  for each surface. Where the value of  $W(\theta)$  lies between 0 to 1. The value of  $\theta$  lies between  $0^\circ$  to  $90^\circ$ . In general  $W(\theta)$  tend to increase as the angle of incidence increases, at  $\theta = 90^\circ$ ,  $W(90^\circ)=1$ , and in this case, all the light incidents on the surface of the material is reflected. Phong model calculates the specular reflection light intensity as

$$I_{\text{spec}} = W(\theta) I_i \cos^n\phi$$

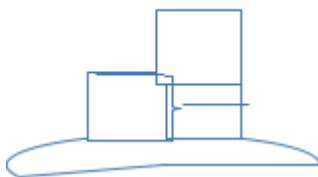
Where  $I_i$  = incident light intensity  
 $W(\theta) = 1$  at  $\theta = 90^\circ$ , and all of the incident light is reflected.  
 $W(\theta)$  = Specular-reflection coefficient



**Figure 8.9:** Plots of  $\cos^n_s\phi$  for several values of specular parameter  $n_s$



**Figure 8.10:** Approximate variation of the specular-reflection coefficient as a function of angle of incidence for different materials.



**Figure 8.11:** Calculation of vector  $R$  by considering projections onto the direction of the normal vector  $N$ .

The projection of  $L$  onto the direction of the normal vector is obtained with the dot product  $N \cdot L$

(The vector projection of  $L$  onto  $N$  is  $NLNN$ )

The scalar projection  $L$  onto  $N$  is  $N \cdot L$ )

So,

$$R + L = (2N \cdot L)N$$

$$R = (2N \cdot L)N - L$$

As seen from the figure for  $W(\theta)$  vs  $\theta$ , the transparent materials, such as glass, only exhibit appreciable specular reflections as  $\theta$  approaches  $90^\circ$ . But many opaque objects exhibit almost constant



specular reflection for all incidence angles. For this case, we can replace  $W(\theta)$ , with a constant specular-reflection coefficient  $k_s$ , whose value can be assigned in the range 0 to 1.

Also,

$$\cos\phi \quad VRVR = V.R$$

$$so, I_{spec} = K_s I_l (V.R)^n$$

Now,

Combined diffuse and specular reflection,

$$\begin{aligned} I &= I_{diff} + I_{spec} \\ &= K_a I_a + k_d I_l (N.L) + k_s I_l (V.R)^n \end{aligned}$$

If there are more than one light sources in the scene, then

$$I = K_a I_a + \sum_{i=1}^n I_{li}$$

or

$$I = K_a I_a + \sum_{i=1}^n I_{li}$$

Where  $H$  is halfway vector between  $L$  and  $V$  and given by;

$$H = LVLV$$

If both the viewer and light source are sufficiently far from the surface, both  $V$  and  $L$  are constant, so,  $H$  is also constant for all surface points. So, for nonplanar surfaces,  $N.H$  is less computative than  $V.R$ .

## Intensity Attenuation

As radiant energy from a point source travels through space, its amplitude is attenuated by the factor  $1/d^2$ , where  $d$  is the distance that the light has traveled. A surface close to the light source (small  $d$ ) receives higher incident intensity from the source than a distant surface (large  $d$ ). So, for realistic lighting effects, we should take into account the intensity attenuation, otherwise, it produces unrealistic effect as we will be illuminating all surfaces with the same intensity, no matter how far they might be from the light source. But simple point source illumination model does not

always produce realistic pictures, if we use the factor  $1/d^2$  to attenuate intensities, as it produces very little variation when  $d$  is large. Graphics package have compensated this problem by using universal quadratic attenuation function as the attenuation factor.

$$f(d) = 1/K_c + K_l d + K_q d^2$$

Where  $d$  is the distance between the light and the surface being shaded,  $K_c$  is constant attenuation factor,  $K_l$  is linear attenuation factor,  $K_q$  is quadratic attenuation factor.

Using the attenuation function

$$I = K_a I_a + \sum_{i=1}^n f(d_i) I_{1i}$$

## 8.5 Surface Rendering Methods

The surface rendering method is used to find the total light intensity in the whole surface. To find the total light intensity, it interpolates the small set of intensity of certain pixel, calculated by using illumination model, across the surface. The standard objects that are formed with polygon surfaces are rendered by the application of polygon rendering methods. Some of the methods are as follows;

1. Constant shading
2. Gouraud shading
3. Phong shading
4. Fast Phong shading

### 1. Constant Shading

Constant shading is also called flat shading. It is fast and simple method. In this method a single intensity is calculated for each polygon and all points over the surface of the polygon are then displayed with the same intensity value. This method is useful for quickly displaying the general appearance of a curved surface. This method provides an accurate rendering for an object if all of the following assumptions are valid

- i. The object is a polyhedron and is not an approximation of an object with a curved surface.

- ii. All light source illuminating the object are sufficiently far from the surface so that N.L and the attenuation function are constant over the surface.
- iii. The viewing position is constant over the surface or sufficiently far from the surface so that V.R is constant over the surface.

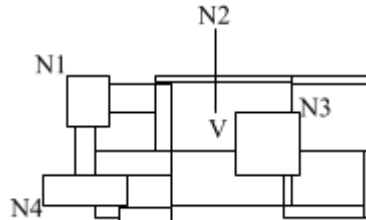
The demerit of this method is that the sharp intensity discontinuation is seen in the border between two polygons.

## 2. Gouraud Shading (Intensity Interpolation Method)

In Gouraud shading method rendering is done by linearly interpolating intensity values across the surface. Intensity values for each polygon are matched with the values of adjacent polygons along the common edges. This eliminates the intensity discontinuities that can occur in flat shading. This method produces more realistic results, but requires considerably more calculations.

Each polygon surface is rendered with Gouraud shading by performing the following calculations:

- i. Determine the average unit normal vector at each polygon vertex

$$\vec{N}_v = \frac{\sum_{k=1}^n \vec{N}_k}{\left| \sum_{k=1}^n \vec{N}_k \right|}$$


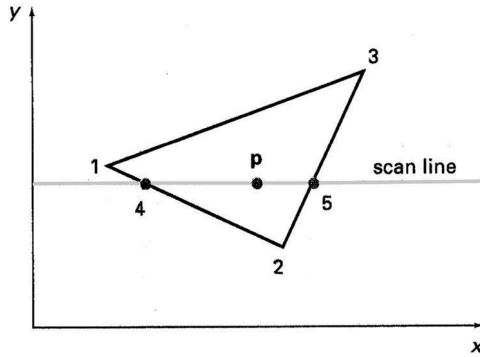
**Figure 8.12:** The normal vector at vertex *V* is calculated as the average of the surface normals for each polygon sharing that vertex.

- ii. Apply an illumination model to each vertex to calculate the vertex intensity.

$$I = K_a I_a + k_d I_l (N.L) + k_s I_l (V.R)^n$$

- iii. Linearly interpolate the vertex intensities over the surface of the polygon.

For each scan line, the intensity at the intersection of the scan line with a polygon edge is linearly interpolated from the intensities at the edge endpoints.



**Figure 8.13:** For ground shading, the intensity at point 4 is linearly interpolated from the intensities at vertices 1 and 2. The intensity at point 5 is linearly interpolated from intensities at vertices 2 and 3. An interior point P is then assigned an intensity value that is linearly interpolated from intensities at positions 4 and 5.

As shown in figure, a scan line intersects two edge 12 and 23 now, obtain intensity value at any point P along the scan line, we do calculation in the following manner,

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} \times I_1 + \frac{y_1 - y_4}{y_1 - y_2} \times I_2$$

Similarly,

$$I_5 = \frac{y_5 - y_3}{y_2 - y_3} \times I_2 + \frac{y_2 - y_5}{y_2 - y_3} \times I_3$$

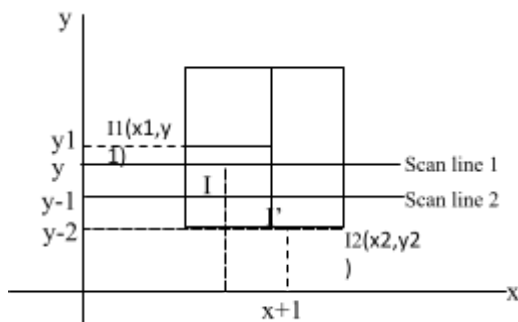
$$\text{then, } I_p = \frac{xp - x_5}{x_4 - x_5} \times I_4 + \frac{x_4 - xp}{x_4 - x_5} \times I_5$$

And, along the edge we make incremental calculations for intensity values

$$I = \frac{y - y_2}{y_1 - y_2} \times I_1 + \frac{y_1 - y}{y_1 - y_2} \times I_2$$

$$\begin{aligned}
 I' &= \times I_1 + \times I_2 \\
 &= \times I_1 + \times I_2 \\
 &= \times I_1 - + \times I_2 + \\
 &= I - + \\
 &= I +
 \end{aligned}$$

$$\text{so, } I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$



**Figure 8.14:** Incremental interpolation of intensity values along polygon edge for successive scan lines.

We make similar calculations to obtain successive intensity values along horizontal line.

### Advantages

- Removes the intensity discontinuities associated with the constant shading model.

### Disadvantages

- Linear intensity interpolation can cause bright or dark intensity streaks, called mach bands, to appear on the surface. This effect can be reduced by increasing the number of polygons while representing the object.

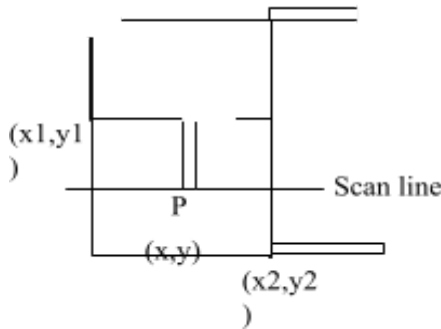
## 3. Phong shading (Normal Vector Interpolation shading):

Phong shading model is normal vector interpolation method. It greatly reduces mach band effect. It is more accurate interpolation method as it interpolates the vector normal over the

surface of the polygon. It displays more realistic surface reducing mach band effect.

### Steps:

- i. Determine the average unit normal vector at each polygon vertex.
- ii. Linearly interpolate the vertex normals over the surface of the polygon
- iii. Apply an illumination model along each scanline to calculate projected pixel intensities for the surface points.



**Figure 8.15:** Interpolation of surface normals along a polygon edge.

As shown in figure, intensity at point along an edge is calculated first by interpolating the normal vectors for the end points of the edge, and finally applying the illumination model.

$$\vec{N} = \frac{y-y_2}{y_1-y_2} \times \vec{N}_1 + \frac{y_1-y}{y_1-y_2} \times \vec{N}_2$$

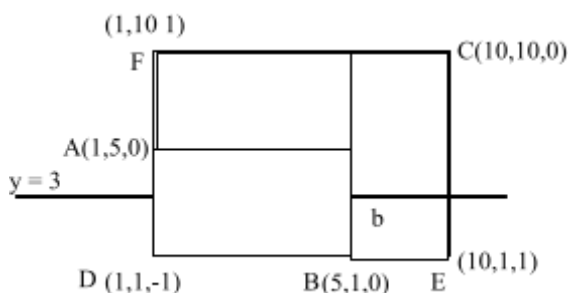
### Advantages

- Displays more realistic highlights in a surface.
- Reduce mach band effect.

### Disadvantages

- Requires more computation than the Gouraud shading method.
- It is expensive rendering method.

- Find out the intensity of light reflected from the midpoint  $P$  on scan line  $y = 3$  in the above given figure using Gouraud shading model. Consider a single point light source located at positive intensity on  $z$ -axis and assume vector to eye as  $(1, 1, 1)$ . Given  $d = 0$ ,  $k_c = k_l = k_q = 1$ ,  $l_a = 1$ ,  $l_l = 10$ ,  $n_s = 2$ ,  $K_a = K_d = 0.1$ ,  $K_s = 0.8$  for use in a simple illumination model.



**Solution:**

We know that,

$$\rightarrow \times \rightarrow = \rightarrow, \quad \rightarrow \times \rightarrow = \rightarrow, \quad \rightarrow \times \rightarrow = \rightarrow$$

Total intensity is,

$$I = I_a K_a + NLVR$$

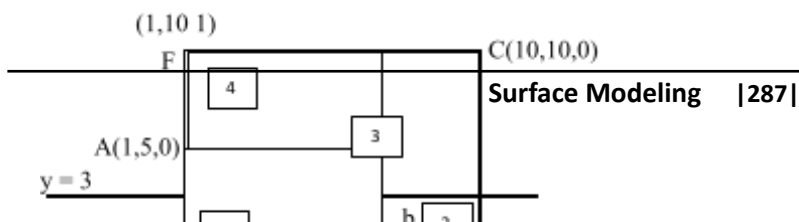
According to the question, vector to eye is  $(1, 1, 1)$ . So,

$$\rightarrow = \rightarrow + \rightarrow + \rightarrow$$

$$\rightarrow = VV = ijk = 0.58 \rightarrow + 0.58 \rightarrow + 0.58 \rightarrow$$

Let us denote the  $\triangle ADB$  as 1,  $\triangle BCE$  as 2,  $\triangle ABC$  as 3,  $\triangle ACF$  as 4.

Normal vectors of  $\triangle ADB$ ,  $\triangle BCE$ ,  $\triangle ABC$ ,  $\triangle ACF$  be  $\rightarrow$ ,  $\rightarrow$ ,  $\rightarrow$ , and  $\rightarrow$  respectively.



In  $\triangle ADB$ ,

$$\begin{aligned}
 \rightarrow &= \rightarrow \times \rightarrow \\
 &= (\rightarrow - \rightarrow) \times \rightarrow - \rightarrow \\
 &= \{(5-1) \rightarrow + (1-1) \rightarrow + (0+1) \rightarrow\} \times \{(1-1) \rightarrow + (5-1) \rightarrow + (0+1) \rightarrow\} \\
 &= (4 \rightarrow + \rightarrow) \times (4 \rightarrow + \rightarrow) \\
 &= 16 \rightarrow - 4 \rightarrow - 4 \rightarrow \\
 &= -4 \rightarrow - 4 \rightarrow + 16 \rightarrow
 \end{aligned}$$

In  $\triangle BEC$ ,

$$\begin{aligned}
 \rightarrow &= \rightarrow \times \rightarrow \\
 &= (5 \rightarrow + \rightarrow) \times (5 \rightarrow + 9 \rightarrow) \\
 &= 45 \rightarrow + 5 \rightarrow - 9 \rightarrow \\
 &= -9 \rightarrow + 5 \rightarrow + 45 \rightarrow
 \end{aligned}$$

In  $\triangle ABC$ ,

$$\begin{aligned}
 \rightarrow &= \rightarrow \times \rightarrow \\
 &= (5 \rightarrow + 9 \rightarrow) \times (-4 \rightarrow + 4 \rightarrow) \\
 &= 20 \rightarrow + 36 \rightarrow \\
 &= 56 \rightarrow
 \end{aligned}$$

In  $\triangle ACF$ ,

$$\begin{aligned}
 \rightarrow &= \rightarrow \times \rightarrow \\
 &= (9 \rightarrow + 5 \rightarrow) \times (5 \rightarrow + \rightarrow) \\
 &= 45 \rightarrow - 9 \rightarrow + 5 \rightarrow
 \end{aligned}$$

Average normal vector in A,

$$\begin{aligned}
 \rightarrow &= \rightarrow + \rightarrow + \rightarrow \\
 &= -4 \rightarrow - 4 \rightarrow + 16 \rightarrow + 56 \rightarrow + 45 \rightarrow - 9 \rightarrow + 5 \rightarrow
 \end{aligned}$$



$$= -13 \rightarrow +117 \rightarrow$$

Average unit normal vector in A,

$$\bar{n} = \frac{1}{3} \sum \mathbf{n}_i$$

$$= 0.01 \rightarrow -0.11 \rightarrow +0.99 \rightarrow$$

Similarly,

$$\rightarrow = \rightarrow + \rightarrow + \rightarrow$$

$$= -4 \rightarrow -4 \rightarrow +16 \rightarrow -9 \rightarrow +5 \rightarrow +45 \rightarrow +56 \rightarrow$$

$$= -13 \rightarrow + \rightarrow +117 \rightarrow$$

$$\bar{n} = -0.11 \rightarrow +0.01 \rightarrow +0.99 \rightarrow$$

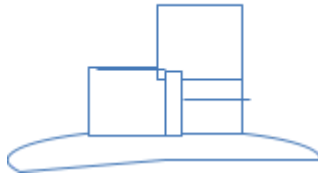
$$\rightarrow = \rightarrow + \rightarrow + \rightarrow$$

$$= -4 \rightarrow -4 \rightarrow +16 \rightarrow -9 \rightarrow +5 \rightarrow +45 \rightarrow +45 \rightarrow -9 \rightarrow +5 \rightarrow$$

$$= 146 \rightarrow -4 \rightarrow -4 \rightarrow$$

$$\bar{n} = -0.027 \rightarrow -0.027 \rightarrow +0.99 \rightarrow$$

Now, calculate unit reflection vectors  $\rightarrow$ ,  $\rightarrow$ ,  $\rightarrow$  at vertices A, B, and C respectively.



The projection of L onto the direction of the normal vector is obtained with the dot product  $\mathbf{N} \cdot \mathbf{L}$ .

So,

$$\mathbf{R} + \mathbf{L} = 2(\mathbf{N} \cdot \mathbf{L})\mathbf{N}$$

$$\mathbf{R} = 2(\mathbf{N} \cdot \mathbf{L})\mathbf{N} - \mathbf{L}$$

$$\rightarrow = 2(\bar{n} \cdot \rightarrow) \bar{n} - \rightarrow$$

$$= 2\{(0.01 \rightarrow -0.11 \rightarrow +0.99 \rightarrow) \cdot \rightarrow\}(0.01 \rightarrow -0.11 \rightarrow +0.99 \rightarrow) - \rightarrow$$

$$= 2 \times 0.99 \times (0.01 \rightarrow -0.11 \rightarrow +0.99 \rightarrow) - \rightarrow$$

$$= 0.02 \rightarrow -0.22 \rightarrow +1.96 \rightarrow - \rightarrow$$

$$\begin{aligned}
 &= 0.02 \rightarrow -0.22 \rightarrow +0.96 \rightarrow \\
 \blacktriangle &= \rightarrow \rightarrow \rightarrow \\
 &= 0.02 \rightarrow -0.22 \rightarrow +0.97 \rightarrow
 \end{aligned}$$

Again,

$$\begin{aligned}
 \rightarrow &= 2 ( \blacktriangle . \blacktriangle ) \blacktriangle - \blacktriangle \\
 &= 2 \{ (0.11 \rightarrow + 0.01 \rightarrow + 0.99 \rightarrow), \rightarrow \} (0.11 \rightarrow + 0.01 \\
 &\quad \rightarrow + 0.99 \rightarrow) - \rightarrow \\
 &= 2 \times 0.99 \times (-0.11 \rightarrow + 0.01 \rightarrow + 0.99 \rightarrow) - \rightarrow \\
 &= 0.22 \rightarrow -0.02 \rightarrow + 1.96 \rightarrow - \rightarrow \\
 &= 0.02 \rightarrow -0.22 \rightarrow + 0.96 \rightarrow \\
 \blacktriangle &= -0.22 \rightarrow + 0.02 \rightarrow + 0.97 \rightarrow
 \end{aligned}$$

Similarly,

$$\begin{aligned}
 \rightarrow &= 2 ( \blacktriangle . \blacktriangle ) \blacktriangle - \blacktriangle \\
 &= 2 \{ (146 \rightarrow -4 \rightarrow -4 \rightarrow), \rightarrow \} (146 \rightarrow -4 \rightarrow -4 \rightarrow) - \\
 &\rightarrow \\
 &= 2 \times 0.99 \times (-0.027 \rightarrow -0.027 \rightarrow + 0.99 \rightarrow) - \rightarrow \\
 &= -0.054 \rightarrow -0.054 \rightarrow + 1.96 \rightarrow - \rightarrow \\
 &= 0.05 \rightarrow -0.054 \rightarrow + 0.96 \rightarrow \\
 \blacktriangle &= -0.056 \rightarrow -0.056 \rightarrow + 0.99 \rightarrow
 \end{aligned}$$

Now,

$$\begin{aligned}
 \blacktriangle . \blacktriangle &= (0.58 \rightarrow + 0.58 \rightarrow + 0.58 \rightarrow), (0.02 \rightarrow -0.22 \\
 &\quad \rightarrow + 0.97 \rightarrow) \\
 &= 0.0116 - 0.1276 + 0.5626 \\
 &= 0.4466 \\
 \blacktriangle . \blacktriangle &= (0.58 \rightarrow + 0.58 \rightarrow + 0.58 \rightarrow), (-0.22 \rightarrow + 0.02 \\
 &\quad \rightarrow + 0.97 \rightarrow) \\
 &= -0.1276 + 0.0116 + 0.5626 \\
 &= 0.4466 \\
 \blacktriangle . \blacktriangle &= (0.58 \rightarrow + 0.58 \rightarrow + 0.58 \rightarrow), (-0.056 \rightarrow - \\
 &\quad 0.056 \rightarrow + 0.99 \rightarrow) \\
 &= -0.0325 - 0.0325 + 0.5742
 \end{aligned}$$

$$= 0.50924$$

Similarly,

$$I_{a_1} = 0.99$$

$$I_{a_2} = 0.99$$

$$I_{a_3} = 0.99$$

Calculate intensities  $I_A$ ,  $I_B$ ,  $I_C$  at vertices A, B, and C respectively.

$$I = I_a K_a + NLVR$$

$$I_A = 1 \times 0.1 + 0.1 \times 10 \times 0.99 + 0.8 \times 10 \times (0.44)^2 \\ = 2.64$$

$$I_B = 2.64$$

$$I_C = 3.1$$

Interpolate intensities  $I_a$ ,  $I_b$ , and  $I_p$  at a, b and P respectively,

Referring the figure, we have,

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

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

$$z = z_1 + u (z_2 - z_1)$$

So,  $u =$

In line AaB,

$$u = (3 - 5) / (1 - 5) \\ = 0.5$$

$$z = z_1 + \Delta z u \\ = 0 + 0 \times 0.5 \\ = 0$$

$$y = y_1 + \Delta y u \\ = 5 + (-4) \times 0.5 \\ = 3$$

$$x = x_1 + \Delta x u \\ = 1 + (4) \times 0.5 \\ = 3$$

For line Bbc,

$$u = (3 - 1) / (10 - 1) \\ = 2/9$$

$$z = 0 + 0 \times 2/9 \\ = 0$$

$$x = 5 + 5 \times 2/9 \\ = 6.11$$

$$y = 1 + 9 \times 2/9 \\ = 3$$

Using the slope of the edges, the co-ordinates of a and b are found to be (3, 3, 0) and (6.11, 3, 0) respectively.

The coordinates of P, the midpoint of a b is found (4.56, 3, 0).

Now, we have apply 3 stage interpolation technique to determine  $I_p$ .

$$= \\ I_a = \times I_A + \times I_B \\ = \times 2.64 + \times 2.64 \\ = 2.64$$

$$I_b = \times I_C + \times I_B \\ = \times 3.1 + \times 2.64 \\ = 2.74$$

$$\text{Then } x_p = 4.555$$

Intensity of P is,

$$I_p = \times I_b + \times I_a \\ = \times 2.74 + \times 2.64 \\ = 2.69$$

The intensity of light reflected from the midpoint P is 2.69 candela.

2. *Find out the total intensity at the centroid of a triangle defined by A(2, 1, 1), B(0, 1, 1), C(0, 0, 1). When illuminated by a point light source of intensity  $I_l = 0.6$  at (2, 2, 6) using illumination model. The viewer is at (2, 3, 6).*

Assume ambient intensity  $I_a = 0.1$  and parameters  $K_a = 0.5$ ,  $K_d = 0.8$ ,  $K_s = 0.7$ , take  $n = 10$ .

[Centroid =  $(x_1 + x_2 + x_3)/3$ ,  $(y_1 + y_2 + y_3)/3$ ,  $(z_1 + z_2 + z_3)/3$ ]

[2079 Baishakh]

### Solution:

The vertices of triangle A, B, C are

A = (2, 1, 1)

B = (0, 1, 1)

C = (0, 0, 1)

Position of light source = (2, 2, 6)

Position of the viewer = (2, 3, 6)

$I_l = 0.6$ ,  $I_a = 0.1$

$K_a = 0.5$

$K_d = 0.8$

$K_s = 0.7$

$n = 10$

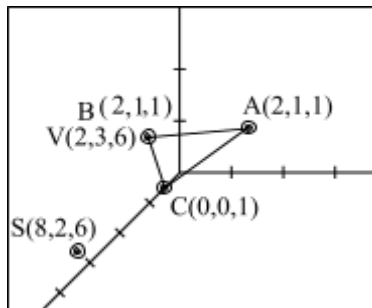
The centroid of the triangle is

$A_c = 0.67$

$B_c = 0.67$

$C_c = 1$

$(A_c, B_c, C_c) = (0.67, 0.67, 1)$



Now,

$I = I_a K_a + NLVR$

Ambient light

$$I_a K_a = 0.1 \times 0.5 = 0.05$$

Diffuse light

$$N = \rightarrow \times \rightarrow$$

$$= [(2-0)i + (1-0)j + (1-1)k] \times [(0-0)i + (1-0)j + (1-1)k]$$

$$= (2i + j) \times j$$

$$= 2k$$

$$N = k$$

$$L = (2-0.67)i + (2-0.67)j + (6-1)k$$

$$= 1.33i + 1.33j + 5k$$

$$L = ijk$$

$$= 0.2489i + 0.2489j + 0.9358k$$

$$I_a K_d (N.L) = 0.6 \times 0.8 (k.(0.2485i + 0.2489j + 0.9358k))$$

$$= 0.6 \times 0.8 \times 0.9358$$

$$= 0.44918$$

Specular light

$$R = 2N (N.L) - L$$

$$= 2 \times k (k.(0.2489i + 0.2489j + 0.9358k))$$

$$- (0.2489i + 0.2489j + 0.5358k)$$

$$= 2 \times k (0.9358) - 0.2489i - 0.2489j - 0.9358k$$

$$= -0.2489i - 0.2489j + 0.9358k$$

$$R = -0.249i - 0.249j + 0.9367k$$

$$V = (2, 3, 6) - (0.67, 0.67, 1)$$

$$= (1.33, 2.33, 5)$$

$$= 1.33i + 2.33j + 5k$$

$$V = 0.2344i + 0.4106j + 0.88117k$$

$$R.V = (-0.249i - 0.249j + 0.9367k).(0.2344i$$

$$+ 0.4106j + 0.88117k)$$

$$= -0.05836 - 0.102239 + 0.82539$$

$$= 0.664791$$

$$I_a K_s (R.V) = 0.6 \times 0.7 \times (0.664791)^{10} = 0.00708$$

$$\begin{aligned}\text{Total} &= \text{Ambient} + \text{Diffuse} + \text{Specular} \\ &= 0.05 + 0.44918 + 0.00708 \\ &= 0.50626\end{aligned}$$

---

# Introduction to OpenGL

---

## 9.1 Introduction

Open Graphics Library (OpenGL) is a graphics rendering application programming interface (API) to create and manipulate two and three dimensional graphics images. It is language independent, operating system independent, and platform independent software interface with large set of function which generates high-quality color images.

This interface consists of 250 distinct commands (200 in core OpenGL and 50 in OpenGL Utility Library) to draw complex scenes from simple geometric primitives such as points, lines, and polygons.

## 9.2 OpenGL Libraries

There are number of windowing system and interface libraries available in OpenGL. They are *OpenGL library*, *OpenGL Utility library*, and *OpenGL Utility Toolkit library*.

*OpenGL library* fuction begins with a prefix "gl", *OpenGL Utility Library* (GLU) functions start with a prefix "glu", *OpenGL Utility Toolkit library* (GLUT) functions start with a prefix of "glut". So, from the prefix, we can identify which library the function belongs to.

### i. GL or OpenGL

*Core OpenGL library* fuction begins with a prefix "gl" and initial capital letters for each word of the command name, e.g., glBegin(), glColor(), glVertex(), glTranslate(), glRotate(), etc. The openGL library function models an object via a set of geometric primitives, such as point, line, and polygon.

OpenGL functions are used for creating primitives, coloring the primitives, fixing lighting system etc. Some GL functions or commands are as follows:



- `glClearColor(...);`
- `glBegin(...);`
- `glColor3f(...);`
- `glVertex3f(...);`
- `glEnd(...);`
- `glEnable(...);`
- `glDisable(...);`

## ii. **GLU or OpenGL Utility Library**

The *OpenGL Utility Library* (GLU) functions start with a prefix "glu", like, `gluPerspective()`, `gluBeginCurve()`. The GLU function is used to represent complex building primitives, like, quadric surfaces, transparency, and three dimensional projection. It contains several routines for setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. It provides several utility functions that make the interaction easier for dealing with camera set-up and high level shape descriptions. Some GLU functions or commands are as follows:

- `gluPerspective(...);`
- `gluLookAt(...);`

## iii. **GLUT or OpenGL Utility Toolkit Library**

The *OpenGL Utility Toolkit library* (GLUT) functions start with a prefix of "glut", e.g., `glutInit()`, `glutCreatewindow()`, `glutMouseFunc()`. The GLUT library provides support to interact with the Operating System, such as creating a window, interacting with input-output devices, handling key and mouse inputs; and more building models, such as sphere and torus. Some GLUT functions or commands are as follows:

```
glutInit(&argc, argv);
glutInitDisplayMode(...);
glutInitWindowPosition(...);
```

```

glutInitWindowSize(...);
glutCreateWindow(...);
glutDisplayFunc(...);
glutKeyboardFunc(...);
glutMouseFunc(...);
glutMainLoop();

```

### Constants:

OpenGL defined constants begin with GL\_ and use all capital letters and underscores to separate words.

e.g. GL\_COLOR\_BUFFER\_BIT

### Data types

Data type	Corresponding C Language	OpenGL Type Definition
8 bit integer	signed char	GLbyte
16 bit integer	short	GLshort
32 bit integer	int or long	GLint, GLsizei
32 bit float	float	GLfloat
64 bit float	double	GLdouble

## 9.3 OpenGL Program Structure

After installation of OpenGL, GLUT Library provides template code. We can run it. But, to begin with scratch, we can start the code by removing all the template code and including following header file.

```

#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>

```

The header file for OpenGL library is <GL/gl.h>. The header file for OpenGL Utility Library is <GL/glu.h>. The header file for OpenGL Utility Toolkit Library is <GL/glut.h>. gl.h and glu.h are already included in OpenGL package. But glut.h is not present by

default in the compiler in OpenGL package. We have to install it. In some version `gl.h` and `glu.h` is included in `glut.h` but in some version it may not be included.

### 9.3.1 The OpenGL program structure

The basic program structure in OpenGL is as follows

#### (i) **Configure and open window and Initialize OpenGL state**

We write the code in `main ()` to configure and open window and initialize OpenGL state. Five routines that perform necessary tasks to initialize a window are as follows:

```
glutInit(&argc, argv);  
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE);  
glutInitWindowPosition(int x, int y);  
  
glutInitWindowSize(int width, int size);  
  
glutCreateWindow(char *string);
```

The description of these functions are as follows.

- `glutInit(&argc, argv)` initializes GLUT and processes. `glutInit(&argc, argv)` should be called before any other GLUT routine.
- `glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE)`. We pass the arguments in the function for different purpose, `GLUT_RGBA` for Red, Green, Blue and Alpha color mode, `GLUT_DOUBLE` for a window with double buffering.
- `glutInitWindowPosition(int x, int y)`, specifies the screen location for the upper-left corner of the window. E.g., `glutInitWindowPosition(100, 90)` displays the output screen with the upper left corner location, 100 pixel in x axis and 90 pixel in y axis.

- `glutInitWindowSize(int width, int size)` specifies the size, in pixels, of the window. E. g., `glutInitWindowSize(640, 480)` displays a window of size  $640 \times 480$  pixel.
- `int glutCreateWindow(char *string)` creates a window with an OpenGL context. It returns a unique identifier for the new window. E. g., `glutCreateWindow("winodow1")` displays the name of the window1.

## (ii) Register input callback functions

We register the input callback functions to render, resize and input output activities. We do this in `main()`

E.g.,

```
/*Register the call back functions */
    glutDisplayFunc(render);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
```

## (iii) The callback function code

We declare the user defined function before `main()` and define after `main()`. We can declare and define this before `main()` function also.

E.g.,

```
void render(void)
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(1,0,0);
    glVertex2f(-0.5,-0.5);
    glColor3f(0,1,0);
    glVertex2f(0.5,-0.5);
    glColor3f(0,0,1);
    glVertex2f(0.0,0.5);
    glEnd();
```

```

    glFlush();
}
void keyboard( unsigned char c, int x, int y)
{
    if(c == 'a')
    {
        exit(0);
    }
}
void mouse(int button, int state, int x, int y)
{
    if(button == GLUT_RIGHT_BUTTON)
    {
        exit(0);
    }
}

```

#### (iv) Enter event processing loop

The `glutMainLoop()` makes the program goes into an infinite loop waiting for events. Until `glutMainLoop()` is not called, the window is not yet displayed. We do this in `main()`

E. g.,

```

/*The program goes into an infinite loop waiting for events
*/
glutMainLoop();

```

### 9.3.2 Code in OpenGL

Now, let us study some code in OpenGL, and then we will understand more clearly the program structure in OpenGL

#### Sample Program in OpenGL

```

#include <stdio.h>
#include <stdlib.h>

```

```
#include <GL/glut.h>
```

```
void display(void); /* function declaration*/
```

```
void reshape(int, int); /* function declaration for viewport*/
```

```
void init(void); /* function declaration*/
```

```
int main(int argc, char **argv)
```

```
/* pass the arguments in main function to initialize the glut library  
*/
```

```
{
```

```
glutInit(&argc, argv);
```

```
/* initialize the glut library, GLUT Configuration, the function that  
is present in glut library has the prefix glut */
```

```
glutInitDisplayMode(GLUT_RGB|GLUT_DOUBLE);
```

```
/* initialize display mode; arguments passed are constants, multiple  
flags can be used using | (OR) operator, RGB means Red, Green,  
and Blue color, DOUBLE means two frame buffer, front and back  
buffer, front buffer is on the screen, back buffer is in memory  
double buffer is used in animation purpose */
```

```
glutInitWindowPosition(200, 200); /* initialize position of the  
window in terms of screen coordinate pixel */
```

```
glutInitWindowSize(500, 500 ); /* set width and height of the  
window */
```

```
glutCreateWindow("window1"); /* it creates title of the window */
```

```
glutDisplayFunc(display); /* call display function */
```

```
glutReshapeFunc(reshape); /* call reshape function for viewport */
```

```
init(); /* set basic openGL states */
```

```

glutMainLoop(); /* enter GLUT event processing loop, which
interprets events and calls respective callback routines */
return 0;
}

```

```

/* function definition starts */

```

```

void init( ) /* called once to set up basic opengl state */

```

```

{

```

```

glClearColor(1.0, 1.0, 0.0); /*for background color, arguments
passed are RGB values, 0 for minimum intensity, 1 for maximum
intensity */

```

```

}

```

```

void display( ) /* display is called by the glut main loop once for
every animated frame */

```

```

{

```

```

glClear(GL_COLOR_BUFFER_BIT);

```

```

/* clear all the pixel of frame buffer to default color */

```

```

glLoadIdentity();

```

```

/* resets model view matrix or transformation matrix carried out
from previous display*/

```

```

//code to draw primitive

```

```

    glPointSize(10.0);

```

```

    glBegin(GL_POINTS);

```

```

    glVertex2f(5, -5);

```

```

    /* In 2f, 2 means two dimensional, f means float*/

```

```

    glVertex2f(-5, -5);

```

```

    glBegin(GL_TRIANGLES);

```

```

    glVertex2f(0.0, 5.0);

```

```

    glVertex2f(-4.0, 3.0);

```

```

    glVertex2f(4.0, -3.0); /*primitive color is white, specify
        vertices in anticlock wise to display front side */

    glEnd();

    // glutSwapBuffers();
    /* to swap the front buffer and back buffer*/
    glFlush(); /* to display the framebuffer in screen*/
}

void reshape(int w, int h )
{
    gl Viewport(0,0, (GLsizei)w, (GLsizei) h);
    /* set viewport inside the screen*/
    glMatrixMode(GL_PROJECTION);
    /* go to projection matrix to change the projection */
    glLoadIdentity();
    /* resets projection matrix, we go to projection matrix to change
        the projection */
    gluOrtho2D(-10, 10, -10, 10); /* left , right, bottom, top */
    glMatrixMode(GL_MODELVIEW); /* switch back to model view
        matrix, we draw the stuff in model view */
    /*ModelView matrix is the concatenation of Model matrix and
        View Matrix. View Matrix defines the position(location and
        orientation) of the camera, while model matrix defines the frame's
        position of the primitives we are going to draw.
        Projection matrix defines the characteristics of your camera, such
        as clip planes, field of view, projection method etc. */
}

```

### 9.3.3 Callback Function

Callback function is user-defined function used to react on specific event like to redraw contents, to resize window, to react on



keyboard, to handle mouse motions etc. We have to register callback function before to use it.

A callback function is a function which the library (GLUT) calls when it needs to know how to process something. E.g. when glut gets a key down event it uses the `glutKeyboardFunc()`, callback routine to find out what to do with a key press.

The `glutKeyboardFunc()` deals with events generated by keys which have an ASCII code, for instance 'a', '1', or even ' '. `glutSpecialFunc()` deals with the "special keys", like F1-F12, Home, Up, etc. GLUT supports many callback functions. Some are listed below.

## **The Display Callback function**

- `glutDisplayFunc()` is called when pixels in the window need to be refreshed. It is most important event callback function. It can be used by passing user defined function. Eg., `glutDisplayFunc(render)`. Here function `render` is registered. We write the code in `render()` function for the graphics we want to display. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by `glutDisplayFunc()` is executed. Therefore, we should put all the routines we need to redraw the scene in the display callback function.
- If our program changes the contents of the window, sometimes we will have to call `glutPostRedisplay(void)`, which gives `glutMainLoop()` a nudge to call the registered display callback at its next opportunity.
- `glutKeyboardFunc()` is registered and called for the action when any key is pressed. It allows us to link a keyboard key with a routine that's invoked when the key is pressed or released.

- `glutMouseFunc()` is called for the action when mouse button is pressed. It allows us to link a mouse button with a routine that's invoked when the key is pressed or released.
- `glutReshapeFunc()` is called when the window is resized or moved.
- `glutIdleFunc()` is used to specify a function that's to be executed if no other events are pending. This routine takes a pointer to a function as its only argument

### Events in OpenGL

Event	Example	OpenGL Callback Function
Keypress	KeyDown KeyUp	<code>glutKeyboardFunc()</code>
Mouse	leftButtonDown leftButtonUp	<code>glutMouseFunc()</code>
Motion	With mouse press Without	<code>glutMotionFunc()</code> <code>glutPassiveMotionFunc()</code>
Window	Moving Resizing	<code>glutReshapeFunc()</code>
System	Idle Timer	<code>glutIdleFunc()</code> <code>glutTimerFunc()</code>
Software	What to draw	<code>glutDisplayFunc()</code>

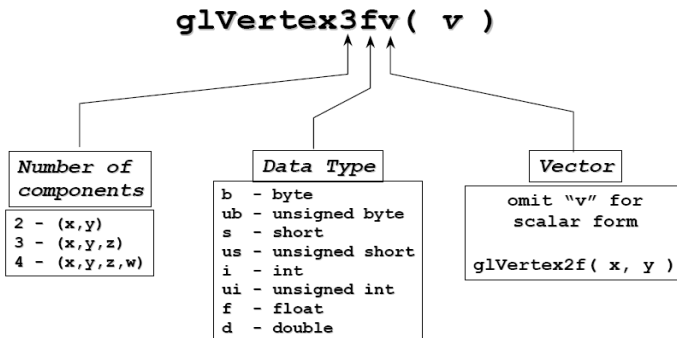
## 9.4 OpenGL Geometric Primitives

The geometry is specified by vertices. Primitives are ways that OpenGL interprets vertex streams, converting them from vertices into triangles, lines, points and so forth. There are different primitive types:

`GL_POINTS`  
`GL_LINES`  
`GL_LINES_STRIP`

GL\_LINE\_LOOP  
 GL\_TRIANGLES  
 GL\_TRIANGLE\_STRIP  
 GL\_TRIANGLE\_FAN  
 GL\_QUADS  
 GL\_QUAD\_STRIP  
 GL\_POLYGONS

## OpenGL Command Format



### 9.4.1 Vertices and Primitives

Primitives are specified between glBegin() and glEnd().  
Such as,

```

glBegin( primType );
glVertex *();
.....
glVertex *();
glEnd();

```

#### For example to draw a Parallelogram

```

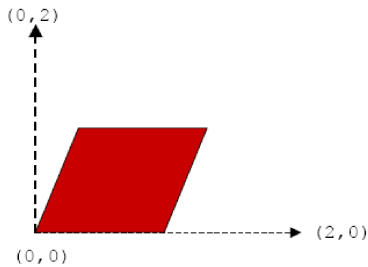
void drawParallelogram()
{
  glBegin( GL_QUADS );
  glColor3fv( 1.0, 0.0, 0.0 );

```

```

glVertex2f( 0.0, 0.0 );
glVertex2f( 1.0, 0.0 );
glVertex2f( 1.5, 1.0 );
glVertex2f( 0.5, 1.0 );
glEnd();
glFlush();
}

```



## Functions to draw different primitives

### i. GL\_POINTS

GL\_POINTS is used to draw points on screen. Point size can be altered.

```

{
    glPointSize(10.0);
    glBegin(GL_POINTS);
    glVertex2f(5, -5);
    glVertex2f(-5, -5);
    glEnd();
    glFlush();
}

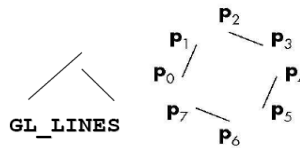
```

  
**GL\_POINTS**

### ii. GL\_LINES

It is used to draw lines on screen. In this primitive, pairs of vertices creates an individual line segment. We can specify line width using `glLineWidth(float width)`.

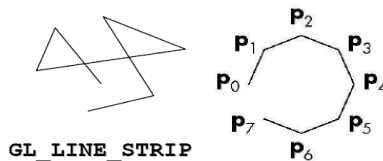
```
glBegin(GL_LINES);
glColor3fv( color );
glVertex2f( P0.x, P0.y );
glVertex2f( P1.x, P1.y );
glVertex2f( P2.x, P2.y );
glVertex2f( P3.x, P3.y );
glVertex2f( P4.x, P4.y );
glVertex2f( P5.x, P5.y );
glVertex2f( P6.x, P6.y );
glVertex2f( P7.x, P7.y );
glEnd();
```



### iii. **GL\_LINE\_STRIP**

It uses vertices defining a sequence of line segments.

Series of connected line segments are drawn using this primitive.

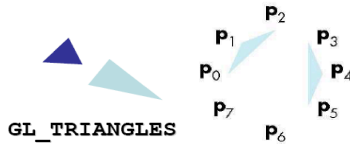


### **Lines, GL\_LINE\_LOOP**

It draws connected lines on screen. The last vertex is connected to first vertex

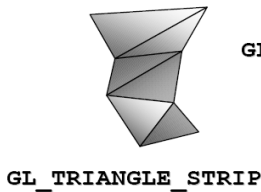
### iv. **GL\_TRIANGLES**

It is used to draw triangles. Every three vertices specified compose a triangle.



v. **GL\_TRIANGLE\_STRIP**

It draws connected triangles on screen. Every vertex specified after three vertices creates linked strip of triangles

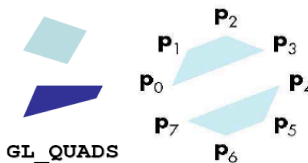


**GL\_TRIANGLE\_FAN**

It draws connected triangles in fan shape. First three vertices create a triangle and each subsequent vertex with the first vertex and the previous vertex for the next triangle.

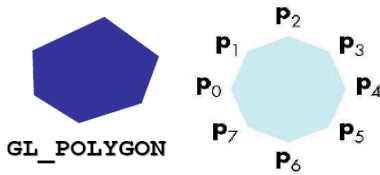
vi. **GL\_QUADS**

It draws quadrilateral on screen. Every specified four vertices create four-sided polygons



vii. **GL\_POLYGON**

It creates a polygon on screen. Polygon can be composed of any number of sides.



While writing program between glBegin and glEnd, those opengl commands are allowed:

- glVertex\*() : set vertex coordinates
- glColor\*() : set current color
- glIndex\*() : set current color index
- glNormal\*() : set normal vector coordinates (Light.)
- glTexCoord\*() : set texture coordinates (Texture)

### OpenGL Program to draw a Triangle

```
/* program to draw a triangle */
#include <stdio.h>
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
void render(void)
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glVertex2f(-0.5,-0.5);
    glVertex2f(0.5,-0.5);
    glVertex2f(0.0,0.5);
}
```

```

glEnd();
}
void keyboard( unsigned char c, int x, int y)
{
if(c == 'a')
{
exit(0);
}
}
void mouse(int button, int state, int x, int y)
{
if(button == GLUT_RIGHT_BUTTON)
{
exit(0);
}
}
int main( int argc, char* argv[])
{
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_DEPTH|GLUT_DOUBLE|
        GLUT_RGBA);
/* Create a window Named "simple GLUT Application"
with starting point (100,100) with resolution 640 x 480 */
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(640,480);
    glutCreateWindow("Simple GLUT Application");

    /* Register the call back functions */
    glutDisplayFunc(render);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);

```



```

    /* The program goes into an infinite loop waiting for events
       */
    glutMainLoop();
}

```

### OpenGL Program to draw a Polygon

```

#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{ /* clear all pixels */
  glClear (GL_COLOR_BUFFER_BIT);
  /* draw white polygon (rectangle) with corners at * (0.25,
    0.25, 0.0) and (0.75, 0.75,0.0) */
  glColor3f (1.0, 1.0, 1.0);
  glBegin(GL_POLYGON);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.25, 0.75, 0.0);
  glEnd();
  /* don't wait!  * start processing buffered OpenGL routines
     */
  glFlush ();
}

void init (void)
{ /* select clearing (background) color    */
  glClearColor (0.0, 0.0, 0.0, 0.0);
  /* initialize viewing values */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

```

```

glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0); }
/*
* Declare initial window size, position, and display mode
* (single buffer and RGBA). Open window with "hello"
* in its title bar. Call initialization routines.
* Register callback function to display graphics.
* Enter main loop and process events. */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("hello");
    init ();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0; /* ISO C requires main to return int. */
}

```

### Sample Code for animation in OpenGL

```

#include<GL/gl.h>
#include<GL/glu.h>
#include<GL/glut.h>
void timer(int);
int main(int argc, char **argv)
{
    glutInitdisplayMode(GLUT_RGB | GLUT_DOUBLE);
    glutTimerFunc(1000, timer, 0);

    /* arguments are time in millisecond and function you need to call
    and integer*/

```

```

}

float x_position = -10.0

void display( ) /* define function*/
{
    glClear(GL_COLOR_BUFFER_BIT );
    glLoadIdentity();
    glBegin(GL_POLYGON);
    glVertex2f(x_position, 1.0);
    glVertex2f(x_position, -1.0);
    glVertex2f(x_position+2, -1.0);
    glVertex2f(x_position+2, 1.0);

    glEnd();
    glutSwapBuffers();
}

```

```

Void timer(int)
{
    glutPostRedisplay();
    glutTimerFunc(1000/60, timer, 0);
    switch(state)
    {
        case 1:
            if(x_position<8)
                x_position+=0.15;
            else
                state = -1;
            break;
        case -1:
            if(x_position>-10)

```

```

        x_postion-=0.15;
    else
        state= 1;
        break;
    }
}

```

## 9.5 Color Command

OpenGL supports both RGBA (true color) mode and color index (color map) mode. In general, an OpenGL programmer first sets the color or coloring scheme and then draws the objects. Until the color or coloring scheme is changed, all objects are drawn in that color or using that coloring scheme.

### RGBA mode

In RGBA mode, color is specified by three intensities for Red, Green and Blue components of the color and the fourth optional value of Alpha which controls transparency. To set a color, use the command `glColor3f()` or `glColor4f()`. `glColor3f()` takes three parameters, all of which are floating-point numbers between 0.0 and 1.0. The parameters are, in order, the red, green, and blue components of the color. We can think of these three values as specifying a "mix" of colors: 0.0 means don't use any of that component, and 1.0 means use all you can of that component. Thus, the code `glColor3f(1.0, 0.0, 0.0);` makes the brightest red the system can draw, with no green or blue components. All zeros makes black; in contrast, all ones makes white. Setting all three components to 0.5 yields gray (halfway between black and white). Here are eight commands and the colors they would set.

<code>glColor3f(0.0, 0.0, 0.0);</code>	black
<code>glColor3f(1.0, 0.0, 0.0);</code>	red
<code>glColor3f(0.0, 1.0, 0.0);</code>	green
<code>glColor3f(1.0, 1.0, 0.0);</code>	yellow

<code>glColor3f(0.0, 0.0, 1.0);</code>	blue
<code>glColor3f(1.0, 0.0, 1.0);</code>	magenta
<code>glColor3f(0.0, 1.0, 1.0);</code>	cyan
<code>glColor3f(1.0, 1.0, 1.0);</code>	white

`glClearColor()` is used to specify the initial background color. It takes four parameters Red, Green, Blue and Alpha value.

In color index mode OpenGL uses color map(look up). It does not support transparency.

### Sample Code for Color in OpenGL

```
// default background color is black
// default color of primitive is white
//value of state variable remain same unless it is not changed
void init()
{
    glClearColor(0.0, 0.0, 0.0); /*to set background color */
    /* color of primitive can also be assigned here by using
    glColor3f() function but it is better to use in display function */
}

void display( ) /* define function*/
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glShadeModel(GL_FLAT); /* different shading model can
    be used */
    glBegin(GL_POLYGON);
    glColor3f(1.0,0.0,0.0); /* to set the color of vertex of the
    primitive, to enter four arguent that is alpha too we have change in
```

glutInitDisplayMode too, the following corner will be displayed in red color \*/

```
glVertex2f(-1.0, 1.0);
glColor3f(0.0,1.0,0.0);
/* following corner will be displayed in green color */

glVertex2f(-1.0, -1.0);
glColor3f(0.0,0.0,1.0);
/* following two corners will be displayed in blue color */

glVertex2f(1.0, -1.0);
glVertex2f(1.0, 1.0);

glEnd();
glutFlush();
}
```

## 9.6 OpenGL viewing

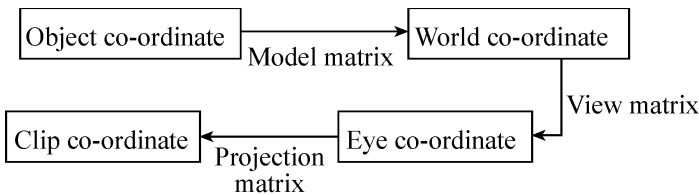
In OpenGL, viewing and modeling transformation is combined in single modelview matrix. Modelview matrix is the concatenation of Model matrix and View Matrix. Model matrix defines the frame's position of the primitives and View Matrix defines the position (location and orientation) of the camera.

The default matrix is modelview matrix. The creation and transformations of models are done in modelview matrix. `glMatrixMode()` is used to change the current matrix, modelview to projection matrix and viceversa. The argument is passed in `glMatrixMode()` to switch to the matrix. If we want to switch to projection matrix then we write `glMatrixMode(GL_PROJECTION)`. And, we reset the projection matrix writing `glLoadIdentity()`. As we write `glLoadIdentity()` in

ModelView matrix to reset the ModelView matrix, we use `glLoadIdentity()` after `glMatrixMode(GL_PROJECTION)` to reset the parameters of the projection matrix. To specify the orthographic projection in two dimension we write `gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)`. We switch to projection matrix only if we need to change the projection. We always draw models in modelview matrix. We come back to modelview matrix writing `glMatrixMode(GL_MODELVIEW)`

Projection matrix defines the characteristics of the camera, such as clip planes, field of view, projection method etc.

In OpenGL, the modelview matrix is used to position the camera (i.e. viewing), it can be done by rotations and translation but is easier to use `gluLookAt ()`. The modelview matrix is also used to build models of objects. The projection matrix is used to define the view volume and to select a camera lens.



The model matrix transforms a position in a model to the position in the world. This position is affected by the position, scale and rotation of the model that is being drawn. If the vertices in world coordinates is specified already, then this matrix can simply be set to the identity matrix.

In real life to alter the view of a certain scene we can move camera but the camera in OpenGL cannot move and is defined to be located at (0, 0, 0) facing the negative Z direction. That means that instead of moving and rotating the camera, the world is moved and rotated around the camera to construct the appropriate view.

After the world has been aligned with our camera using the view transformation, the projection transformation can be applied, resulting in the clip coordinates.

## Transformation command

### **glMatrixMode()**

It specifies the model view, projection or texture matrix modification passing the argument GL\_MODELVIEW, GL\_PROJECTION, GL\_TEXTURE. Only one matrix can be modified at a time

### **glLoadIdentity (),**

- It sets the currently modifiable matrix to identity matrix.

## Viewing transformation

- gluLookAt (eye.x, eye.y, eye.z, center.x, center.y, center.z, up.x, up.y, up.z)
- The argument of this command indicates where the camera (or eye position) is placed, where it is aimed and which way is up.
- Viewing direction is center to eye.
- Up is the upward direction.
- Viewing direction and up vector indicates eye co-ordinate system.
  - x-axis points to the right of viewer.
  - y-axis points to upward.
  - z-axis points to the back of viewer.

## Viewport transformation

- The viewport transformation in OpenGL is controlled by function glviewport (GLint x, GLint y, GLsizei width, GLsizei height)
- It is used to set the size and position of the viewport.
- The (x, y) defines the lower left corner of the viewport and width and height are the size of the viewport rectangle.

## The Viewing Transformation example

The viewing transformation is analogous to positioning and aiming a camera. In this code example, before the viewing transformation can be specified, the current matrix is set to the



identity matrix with `glLoadIdentity()`. This step is necessary since most of the transformation commands multiply the current matrix by the specified matrix and then set the result to be the current matrix. If we don't clear the current matrix by loading it with the identity matrix, we continue to combine previous transformation matrices with the new one we supply.

In the following example, after the matrix is initialized, the viewing transformation is specified with `gluLookAt()`. The arguments for this command indicate where the camera (or eye position) is placed, where it is aimed, and which way is up. The arguments used here place the camera at (0, 0, 5), aim the camera lens towards (0, 0, 0), and specify the up-vector as (0, 1, 0). The up-vector defines a unique orientation for the camera.

If `gluLookAt()` was not called, the camera has a default position and orientation. By default, the camera is situated at the origin, points down the negative z-axis, and has an up-vector of (0, 1, 0). So in this example, the overall effect is that `gluLookAt()` moves the camera 5 units along the z-axis.

## Transformation of object

Transformation of object can be done by using transformation function in OpenGL. We can transform an object by using following functions.

i. Translation

`glTranslatef(tx, ty, tz),`

where tx, ty, tz are translation co-ordinate

ii. Rotation

`glRotatef(A, x, y, z),`

where A is angle of rotation

iii. Scaling

`glScalef(float x, float y, float z)`

## Sample Code for transformation in OpenGL

```

void display( ) /* define function*/
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    /* resets all the matrix of transformations or resets the
    coordinate system, if it is not used the translation will be
    accumulated */

    glTranslatef(5.0, 5.0, 0.0); //shifting factor in x,y,z direction
    glBegin(GL_POLYGON);
    glColor3f(1.0,0.0,0.0);

    /* to set the color of vertex of the primitive, to enter four
    argument that is alpha too we have change in glutInitDisplayMode
    too, the following corner will be displayed in red color */

    glVertex2f(-1.0, 1.0);
    glColor3f(0.0,1.0,0.0);
    /* following corner will be displayed in green color */

    glVertex2f(-1.0, -1.0);
    glColor3f(0.0,0.0,1.0);
    /* following two corners will be displayed in bluecolor */

    glVertex2f(1.0, -1.0);
    glVertex2f(1.0, 1.0);

    glEnd();
    glutFlush();
}

```

## OpenGL code to draw a Transformed Cube

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

void init(void)
{   glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
}

void display(void)
{   glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glLoadIdentity ();          /* clear the matrix */
    /* viewing transformation */
    gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
    glScalef (1.0, 2.0, 1.0);    /* modeling transformation */
    glutWireCube (1.0);
    glFlush ();
}

void reshape (int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
    glMatrixMode (GL_MODELVIEW);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
```

```

glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}

```

## 9.7 Lighting in OpenGL

Lighting provides realistic view of objects by simulating the light intensity reflected by the object. OpenGL divides the lighting process in various parts. They are as follows

- Enabling lighting and light sources
- Creating and positioning light sources
- Specifying material properties
- Specifying normal vectors
- Specifying properties of the lighting model

### Enabling lighting and light sources

To describe light in an OpenGL application, first, we have to enable the lighting system by using `glEnable (GL_LIGHTING)` and then configure at least one source of light by using `glEnable(GL_LIGHTn)`. Where `GL_LIGHTn` is one of the constants `GL_LIGHT0`, `GL_LIGHT1`,..... `GL_LIGHT7`.

The function to enable lighting in general is,

- `glEnable(GL_LIGHTING)`

The function to enable specific lights is,

- `glEnable(GL_LIGHTn)`

## Creating and Positioning Light Sources

### Creating Light Sources

OpenGL supports eight light sources, which are identified by the constants `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`. The light source is created by using `glLightfv()`. Three parameters are set: first, light source name; second, its property like diffuse, specular, ambient; and third, color of the light. Before they are enabled only `GL_LIGHT0` is enabled by default. Light properties can be set as follows,

```
void glLightfv( int light, int property, float* valueArray );
```

Among the parameters in the above function, the first parameter represents `GL_LIGHT0`, `GL_LIGHT1`, ..., `GL_LIGHT7`. It specifies which light is being configured.

The second parameter denotes which property of the light is being set. It can be `GL_DIFFUSE`, `GL_SPECULAR`, `GL_AMBIENT`, or `GL_POSITION`.

The third parameter specifies the red, green, blue, and alpha components of the color. It is written in an array.

```
float blue1[4] = { 0.4, 0.4, 0.6, 1 };
```

```
glLightfv( GL_LIGHT1, GL_DIFFUSE, blue1 );
```

### Positioning Light Sources

The `GL_POSITION` property is used to both to set whether the light is a point light or a directional light, and to set its position or direction. The value for `GL_POSITION` is an array of four numbers ( $x, y, z, w$ ). For directional light source, the value of  $w$  is zero and the point ( $x, y, z$ ) specifies the direction of the light. The light rays shine in the direction of the line from the point ( $x, y, z$ ) towards the origin. For point light source, the value of  $w$  is non-zero, and it is located at the point ( $x/w, y/w, z/w$ ). Usually,  $w$  is 1. The value ( $x, y, z, 1$ ) gives a point light at ( $x, y, z$ ).

The position that is specified for a light is transformed by the modelview transformation using `glLightfv`.

```
float position[4] = { 1, 2, 3, 1 };
```

*glLightfv(GL\_LIGHT1, GL\_POSITION, position);* puts the light in the same place as

*glTranslatef(1,2,3);*

*float position[4] = { 0,0,0,1 }*

*glLightfv(GL\_LIGHT1, GL\_POSITION, position);*

## Specifying Material Properties

The function for setting the values of material properties is *glMaterialfv()*. It includes three parameters: first, side, i. e., front or back; second, property like ambient, diffuse, etc.; third, color intensity. It is written as follows,

*void glMaterialfv( int side, int property, float\* valueArray )*

The first parameter side may be GL\_FRONT, GL\_BACK or GL\_FRONT\_AND\_BACK. It represents the material property value for the front face, the back face, or both.

The front face and back face of a polygon can have different materials. Two sets of material property values are stored for each vertex: the front material and the back material. The back material isn't actually used unless we turn on two-sided lighting.

The second parameter tells which material property is being set. It is GL\_AMBIENT, GL\_DIFFUSE, GL\_SPECULAR, GL\_EMISSION, or GL\_AMBIENT\_AND\_DIFFUSE. It is possible to set the ambient and diffuse colors to the same value with one call to *glMaterialfv* by using GL\_AMBIENT\_AND\_DIFFUSE as the property name.

The last parameter of *glMaterialfv* is an array containing four float number for the RGBA color components. It ranges from 0.0 to 1.0.

Suppose that we want to set the ambient and diffuse colors to a bluish green. In C, we can write,

*float bgcolor[4] = { 0.0, 0.7, 0.5, 1.0 };*

*glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT\_AND\_DIFFUSE, bgcolor );*

The shininess material property is a single number rather than an array. And the value is a float in the range 0.0 to 128.0.

*void glMaterialf( int side, int property, float value )*

In C, the third parameter can be set as follows,

```
float gold[13] = { 0.24725, 0.1995, 0.0745, 1.0,    /* ambient */
                  0.75164, 0.60648, 0.22648, 1.0,    /* diffuse */
                  0.628281, 0.555802, 0.366065, 1.0, /* specular */
                  50.0                                /* shininess */
                };
```

where the first four numbers in the array specify an ambient color; the next four, a diffuse color; the next four, a specular color; and the last number, a shininess exponent. This array can be used to set all the material properties:

```
glMaterialfv( GL_FRONT_AND_BACK, GL_DIFFUSE, &gold[4] );
glMaterialf( GL_FRONT_AND_BACK, GL_SHININESS, gold[12] );
```

`glColor3f( 1, 0, 0 )` has the same effect as calling `glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, array)`. Where array contains the values 1, 0, 0, 1.

## Specifying Normal Vectors

The function of normal of a vertex is written as `glNormal()`. All normal vectors have three components. The normal vector for a vertex is specified before `glVertex()`. E. g.;

```
glNormal3f( 0.0, 0.0, 1.0 ); // (This is the default value.)
glNormal3d( 0.707, 0.707, 0.0 );
float normalArray[3] = { 0.577, 0.577, 0.577 };
glNormal3fv( normalArray );
```

A "v" means values in an array, "f" means values floats, and "d" means the values are doubles.

For a polygon that is supposed to look flat, the same normal vector is used for all of the vertices of the polygon. For example, to draw one side of a cube, say the "top" side, facing in the direction of the positive *y*-axis:

```
glNormal3f( 0, 1, 0 ); // Points along positive y-axis
glBegin(GL_QUADS);
```

```
glVertex3fv(1,1,1);
glVertex3fv(1,1,-1);
glVertex3fv(-1,1,-1);
glVertex3fv(-1,1,1);
glEnd();
```

## Specifying the Lighting Model

In addition to the properties of individual light sources, the OpenGL lighting system uses three global properties: the global ambient light, the number of lighting side, the direction from a surface to the viewer.

By default, the global ambient light is black. The value can be changed using the function

```
void glLightModelfv( int property, float* value )
```

Where the *property* is `GL_LIGHT_MODEL_AMBIENT` and the *value* is an array of RGBA values of the global ambient light. For example, in C:

```
float ambientLevel[] = { 0.15, 0.15, 0.15, 1 };
glLightModelfv( GL_LIGHT_MODEL_AMBIENT, ambientLevel );
```

The two properties of light model; the number of lighting side, and the direction from a surface are optional. A value equal to 0 indicates off and 1 indicates on. The symbolic constants `GL_FALSE` and `GL_TRUE` are also used for the value.

The number of lighting side property `GL_LIGHT_MODEL_TWO_SIDE` can be set using the following function,

```
void glLightModeli( int property, int value )
```

`GL_LIGHT_MODEL_TWO_SIDE` is used to turn on two-sided lighting. A polygon can have two sets of material properties, a front material and a back material. When two-sided lighting is off, which is the default, only the front material is used.

When two-sided lighting is on, the back material is used on the back face and the direction of the normal vector is reversed when it is used in lighting calculations for the back face.



The direction from a surface to the viewer in the lighting equation property `GL_LIGHT_MODEL_LOCAL_VIEWER` can set using the function, *void glLightModeli( int property, int value)*. But it is of less importance. By default, this direction is always taken to point directly out of the screen, which is true for an orthographic projection but is not accurate for a perspective projection.

After setting up light model the next step is setting up the shading model. This is done by calling `glShadeModel`.

```
glShadeModel(GL_SMOOTH);
```

Here, smooth shading model is used. The `SMOOTH` shading model specifies the use of Gouraud-shaded polygons to describe light while the `FLAT` shading model specifies the use of single-colored polygons.



## **BIBLIOGRAPHY:**

1. Donald D. Hearn and M. Pauline Baker, “Computer Graphics”, (Second Edition)
2. Foley, Van Dam, Feiner, Hughes “Computer Graphics Principles and Practice”,(Second Edition in C)
3. Udit Agrawal, “COMPUTER GRAPHICS”, (Fourth Edition)
4. Rajiv Chopra, “COMPUTER GRAPHICS (A PRACTICAL APPROACH)”, (Third Edition)
5. Neeta Nain, “COMPUTER GRAPHICS”, (First Edition)
6. G. S. Baluja, “COMPUTER GRAPHICS”, (Revised Edition 2008)