# 302-COMPUTER GRAPHICS

**(MCA-S.V.UNIVERSITY,TIRUPATI)**

**III-SEMESTER**

**STUDY MATERIAL**

**PREPARED BY: MISS.C.YAMINI M.C.A,**

**Department of Computer Science,**

**KMM INSTITUTE OF POST GRADUATION STUDIES.**


**KMM INSTITUTE OF POST GRADUATION STUDIES**

**Ramireddipalle, TIRUPATI-517102**

**Unit-1**

**COMPUTER GRAPHICS**

**1.Computer Graphics:** is a mechanism that performs functions like Rotating, Scaling and Transforming on an image or picture.

CG is the discipline of producing picture or images using a computer which include modeling, creation, manipulation, storage of geometric objects, rendering, converting a scene to an image, the process of transformations, rasterization, shading, illumination, animation of the image, etc.

CG is used on different areas as science, engineering, medicine, business, entertainment, advertise, industry, art, education and training.

**2.Applications of Computer Graphics**

**CAD (Computer Aided Design)**

- A major use of CG is in design processes, for engineering and architectural system.
- CAD methods are now used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles and many other products.
- In some applications, objects are first displayed in a wireframe outline form which shows the overall shape and internal features of objects.
- By using wireframe displays, we can understand easily and we can change the shapes very fastly.
- Software packages for CAD applications provide multi-window environment for designer.
- By using multiple windows we can show the objects in different views and in various sizes.
- We can design circuits and networks by using few graphical shapes.
- The standard shapes for electrical, electronic and logic circuits are supplied by design packages.
- For other applications, a designer can create symbols.
- Graphics package will provide a connection between components after placing into the layout.
- **Animations** are also used in CAD applications.
- In real-time animations the wireframe displays are used to test the performance of an object (vehicle or system)
- Wireframe displays allow the designer to see into the interior of the vehicle and to watch the behavior of inner components during motion.
- In virtual-reality environments, animations are used to determine how vehicle operators are affected by certain motions.
- Architects use interactive graphics methods to layout floor plans such as positioning of rooms, doors, windows, etc.,

**Presentation Graphics**

- It is used to produce illustrations for reports or to generate 35-mm slides or transparencies for use with projectors.
- It is commonly used to summarize financial, statistical, mathematical, scientific and economic data for research report, managerial reports and other types of reports.
- Examples, bar charts, line graphs, surface graphs, pie-charts and other displays showing relationships between multiple parameters.

**Computer Art**

- CG is widely used in fine and commercial art applications
- For designing object shapes and specifying object motions, artists use a variety of computer methods, paintbrush programs (Lumena), paint packages (Pixel paint, super paint), specially developed software, symbolic mathematics packages, CAD packages, animation packages etc.

- Paintbrush program allows artists to paint pictures on the screen of a video monitor. Usually it is painted electronically on a graphics tablet using styles.
- The stylus (like pen) translates changing hand pressure into variable line widths, brush sizes and color gradations.
- Fine artists use a variety of other computer technologies to produce images. They also use a combination of 3-d modeling packages, texture mapping, drawing programs and CAD Software.

**Entertainment**

- Computer graphics methods are now commonly used in making motion pictures, music videos, and television shows.
- The graphics scenes are displayed by themselves only, and sometimes objects are combined with the actors and live scenes.
- Videos use graphics in several ways.
    - Graphics objects can be combined with the live action
    - Graphics and image processing techniques can be used to produce a transformation of one person or object into another (morphing).

**Education And Training**

- In the Education the computer Graphics are used to demonstrate the various computer models of physical, financial and economic systems.
- For some training applications, special systems are designed. Examples of such systems are training of
    - Ship captains
    - Aircrafts pilot
    - Heavy equipment operators
    - Air traffic control personnel

**Image Processing**

- In computer graphics, a computer is used to create a picture. Image processing, on the other hand applies techniques to modify or interpret existing pictures, such as photographs and TV scans.
- The following are the main applications of image processing:
    - improving picture quality
    - Machine observation of visual information, as used in robotics.
- To apply image processing methods, we first make the picture into an image file. Then methods can be applied to rearrange picture parts, to enhance color separations, or to improve the quality of shading.
- Image processing is also used in medical applications, to generate x-ray photography.
- Image processing and computer graphics are combined in many applications.
- In medicine these techniques are used to model and study physical functions, to design artificial limbs, and to plan and practice surgery.

**Graphical User Interfaces**

- All the software packages are now with the graphical interface.
- A major component is a window manager that allows a user to display multiple windows.
- Each window will contain different processes and display with graphical and nongraphical.
- To active a window, just click on the window with the interactive pointing device.
- Interfaces also display menus and icons for fast selection of processing options or parameter values.
- An icon is a graphical symbol that is designed to look like the processing option it represents. Advantage is we can easily understand.
- Menus contain lists of textual descriptions and icons.

In this chapter we are going to learn the basic features of graphics hardware components and graphics software packages.

3.**Video Display Devices:**

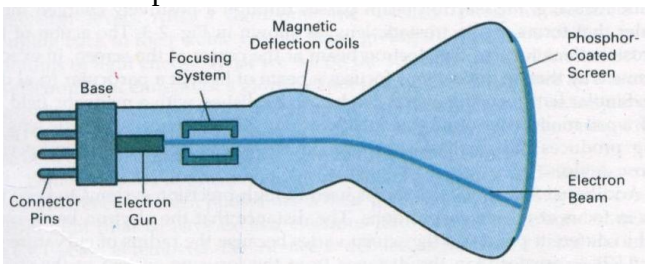
Different types of display devices:

1. Raster or Refresh Scan Displays
2. Random Scan or Calligraphic Displays
3. Color CRT Monitors
4. Direct View Storage Tube (DVST).
5. Flat panel Displays
6. Three Dimensional Viewing Devices
7. Stereoscopic and Virtual Reality System

- The primary output device in a graphics system is a video monitor
- The operation of most video monitors is based on the standard Cathode Ray Tube (CRT).
- Now –a -days we have other technologies and solid –state monitors may dominate.

➢ **1.CRT**

- A cathode ray tube (CRT) is a specialized vacuum tube in which images are produced when an electron beam strikes a phosphorescent surface.
- Most desktop computer displays make use of CRTs.
- The CRT in a computer display is similar to the "picture tube" in a television receiver.
- The basic operation of a CRT is shown below



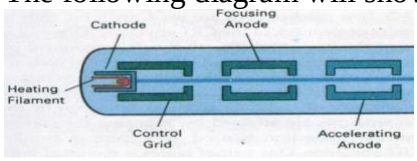

- All CRT's have three main elements: an electron gun, a deflection system, and a screen.
  - The electron gun provides an electron beam
  - Beam passes through Focusing system and Deflection coils to direct it to a specified position on the phosphor coated screen.
  - The screen displays a small spot of light at the point where the electron beam strikes it.
  - The light emitted from the phosphor depends on the number of electrons striking it.

➢ **Refresh CRT:**

- A beam of electrons (cathode rays), emitted by an electron gun.
- These rays will passes through focusing system and deflection systems that through the bean toward specified positions on the screen.
- The screen then emits a small spot of light at each position contacted by the electron beam.
- After the emitting of the light spots on the phosphor screen, it begin to fade (decrease intensity).
- To avoid flicker, we need to redraw the picture again.
- This type of display is called a refresh CRT.

➢ **Electron Gun:**

- The primary components of an electron gun in a CRT are the heated metal cathode and a control grid.
- Heat is supplied to the cathode by giving a current through a coil of wire, called the filament.
- After the cathode is heated, electrons are generated
- The following diagram will shows the electron gun




➢ **Control Grid:**

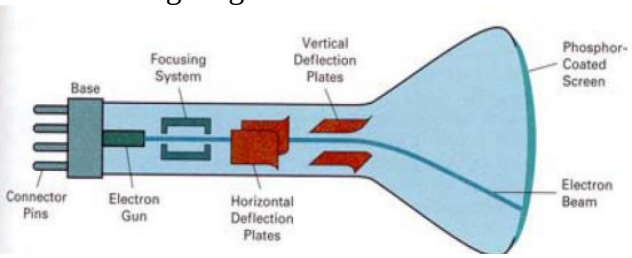
- Control grid controls the intensity or illumination (bright or dark) of a point on the screen.

- Control grid is negatively charged, and the electrons are also negative charge. So electrons will repel from the control grid, and pass through the control grid to strike the phosphor screen
- Amount of voltage on the control grid controls the intensity.
  - High voltage reduces the number of electrons to pass through the control grid. This means low intensity on the screen.
  - Low voltage increases the number of electrons to pass through the control grid. This means high intensity on the screen.

➢ **Focusing Anode:**
- It is positively charged.
- The focusing is used to create a clear picture by focusing the electrons into a narrow beam.
- Focusing is accomplished with either electric or magnetic fields.
- In TV, CG Monitors we use electrostatic focusing.

➢ **Accelerating Anode:**
- It is positively charge.
- It will accelerate the electrons and leave them to strike the screen.

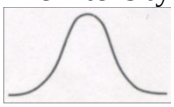
➢ **Deflection Coils**
- Control horizontal and vertical movement of the beam to cover entire screen.
- It can be implemented by either electric or magnetic fields.
- The following diagram will shows the deflection of the electron beam in a CRT




- It is mounted on top and bottom of the CRT and the two opposite sides (front and back) of the CRT

➢ **Persistence:**
- It is defined as "The time it takes for an emitted light to decay (decrease) to one-tenth of its original intensity".

➢ **Intensity Distribution:**
- The intensity is greatest at the center of the spot and decreases at the edges of the spot.

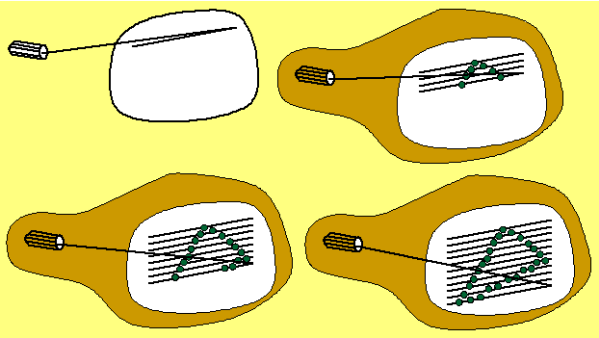


➢ **Resolution:**
- Resolution is defined as the maximum number of points that can be displayed without overlap on the CRT.
- Resolution is the number of points per centimeters to be plotted horizontally and vertically.
- Resolution of a CRT depends on:
  - Type of phosphor.
  - The intensity to be applied.
  - The focusing and deflection systems.

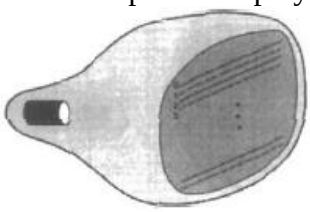
➢ **Aspect Ratio**
- Aspect ratio is the number of vertical points to the number of horizontal points necessary to produce lines in both directions of the screen of equal length.
- Aspect ratio of 4/3 means that:
  - A horizontal line with length 4-points is equal to a vertical line with length 3-point.

2.**Raster Scan Displays**
- **Raster:** A rectangular array of points or dots
- **Pixel:** One dot or picture element of the raster

- **Scan Line:** A row of pixels
- In a raster scan system, the electron beam is swept (cleared) across the screen, one row at a time from top to bottom.
- As the electron beam moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- Picture definition is stored in a memory area called the **refresh buffer** or **frame buffer**.
- **Refresh buffer** or **frame buffer:** This memory area holds the set of intensity values for all the screen points.
- Stored intensity values then retrieved from refresh buffer and "**painted**" on the screen one row (**scan line**) at a time.

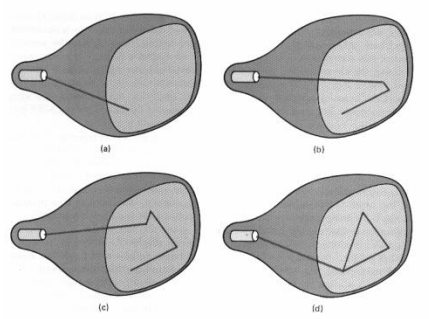


- Intensity range for pixel positions depends on the capability of the raster system.
- A **black-and-white** system: each screen point is either on or off, so only **one bit** per pixel is needed to control the intensity of screen positions.
- On a black-and-white system with **one bit** per pixel, the frame buffer is called **bitmap**.
- For system with **multiple bits per** pixel, the frame buffer is called **pixmap**.
- Sometimes, refresh rates are described in unit of cycles per second, or Hertz (**HZ**).
- Refreshing on raster scan displays is carried out at the rate 60 to 80 frame per second.
- At the end of each scan line, the electron beam returns to the left side of the screen to begin displaying the next scan line.
- The return to the left of the screen, after refreshing each scan line is called **Horizontal retrace**.
- At the end of each frame (displayed in $1/80^{th}$ to $1/60^{th}$ of a second) the electron beam returns to the top left corner of the screen to begin the next frame is called **Vertical retrace**
- In some raster scan systems, each frame is displayed in two passes using an interlaced refresh procedure.
- In the first pass it display all even scale lines and in the second pass it display remaining odd scale lines.



**3.Random Scan Displays:**

- Random scan display is the use of geometrical primitives such as points, lines, curves, and polygons, which are all based upon **mathematical equation**.
- Raster Scan is the representation of images as a collection of **pixels** (dots)
- In a random scan display, a CRT has the electron beam directed only to the **parts of the screen** where a picture is to be drawn.
- Random scan monitors draw a picture one line at a time **(Vector** display, **Stroke –writing** displays).
- The component lines of a picture can be drawn and refreshed. The following diagram will shows random scan system



- **Refresh rate** depends on the number of lines to be displayed.
- Picture definition is now stored as a line-drawing commands an area of memory referred to as **refresh display file** (**display list**).
- Random scan displays are designed for **line-drawing applications.**
- Random scan displays have higher resolution than raster systems.
- Vector displays product smooth line drawing.

**4.Color CRT Monitors**

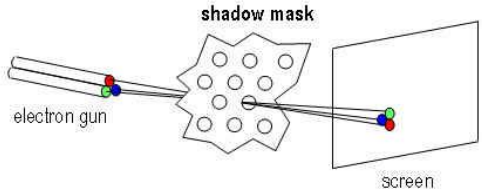
- A CRT monitor displays color pictures by using a combination of phosphors that emit different ***color*** lights.
- The two basic techniques for producing color displays with a CRT are
    i. Beam-penetration method
    ii. Shadow-mask method

**Beam-penetration method**

- Two layers of ***phosphor*** (**red** and **green**) are coated onto the inside of the CRT screen.
- The display color depends on how far the electron beam **penetrates** into the phosphor layers.
- The speed of the electrons, and the screen color at any point, is controlled by the beam **acceleration voltage.**
- It is used with random scan monitors
- At intermediate beam speeds, combinations of red and green light are emitted to show two additional colors, orange and yellow.
- Quality of pictures is not as good as with other methods but it is less expensive.

**Shadow Mask Method**

- This CRT has **three** color ***phosphor*** dots (**red**, **green** and **blue**) at each point on the screen
- This type of CRT has three ***electron guns***, one for each color dot.
- It is used with raster scan system
- The following diagram will shows a shadow mask grid




- The colors we can see depend on the amount of intensity of red, green and blue phosphor. A white area is a result of all three dots with equal intensity while yellow is produced with green and red dots and so on.
- The advantage of this method is it produce the other colors and also shadow scenes
- The disadvantage is it is expensive.
- Color CRTs in graphics systems are designed as RGB monitors.
- These monitors use shadow mask method and take the intensity level for each gun. A RGB color system with 34 bits of storage per pixel is known as **full color system** or **true color system**.

5.**Flat Panel Displays**

- It refers to a class of video devices that have reduced volume and weight compared to a CRT.

- A significant feature of flat panel displays is that they are **thinner** than CRTs.
- We use this type of displays in Small TV's, Laptops, calculator, pocket video game, etc
- The flat panel displays is of two types
    i. Emitters or Emissive Display
    ii. Nonemitters or Nonemissive Display

**Emitters Display:**

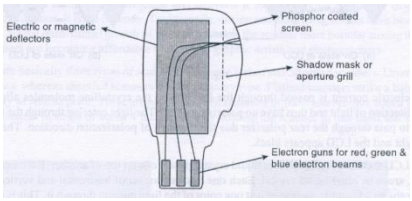
- This displays convert electrical energy into light.
- Examples: Plasma panel, Light-Emitting Diodes (LED) and flat CRT.

**Nonemitters Display:**

- Use **optical effects** to convert sunlight or light from some other source into graphics pattern.
- Example: **L**iquid-**C**rystal **D**evice (**LCD**)

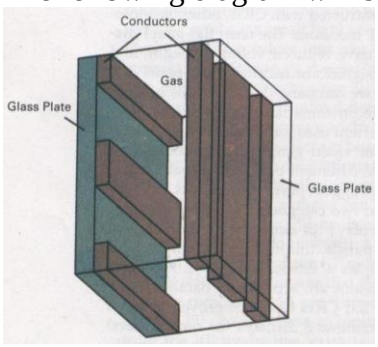
**Flat CRT**

- Electron beams are accelerated parallel to the screen, and then deflected 90° to the screen.




**6.Plasma Panel**

- The plasma panel is composed of two plates of glass with a series of beams filled with color phosphors in between.
- The following diagram will shows a plasma panel display design




- By applying high voltage to a pair of horizontal and vertical conductors, a small section of the gas (tiny neon bulb) at the intersection of the conductors break down into glowing plasma of electrons and ions.

**7.Light Emitting Diode (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 refreshed buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.

**8.Liquid Crystal Displays (LCD)**

- Used in small systems, such as calculators, laptop computers.
- Produce a picture by passing polarized light (from the surrounding or from an internal light source) through a liquid-crystal material that can be aligned to either block or transmit the light.

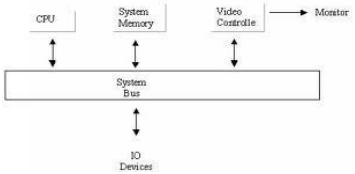
**Three Dimensional Viewing Devices**

- For the display of 3D scenes
- Reflects a CRT image from a vibrating, flexible mirror
- Allows to walk around an object or scene and view it from different sides

❖ **Raster Scan System**

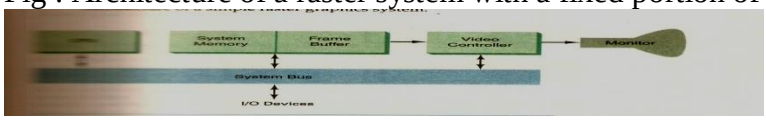
- Interactive raster-graphics systems typically employ several processing units.

- In addition to the CPU, a special purpose processor called the video controller or display controller is used to control the operation of the display device.
- The Architecture of a simple raster graphics system is shown below




- The frame buffer can be anywhere in the system memory, and the video controller access the frame buffer to refresh the screen.

- **Video Controller**
  - A fixed area of the system memory is reserved for the frame buffer.
  - The video controller is given direct access to the frame buffer memory.
  - Fig : Architecture of a raster system with a fixed portion of the system reserved for frame buffer

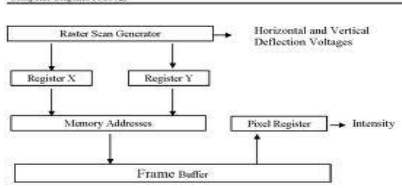



  - Screen positions in frame buffer locations are referenced in Cartesian coordinates
  - The following figure shows the origin of coordinate system for identifying screen positions is usually specified in the lower left corner

  - The screen surface is then represented as the first quadrant of a two-dimensional system.
  - Positive x values increasing to the right and positive y values increasing from bottom to top.
  - Scan lines are then labeled from $Y_{max}$, at the top of the screen to 0 at the bottom.
  - Along each scan line, screen pixel positions are labeled from 0 to $X_{max}$.

**Basic refresh operations**

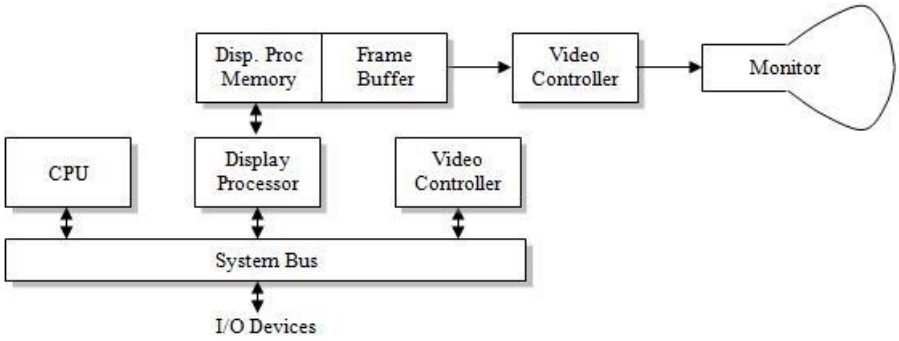
- The following diagram will shows the basic refresh operations of the video controller



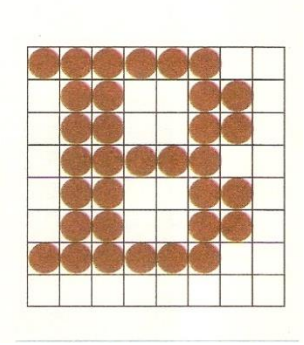

  - 2 registers are used to store the coordinates of the screen pixels (xreg,yreg)
  - Initially xreg is set to zero & yreg is set ymax
  - Value from frame buffer for this pixel position is retrieved & used to set the intensity value of the CRT beam
  - xreg is incremented by 1 & process repeated for each pixel along the scan line
  - Then xreg is reset to zero & yreg is decremented by one
  - The process repeated till y=0, x=xmax
  - Video controller resets the register to the first pixel position on top scan line & the refresh process starts over
  - Refreshing is 60 frames per second
  - Cycle is too slow
  - To speed up pixel processing, video controllers can retrieve multiple pixel values from refresh buffer on each pass
  - Multiple pixel intensities are then stored in pixel register
  - In high quality systems, 2 frame buffers are used
    - 1 – refreshing
    - 2- filling intensity values

- Fast mechanism for generating real-time animations

- **Display Processor**
  - Display processor or graphics controller or display coprocessor
  - Display processor is used to free the CPU from the graphics tasks
  - In addition to the system memory, a separate display processor memory area can be provided as shown below




  - Display processor digitize a picture definition into a set of pixel intensity values for storage in frame buffer
  - This digitization process is called scan conversion
  - Lines and other geometric objects are converted into set of discrete intensity points.
  - Characters can be defined with rectangular grids, or they can be defined with curved outlines.



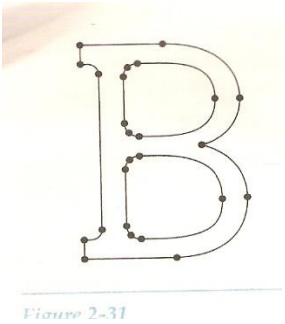


Figure 2-31

  - To reduce memory
    - Frame buffer is organized as linked list
    - Encoding intensity information

**Run-Length Encoding**

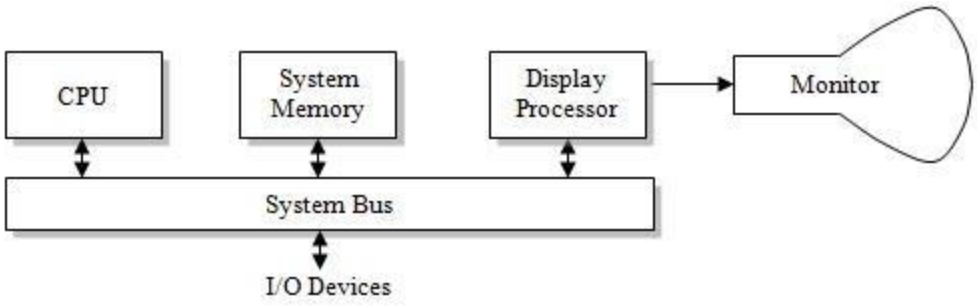
  - To reduce the memory space required to store the image information, each scan line are stored as a set of integer pairs.
  - One number of each pair indicates an intensity value.
  - Second number specifies number of adjacent pixels the scan line that is also having same intensity.
  - This technique is called run-length encoding.
  - Advantage – save storage space if a picture is to be constructed with long runs of single colour

**Cell encoding**

  - Encode the raster as set of rectangular areas
  - Disadvantage - Intensity changes are difficult to make

❖ **Random Scan System**

  - The following diagram will shows the architecture of random scan system:




- Application program is input & stored in system memory along with a graphics package
- Graphics commands in application program are translated to display file by graphics package & stored in system memory
- Display processor access display file to refresh the screen
- At each refresh cycle, display processor cycles through each command in display file
- Display processor in random scan system is also called as display processing unit or a graphics controller
- Graphics patterns are drawn by directing e- beam along the component lines of the picture
- Lines are defined by values for their coordinate endpoints & input coordinate values are converted to x& y deflection voltages
- A scene is then drawn one line at a time by positioning the beam to fill in the line between specified endpoints

❖ **Graphics Monitors And Workstations**

- Most graphics monitors today operate as raster scan displays.
- Graphics systems range from small general-purpose computer systems with graphics capabilities to sophisticated full color systems that are designed specifically for graphics applications

❖ **Various Input Devices**

- Various devices are available for data input on graphics workstations. Most systems have a keyboard and one or more additional devices specially designed for interactive input.

**Keyboards**

- An alphanumeric keyboard on a graphics system is used primarily as a device for entering text strings.
- The keyboard is an efficient device for inputting such non-graphic data as picture labels associated with a graphics display.
- Keyboards can also be provided with features to facilitate entry of screen co-ordinates, menu selections or graphics functions.
- Cursor-control keys and functions keys are common features of general purpose keyboards.
- Function keys allow users to enter frequently used operations in a single keystroke
- Additionally, a numeric keyboard is often included on the keyboard, for fast entry of numeric data.

**Mouse**

- A mouse is a small hand-held device used to position the screen cursor. Wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement. Another method of detecting mouse motion is with an optical sensor. One, two or three buttons are usually included on top of the mouse for signaling the execution of some operation.

**Trackball & Spaceball**

- As the name implies, a trackball is a ball that can be rotated with the finger or palm of the hand, to produce screen-cursor movement. Potentiometers attached to the ball, measure the amount and direction of rotation. Track balls are often mounted on keyboards.
- While a trackball is a two-dimensional positioning device, a Spaceball provides six degrees of freedom. The Spaceball does not move. Strain gauges measure the amount of pressure applied to the Spaceball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. Spaceballs are used for 3D positioning and selection generations in virtual reality systems.

**Joysticks**

- A joystick consists of a small, vertical lever (called the stick) mounted on a based that is used to steer the screen cursor around. The distance that the stick is moved in any direction from its center position corresponds to screen-cursor movement in that direction. Potentiometers mounted at the base of the joystick measure the amount of the movement, and springs return the stick to the centre position when it is released. One or more buttons can be programmed to act as input switches.

**Data Glove**

- Data glove can be used to grasp a "virtual" object. The glove is constructed with a series of sensors that detect hand and finger motions. Electromagnetic coupling between transmitting antennas and receiving antennas is used to provide information about the position and orientation of the hand. Input from a glove can be used to position or manipulate objects in a virtual scene.

**Digitizers**

- A common device for drawing, partitioning or interactively selecting co-ordinates positions on an object is digitizer. These devices can be used to input co-ordinate values in either a 2D or 3D space. One type of digitizer is the graphics tablet which is used to input 2D co-ordinates by activating a hand cursor at selected positions on a flat surface. A hand cursor contains cross hairs for sighting positions, while a stylus is a pencil shaped device that is pointed at positions on the tablet.

**Image Scanners**

- Drawings, graph, photos or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored. The objects are stored in and array, these representations of a picture, can be transformed like rotation, scaling and enhancements.

**Touch Panels**

- It allows displayed objects or screen positions to be selected with the touch of a finger. Touch input can be recorded using optical, electrical or acoustical methods. Optical touch panels employ a line of infrared Light-Emitting Diodes (LED) along one vertical edge and along one horizontal edge of the frame. The opposite vertical and horizontal edges contain light detectors. These detectors are used to record which beams are interpreted when the panel is touched. The two crossing beams that are interpreted identify the horizontal and vertical co-ordinates of the screen position selected.

**Light Pens**

- Light pens are pencil shaped devices are used to select screen positions by detecting the light coming from points on the CRT screen. They are sensitive to short bust of light emitted from the phosphor coating; at the instant the electron beam strikes a particular point.
- An activated light pen, pointed at a spot on the screen as the electron beam lights up the spot, generates on electrical pulse that causes the co-ordinate position of the electron beam to be recorded. As with cursor-positioning devices, recorded light pen co-ordinates can be used to position an object or to select a procession option.

**Voice Systems**

- Speech recognizers are used in some graphics workstations as input devices to accept voice commands. The voice-system input can be used to initialize graphics operations to enter data. These systems operate by matching an input against a predefined dictionary of words and phrases. A dictionary is setup for the operator; each word is spoken several times and stored. When a voice command is given, the system searches the dictionary for a match.

**Graphics Software**

- In CG, Graphics Software is a program or collection of programs that enable a person to manipulate visual images on a computer.
- There are two general classification for Graphics Software:
  - **General Programming Packages**
  - **Special Purpose Application Package**

**General Programming Packages**

- A general programming package provides extensive set of graphics functions that can be used I n high level programming language such as C or FORTRAN.
- An example of a general graphics programming package is the GL (Graphics Library) system on Silicon Graphics equipment.
- Basic functions in a general package include those for generating picture components, setting color and intensity values, selecting views and BC applying transformations.

**Special Purpose Application Packages:**

- Application graphics packages are designed for non-programmers, so that users can generate displays without worrying about how graphics operations work.
- The interface to the graphics routines in such packages allows users to communicate with the programs in their own terms. Example: CAD

**GRAPHICS FUNCTIONS**

- The basic building blocks for picture are referred to as output primitives. They include points for straight lines, curve, lines, filled areas and other general purpose functions for polygon , circles etc.
- The attributes are the properties of output. They include intensity and color specification line styles, text styles and area filling patterns.
- Regardless of the differences in the display device, the most graphics system offers a similar set of graphics primitive commands. Some graphical functions that are most commonly used are described below:
- **Initgraph():** This function initializes the graphic system by loading a graphics driver from disk. Then putting the system into graphic mode. Initgraph() also set graphics settings like color, current position etc. to their defaults. **void Initgraph (int * driver, int * mode, char * path);**
    - *mode: is an integer that specifies the initials graphics mode.
    - *driver: is an integer that specifies the graphics driver to be used.
    - If the *driver=DETECT, Initgraph sets* mode to the highest resolution a variable for the detective driver.
    - *path: specifies the directory where Initgraph function looks for graphics driver(*.BGI) file. It first looks in the path specified, then if they are not there, it looks in the current directory. If path is null, the driver files must be in current directory.

**SOFTWARE STANDARDS**

- The primary goal of standardized graphics software is portability.
- When packages are designed with standard graphics functions, software's can be moved easily from one hardware system to another.
- Without standards, programs designed for one hardware often can't be transformed to another system without extensive rewriting of the programs. Thus we require machine independent graphical language.
- Graphical Kernel System (GKS) was approved as the first graphics software standard by the International standard Organization (ISO) and by various national standards organizations, including the American National Standards Institute (ANSI).
- Although GKS was originally designed as a 2D graphics, a 3D GKS extension was subsequently developed.
- The second software standard to be developed and approved by the standards organization was PHIGS (Programmer's Hierarchical Interactive Graphics Standard) which is an extension of GKS.

- Increased facilities for object modeling, color specifications and picture manipulations are provided in PHIGS.
- Standard Graphics functions are defined as a set of specifications that is independent of any programming language. A language binding is then defined as a particular high level programming language. This language gives the syntax for accessing the various standard graphics functions from this language.

❖ **Points and Lines**

Point plotting is accomplished by converting a single coordinate position furnished by an application program into appropriate operations for the output device. With a CRT monitor, for example, the electron beam is turned on to illuminate the screen phosphor at the selected location

Line drawing is accomplished by calculating intermediate positions along the line path between two specified end points positions. An output device is then directed to fill in these positions between the end points Digital devices display a straight line segment by plotting discrete points between the two end points. Discrete coordinate positions along the line path are calculated from the equation of the line. For a raster video display, the line color (intensity) is then loaded into the frame buffer at the corresponding pixel coordinates. Reading from the frame buffer, the video controller then plots "the screen pixels".

Pixel positions are referenced according to scan-line number and column number (pixel position across a scan line). Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, left to right across each scan line

❖ **The Line drawing algorithm** is a graphical algorithm which is used to represent the line segment on discrete graphical media, i.e., printer and pixel-based media."

A line contains two points. The point is an important element of a line.

Properties of a Line Drawing Algorithm

There are the following properties of a good Line Drawing Algorithm.

- **An algorithm should be precise:** Each step of the algorithm must be adequately defined.
- **Finiteness:** An algorithm must contain finiteness. It means the algorithm stops after the execution of all steps.
- **Easy to understand:** An algorithm must help learners to understand the solution in a more natural way.
- **Correctness:** An algorithm must be in the correct manner.
- **Effectiveness:** Thesteps of an algorithm must be valid and efficient.
- **Uniqueness:** All steps of an algorithm should be clearly and uniquely defined, and the result should be based on the given input.
- **Input:** A good algorithm must accept at least one or more input.
- **Output:** An algorithm must generate at least one output.

Equation of the straight line

We can define a straight line with the help of the following equation.

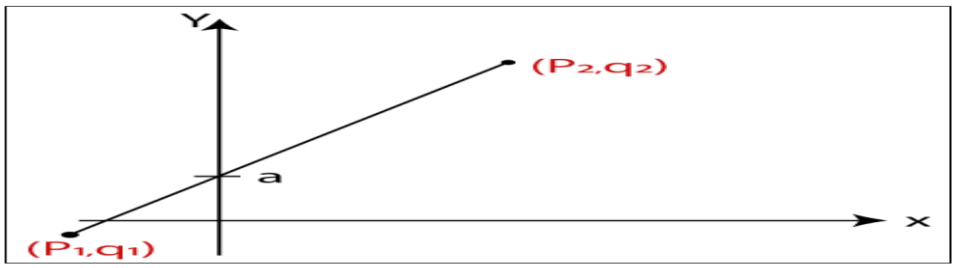
**y= mx + a**

Where,

**(x, y**) = axis of the line.

**m** = Slope of the line.

**a =** Interception point




Let us assume we have two points of the line ($p_1, q_1$) and ($p_2, q_2$).
Now, we will put values of the two points in straight line equation, and we get

$y = mx + a$

$q_2 = mp_2$ ...(1)

$q_1 = mp_1 + a$ ...(2)

We have from equation (1) and (2)

$q_2 – q_1 = mp_2 – mp_1$

$q_2 – q_1 = m (p_2 –p_1)$

The value of $m = (q_2 – q_1)/ (p_2 –p_1)$

$m = ?q / ?p$

**Algorithms of Line Drawing**

There are following algorithms used for drawing a line:

- DDA (Digital Differential Analyzer) Line Drawing Algorithm
- Bresenham's Line Drawing Algorithm
- Mid-Point Line Drawing Algorithm

❖ **DDA (Digital Differential Analyzer) Line Drawing Algorithm**:The Digital Differential Analyzer helps us to interpolate the variables on an interval from one point to another point. We can use the digital Differential Analyzer algorithm to perform rasterization on polygons, lines, and triangles.

Digital Differential Analyzer algorithm is also known as an **incremental method** of scan conversion. In this algorithm, we can perform the calculation in a step by step manner. We use the previous step result in the next step.

As we know the general equation of the straight line is:

$y = mx + c$

Here, **m** is the slope of $(x_1, y_1)$ and $(x_2, y_2)$.

$m = (y_2 – y_1)/ (x_2 – x_1)$

Now, we consider one point $(x_k, y_k)$ and $(x_{k+1}, y_{k+1})$ as the next point.

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

Now, we have to find the slope between the starting point and ending point. There can be following three cases to discuss:

**Case 1:** If $m < 1$

Then **x** coordinate tends to the Unit interval.

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

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

**Case 2:** If $m > 1$

Then **y** coordinate tends to the Unit interval.

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

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

**Case 3:** If **m = 1**

Then **x and y** coordinate tend to the Unit interval.

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

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

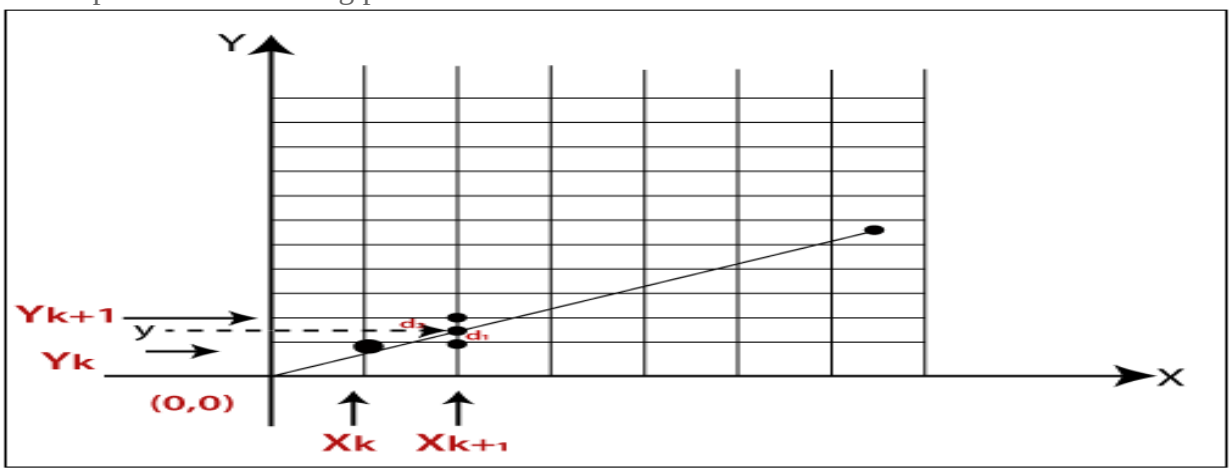
Advantages of Digital Differential Analyzer

- It is a simple algorithm to implement.
- It is a faster algorithm than the direct line equation.
- We cannot use the multiplication method in Digital Differential Analyzer.
- Digital Differential Analyzer algorithm tells us about the overflow of the point when the point changes its location.

Disadvantages of Digital Differential Analyzer

- The floating-point arithmetic implementation of the Digital Differential Analyzer is time-consuming.
- The method of round-off is also time-consuming.
- Sometimes the point position is not accurate.

❖ **Bresenham's Line Drawing algorithm**, we have to calculate the slope (**m**) between the starting point and the ending point.




Algorithm of Bresenham's Line Drawing Algorithm

**Step 1:** Start.

**Step 2:** Now, we consider Starting point as $(x_1, y_1)$ and endingpoint $(x_2, y_2)$.

**Step 3:** Now, we have to calculate **?x** and **?y.**

$?x = x_2 - x_1$

$?y = y_2 - y_1$

$m = ?y/?x$

**Step 4:** Now, we will calculate the decision parameter $p_k$ with following formula.

$p_k = 2?y - ?x$

**Step 5:** Theinitial coordinates of the line are $(x_k, y_k)$, and the next coordinatesare $(x_{k+1}, y_{k+1})$. Now, we are going to calculate two cases for decision parameter $p_k$

**Case 1:** If

$p_k < 0$

Then

$p_{k+1} = p_k + 2?y$

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

$y_{k+1} = y_k$

**Case 2:** If

$p_k >= 0$

Then

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

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

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

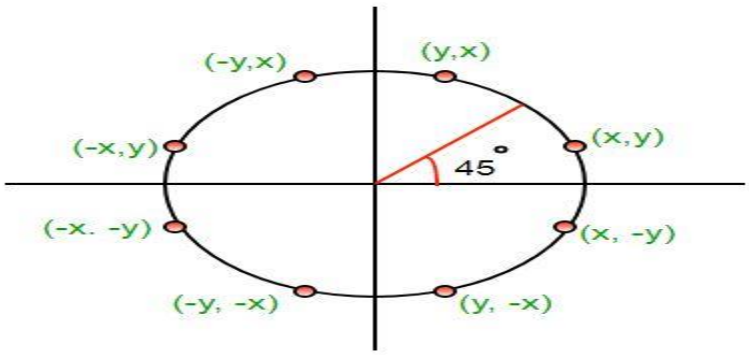
**Step 6:** We will repeat step 5 until we found the ending point of the line and the total number of iterations =**?x-1.**

**Step 7:** Stop.

- **Mid-Point Circle Drawing Algorithm**

The **mid-point** circle drawing algorithm is an algorithm used to determine the points needed for rasterizing a circle.

We use the **mid-point** algorithm to calculate all the perimeter points of the circle in the **first octant** and then print them along with their mirror points in the other octants. This will work because a circle is symmetric about it's centre.




For any given pixel (x, y), the next pixel to be plotted is either **(x, y+1)** or **(x-1, y+1)**. This can be decided by following the steps below.

1. Find the mid-point **p** of the two possible pixels i.e (x-0.5, y+1)
2. If **p** lies inside or on the circle perimeter, we plot the pixel (x, y+1), otherwise if it's outside we plot the pixel (x-1, y+1)

**Boundary Condition :** Whether the mid-point lies inside or outside the circle can be decided by using the formula:-

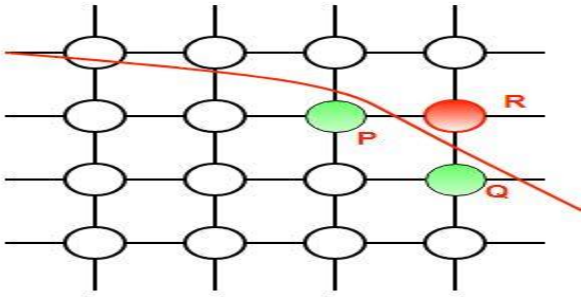
*Given a circle centered at (0,0) and radius r and a point p(x,y)*

***F(p) = $x^2$ + $y^2$ – $r^2$***

*if F(p)<0, the point is inside the circle*

*F(p)=0, the point is on the perimeter*

*F(p)>0, the point is outside the circle*




❖ **Midpoint ellipse drawing algorithm:**

Midpoint ellipse algorithm plots(finds) points of an ellipse on the first quadrant by dividing the quadrant into two regions.
Each point(x, y) is then projected into other three quadrants (-x, y), (x, -y), (-x, -y) i.e. it uses 4-way symmetry.

**Function of ellipse:**

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

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

$f_{ellipse}(x, y) > 0$ then $(x, y)$ is outside the ellipse.

$f_{ellipse}(x, y) = 0$ then $(x, y)$ is on the ellipse.

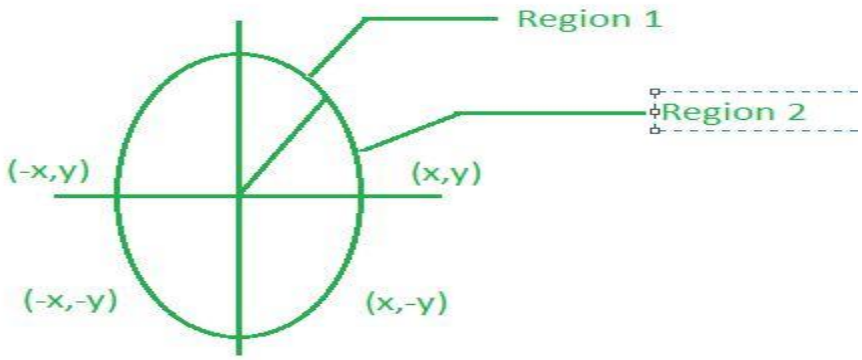
**Decision parameter:**

Initially, we have two decision parameters $p1_0$ in region 1 and $p2_0$ in region 2. These parameters are defined as : $p1_0$ in region 1 is given as :

$p1_0 = r_y^2 + 1/4r_x^2 - r_x^2r_y$

**Mid-Point Ellipse Algorithm :**

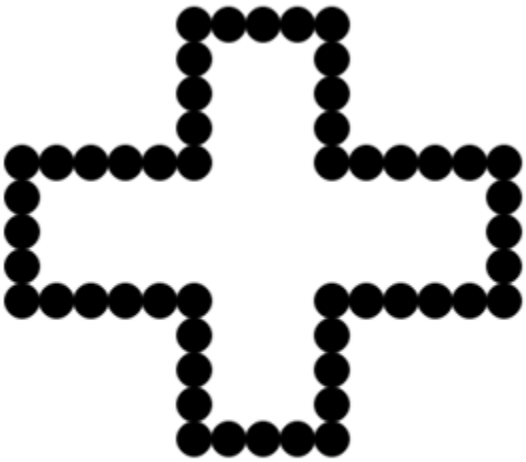
1. Take input radius along x axis and y axis and obtain center of ellipse.
2. Initially, we assume ellipse to be centered at origin and the first point as : $(x, y_0) = (0, r_y)$.
3. Obtain the initial decision parameter for region 1 as: $p1_0 = r_y^2 + 1/4r_x^2 - r_x^2r_y$
4. For every $x_k$ position in region 1 :
   If $p1_k < 0$ then the next point along the is $(x_{k+1}, y_k)$ and $p1_{k+1} = p1_k + 2r_y^2x_{k+1} + r_y^2$
   Else, the next point is $(x_{k+1}, y_{k-1})$
   And $p1_{k+1} = p1_k + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_y^2$
5. Obtain the initial value in region 2 using the last point $(x_0, y_0)$ of region 1 as: $p2_0 = r_y^2(x_0 + 1/2)^2 + r_x^2(y_0 - 1)^2 - r_x^2r_y^2$
6. At each $y_k$ in region 2 starting at k =0 perform the following task.
   If $p2_k > 0$ the next point is $(x_k, y_{k-1})$ and $p2_{k+1} = p2_k - 2r_x^2y_{k+1} + r_x^2$
7. Else, the next point is $(x_{k+1}, y_{k-1})$ and $p2_{k+1} = p2_k + 2r_y^2x_{k+1} - 2r_x^2y_{k+1} + r_x^2$
8. Now obtain the symmetric points in the three quadrants and plot the coordinate value as: x=x+xc, y=y+yc
9. Repeat the steps for region 1 until $2r_y^2x >= 2r_x^2y$


Region 1
Region 2
(-x,y)
(x,y)
(-x,-y)
(x,-y)

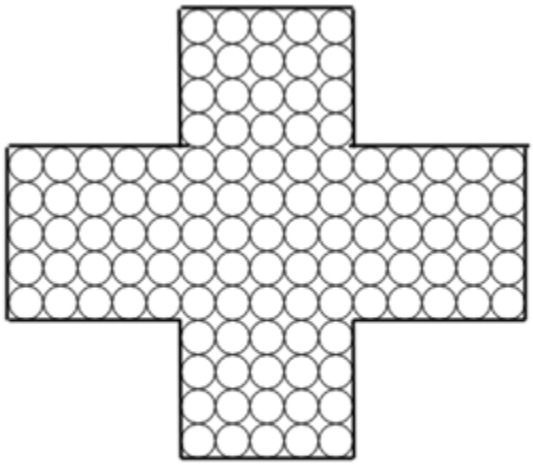
## Unit-2

### ❖ Filled Area Primitives:

Region filling is the process of filling image or region. Filling can be of boundary or interior region as shown in fig. Boundary Fill algorithms are used to fill the boundary and flood-fill algorithm are used to fill the interior.



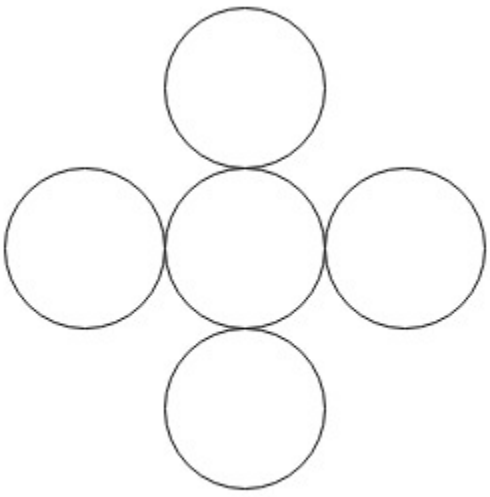
Boundary Filled Region



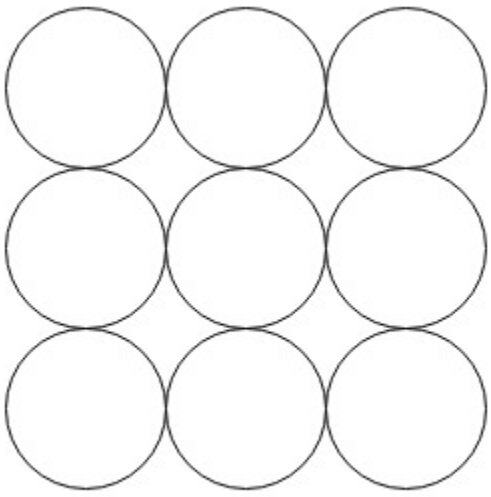
Interior or Flood Filled Region

### 1. Boundary Filled Algorithm:

This algorithm uses the recursive method. First of all, a starting pixel called as the seed is considered. The algorithm checks boundary pixel or adjacent pixels are colored or not. If the adjacent pixel is already filled or colored then leave it, otherwise fill it. The filling is done using four connected or eight connected approaches.



Four Connected



Eight Connected

Four connected approaches is more suitable than the eight connected approaches.

**1. Four connected approaches:** In this approach, left, right, above, below pixels are tested.

**2. Eight connected approaches:** In this approach, left, right, above, below and four diagonals are selected.

Boundary can be checked by seeing pixels from left and right first. Then pixels are checked by seeing pixels from top to bottom. The algorithm takes time and memory because some recursive calls are needed.

Problem with recursive boundary fill algorithm:

It may not fill regions sometimes correctly when some interior pixel is already filled with color. The algorithm will check this boundary pixel for filling and will found already filled so recursive process will terminate. This may vary because of another interior pixel unfilled.

So check all pixels color before applying the algorithm.

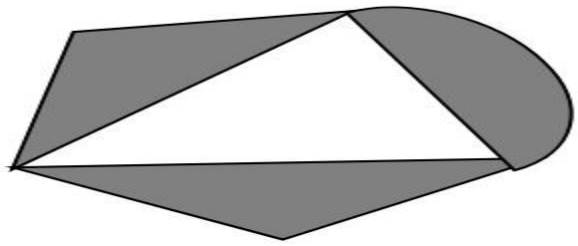
Algorithm:

```
Procedure fill (x, y, color, color1: integer)
int c;
c=getpixel (x, y);
if (c!=color) (c!=color1)
{
   setpixel (x, y, color)
   fill (x+1, y, color, color 1);
    fill (x-1, y, color, color 1);
   fill (x, y+1, color, color 1);
   fill (x, y-1, color, color 1);
}
```

**2.Flood Fill Algorithm:**

In this method, a point or seed which is inside region is selected. This point is called a seed point. Then four connected approaches or eight connected approaches is used to fill with specified color.

The flood fill algorithm has many characters similar to boundary fill. But this method is more suitable for filling multiple colors boundary. When boundary is of many colors and interior is to be filled with one color we use this algorithm.



In fill algorithm, we start from a specified interior point (x, y) and reassign all pixel values are currently set to a given interior color with the desired color. Using either a 4-connected or 8-connected approaches, we then step through pixel positions until all interior points have been repainted.

Disadvantage:

1. Very slow algorithm
2. May be fail for large polygons
3. Initial pixel required more knowledge about surrounding pixels.

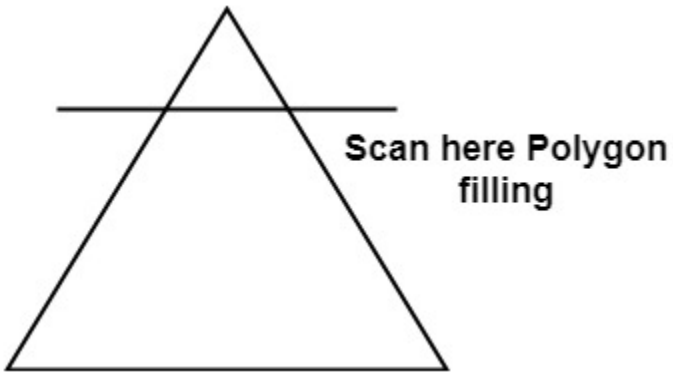
Algorithm:

```
Procedure floodfill (x, y,fill_ color, old_color: integer)
   If (getpixel (x, y)=old_color)
   {
   setpixel (x, y, fill_color);
   fill (x+1, y, fill_color, old_color);
    fill (x-1, y, fill_color, old_color);
   fill (x, y+1, fill_color, old_color);
   fill (x, y-1, fill_color, old_color);
    }
}
```

**3.Scan Line Polygon Fill Algorithm:**

This algorithm lines interior points of a polygon on the scan line and these points are done on or off according to requirement. The polygon is filled with various colors by coloring various pixels.

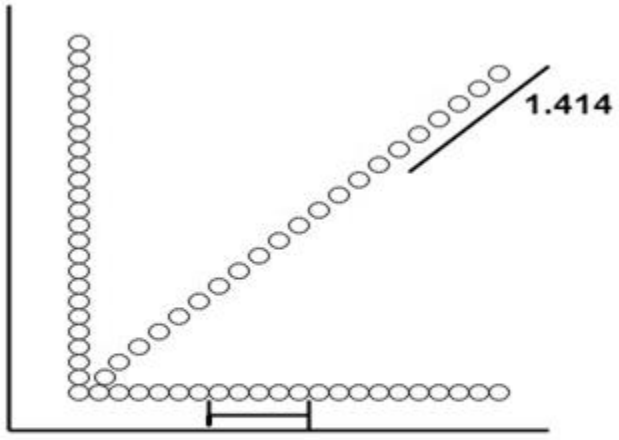
In above figure polygon and a line cutting polygon in shown. First of all, scanning is done. Scanning is done using raster scanning concept on display device. The beam starts scanning from the top left corner of the screen and goes toward the bottom right corner as the endpoint. The algorithms find points of intersection of the line with polygon while moving from left to right and top to bottom. The various points of intersection are stored in the frame buffer. The intensities of such points is keep high. Concept of coherence property is used. According to this property if a pixel is inside the polygon, then its next pixel will be inside the polygon.




Side effects of Scan Conversion:

**1. Staircase or Jagged:** Staircase like appearance is seen while the scan was converting line or circle.

**2. Unequal Intensity:** It deals with unequal appearance of the brightness of different lines. An inclined line appears less bright as compared to the horizontal and vertical line.




**Pixels along with horizontal line are 1 unit apart and vertical.**
**Pixels along diagonal line are 1.414 units.**

- Introduction of Transformations

Computer Graphics provide the facility of viewing object from different angles. The architect can study building from different angles i.e.

1. Front Evaluation
2. Side elevation
3. Top plan

A Cartographer can change the size of charts and topographical maps. So if graphics images are coded as numbers, the numbers can be stored in memory. These numbers are modified by mathematical operations called as Transformation.

The purpose of using computers for drawing is to provide facility to user to view the object from different angles, enlarging or reducing the scale or shape of object called as Transformation.

1. Two essential aspects of transformation are given below: Each transformation is a single entity. It can be denoted by a unique name or symbol.
2. It is possible to combine two transformations, after connecting a single transformation is obtained, e.g., A is a transformation for translation. The B transformation performs scaling. The combination of two is C=AB. So C is obtained by concatenation property.

There are two complementary points of view for describing object transformation.

1. Geometric Transformation: The object itself is transformed relative to the coordinate system or background. The mathematical statement of this viewpoint is defined by geometric transformations applied to each point of the object.
2. Coordinate Transformation: The object is held stationary while the coordinate system is transformed relative to the object. This effect is attained through the application of coordinate transformations.

An example that helps to distinguish these two viewpoints:

The movement of an automobile against a scenic background we can simulate this by

- Moving the automobile while keeping the background fixed-(Geometric Transformation)
- We can keep the car fixed while moving the background scenery- (Coordinate Transformation)

Types of Transformations:

1. Translation
2. Scaling
3. Rotating
4. Reflection
5. Shearing

❖ Translation

It is the straight line movement of an object from one position to another is called Translation. Here the object is positioned from one coordinate location to another.

## Translation of point:

To translate a point from coordinate position (x, y) to another ($x_1$ $y_1$), we add algebraically the translation distances $T_x$ and $T_y$ to original coordinate.

$x_1=x+T_x$

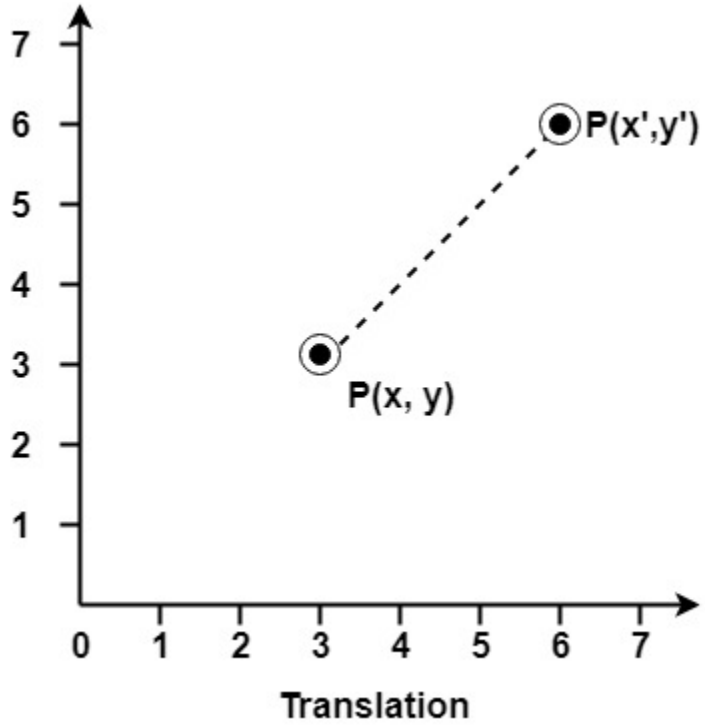
$y_1=y+T_y$

The translation pair ($T_x$,$T_y$) is called as shift vector.

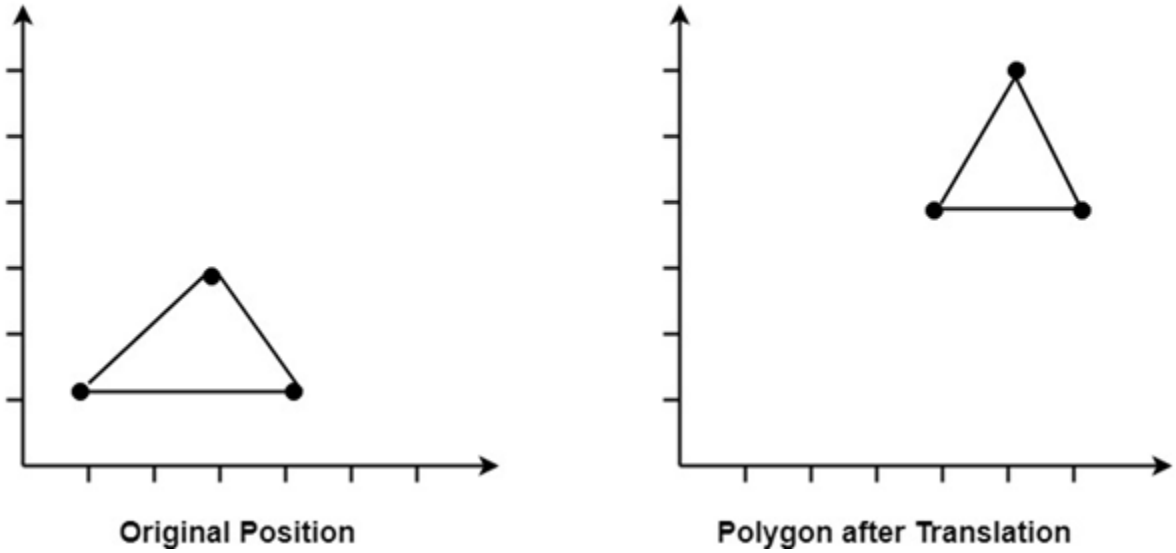
Translation is a movement of objects without deformation. Every position or point is translated by the same amount. When the straight line is translated, then it will be drawn using endpoints.

For translating polygon, each vertex of the polygon is converted to a new position. Similarly, curved objects are translated. To change the position of the circle or ellipse its center coordinates are transformed, then the object is drawn using new coordinates.

Let P is a point with coordinates (x, y). It will be translated as ($x^1$ $y^1$).




Translation of Polygon




Matrix for Translation:

$$\begin{vmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{vmatrix} \quad \text{Or} \quad \begin{vmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{vmatrix}$$

### ❖ Scaling:

It is used to alter or change the size of objects. The change is done using scaling factors. There are two scaling factors, i.e. $S_x$ in x direction $S_y$ in y-direction. If the original position is x and y. Scaling factors are $S_x$ and $S_y$ then the value of coordinates after scaling will be $x^1$ and $y_1$.

If the picture to be enlarged to twice its original size then $S_x = S_y = 2$. If $S_x$and $S_y$ are not equal then scaling will occur but it will elongate or distort the picture.

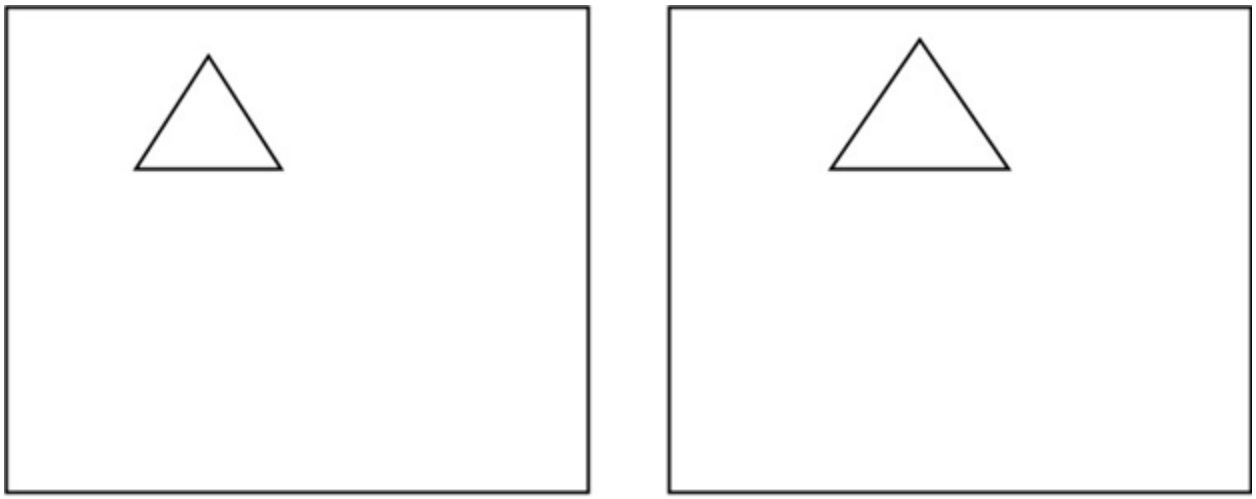
If scaling factors are less than one, then the size of the object will be reduced. If scaling factors are higher than one, then the size of the object will be enlarged.

If $S_x$and $S_y$are equal it is also called as Uniform Scaling. If not equal then called as Differential Scaling. If scaling factors with values less than one will move the object closer to coordinate origin, while a value higher than one will move coordinate position farther from origin.

**Enlargement: If $T_1=\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}$,If $(x_1\ y_1)$is original position and $T_1$is translation vector then $(x_2\ y_2)$ are coordinated after scaling**

$$[x_2y_2] = [\ x_1y_1] \begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} = [2x_1\,2y_1]$$

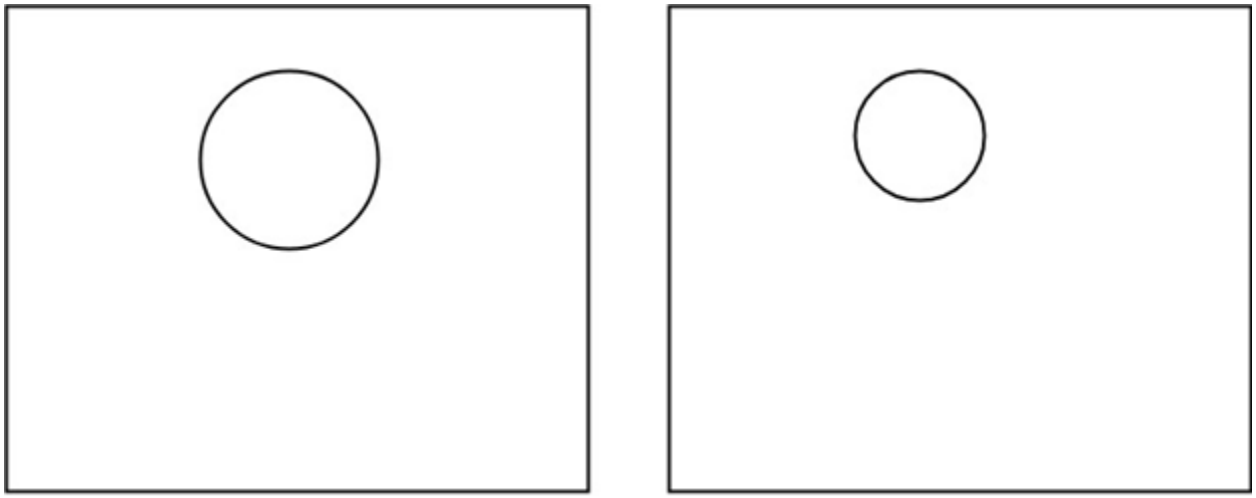
**The image will be enlarged two times**



Original Image Enlarged Image

**Reduction: If $T_1=\begin{bmatrix}0.5 & 0\\ 0 & 0.5\end{bmatrix}$. If $(x_1\ y_1)$ is original position and $T_1$ is translation vector, then $(x_2\ y_2)$ are coordinates after scaling**

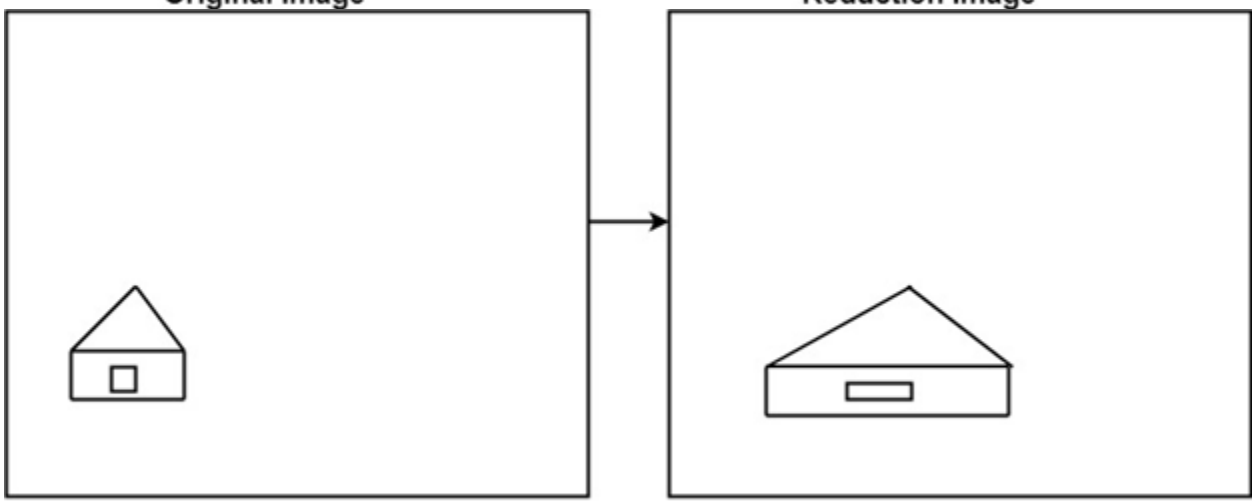
$$[x_2 y_2] = [\ x_1 y_1] \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} = [.5x_1 . 5y_1]$$



Original Image

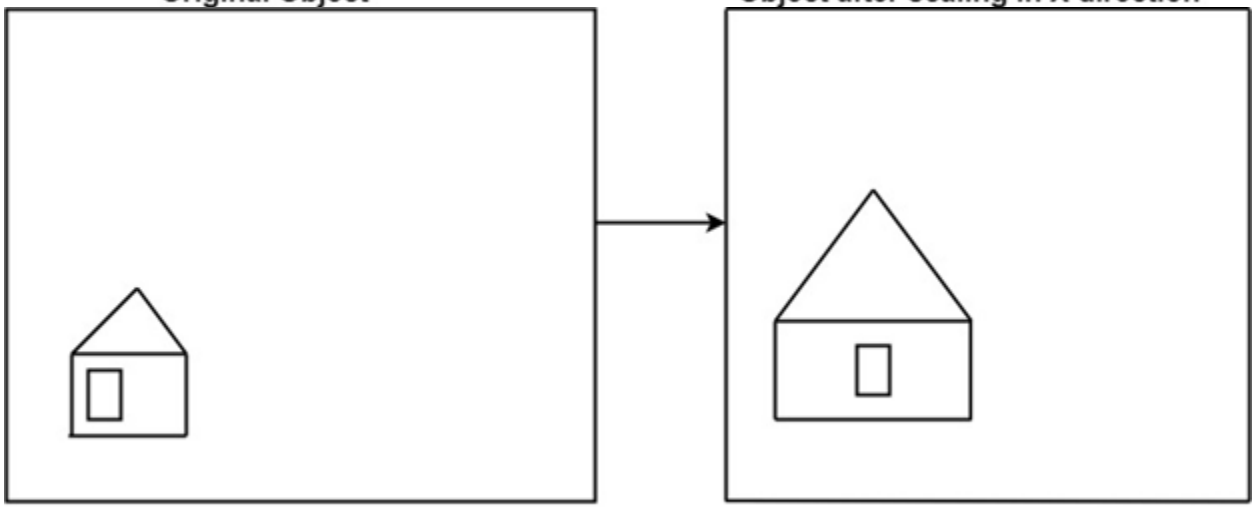


Reduction Image



Original Object

Object after scaling in X direction



Original Object

Object after scaling in Y direction

Matrix for Scaling:

$$S = \begin{vmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

**Example: Prove that 2D Scaling transformations are commutative i.e, $S_1 S_2 = S_2 S_1$.**

**Solution: $S_1$ and $S_2$ are scaling matrices**

$$S_1 = \begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S_1 * S_2 = \begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} Sx_1Sx_2 & 0 & 0 \\ 0 & Sy_1Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\text{equation 1}$$

$$S_2 * S_1 = \begin{bmatrix} Sx_2 & 0 & 0 \\ 0 & Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Sx_1 & 0 & 0 \\ 0 & Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} Sx_2Sx_1 & 0 & 0 \\ 0 & Sy_2Sy_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\text{equation 2}$$

From equation 1 and 2

$$S_1S_2 = S_2S_1. \text{ Hence Proved}$$

- Rotation:

It is a process of changing the angle of the object. Rotation can be clockwise or anticlockwise. For rotation, we have to specify the angle of rotation and rotation point. Rotation point is also called a pivot point. It is print about which object is rotated.

Types of Rotation:

1. Anticlockwise
2. Counterclockwise

The positive value of the pivot point (rotation angle) rotates an object in a counter-clockwise (anti-clockwise) direction.

The negative value of the pivot point (rotation angle) rotates an object in a clockwise direction.

When the object is rotated, then every point of the object is rotated by the same angle.

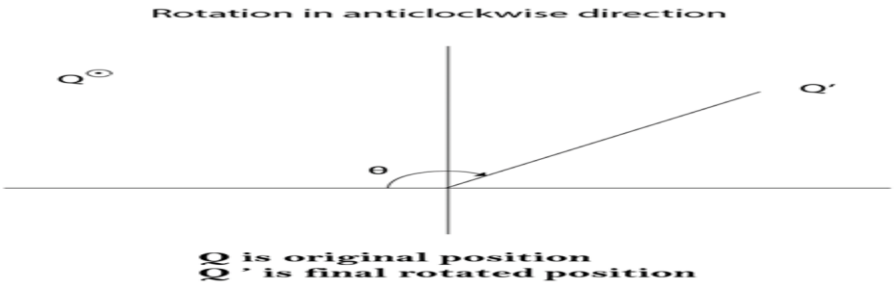
**Straight Line:** Straight Line is rotated by the endpoints with the same angle and redrawing the line between new endpoints.

**Polygon:** Polygon is rotated by shifting every vertex using the same rotational angle.

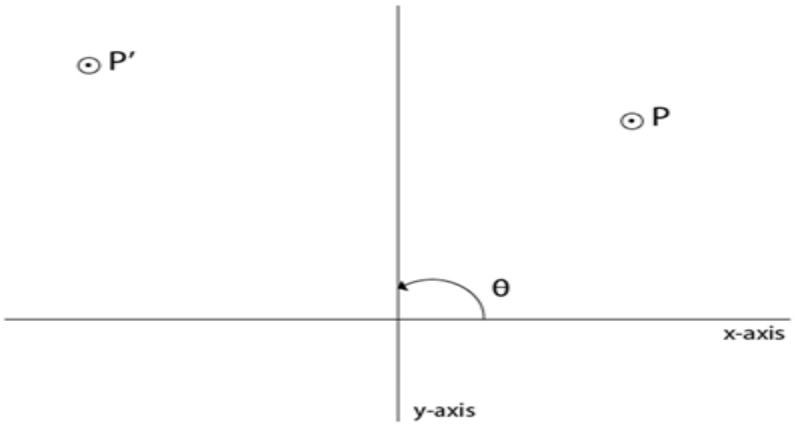
**Curved Lines:** Curved Lines are rotated by repositioning of all points and drawing of the curve at new positions.

**Circle:** It can be obtained by center position by the specified angle.

**Ellipse:** Its rotation can be obtained by rotating major and minor axis of an ellipse by the desired angle.




Rotation of P in clockwise direction




**P is original Position**
**P' is final position or position after rotation**
**where θ is angle of rotation**

Matrix for rotation is a clockwise direction.

$$R=\begin{bmatrix}\cos\theta & \sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}$$

Matrix for rotation is an anticlockwise direction.

$$R=\begin{bmatrix}\cos\theta & \sin\theta \\ -\sin\theta & \cos\theta\end{bmatrix}$$

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

Matrix for homogeneous co-ordinate rotation (anticlockwise)

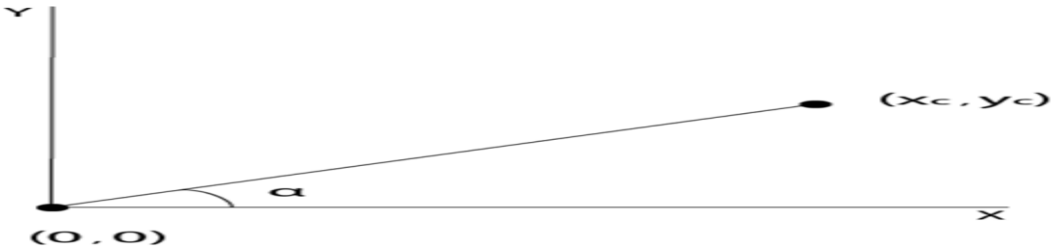
$$R=\begin{bmatrix}\cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1\end{bmatrix}$$

**Rotation about an arbitrary point:** If we want to rotate an object or point about an arbitrary point, first of all, we translate the point about which we want to rotate to the origin. Then rotate point or object about the origin, and at the end, we again translate it to the original place. We get rotation about an arbitrary point.

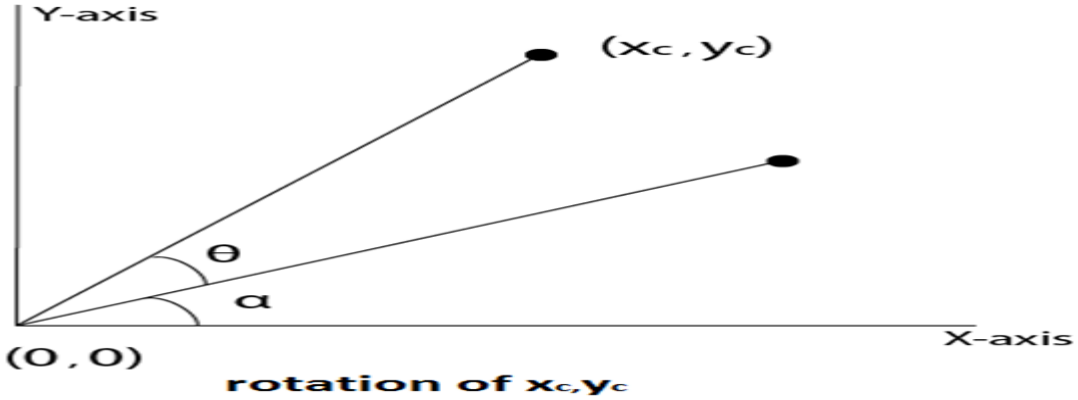
**Example:** The point (x, y) is to be rotated

The ($x_c$ $y_c$) is a point about which counterclockwise rotation is done

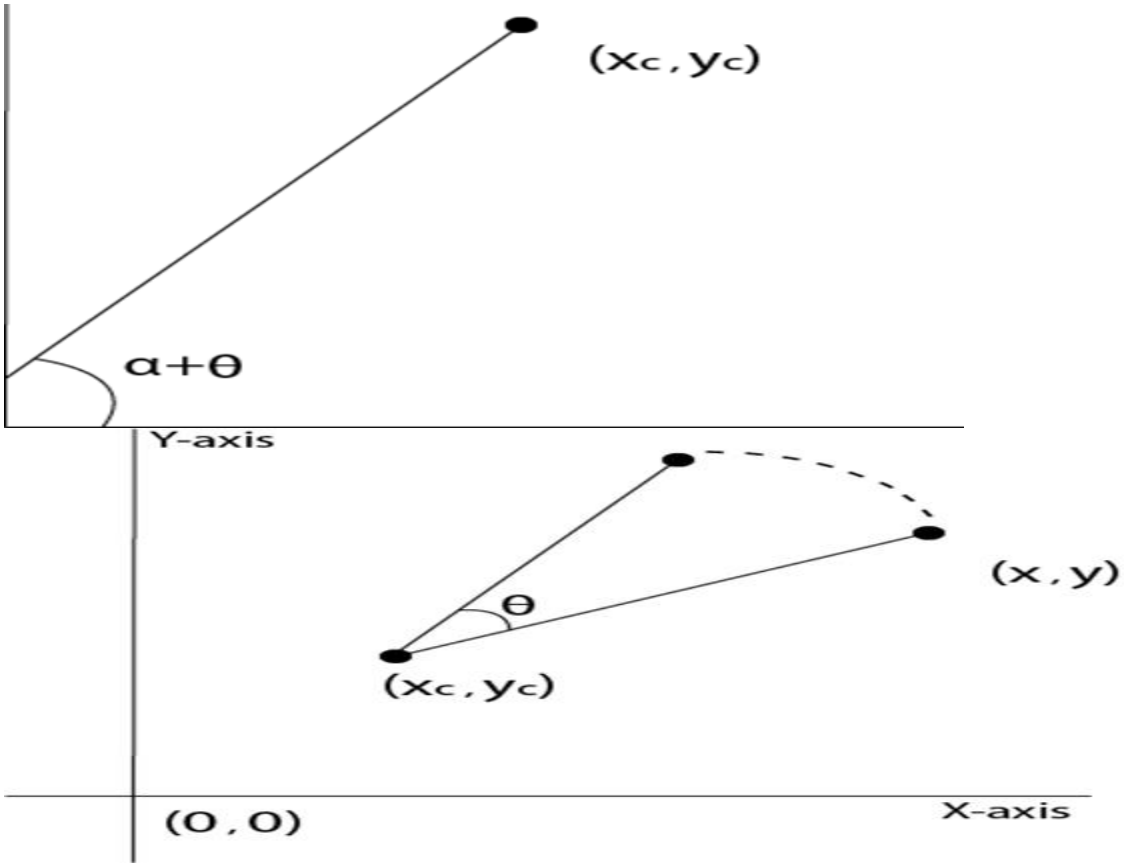
**Step1:** Translate point ($x_c$ $y_c$) to origin




**Step2:** Rotation of (x, y) about the origin




**Step3:** Translation of center of rotation back to its original position




**Example2:** Rotate a line CD whose endpoints are (3, 4) and (12, 15) about origin through a 45° anticlockwise direction.

**Solution:** The point C (3, 4)

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$\theta = 45°$$

Let

$$R = \begin{bmatrix} \cos 45° & \sin 45° \\ -\sin 45° & \cos 45° \end{bmatrix}$$

$$R = \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix}$$

The point A (3, 4) after rotation will be

$$[x, y] = [3, 4] \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix}$$

$$= [3 * 0.707 - 4 * 0.707 \qquad 3 * 0.707 + 4 * 0.707]$$

[2.121-2.8282.21+2.828]

[-.707 4.949]

The rotation of point B (12, 15)

$$R_1 = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix}$$

The point B (12, 15) will be

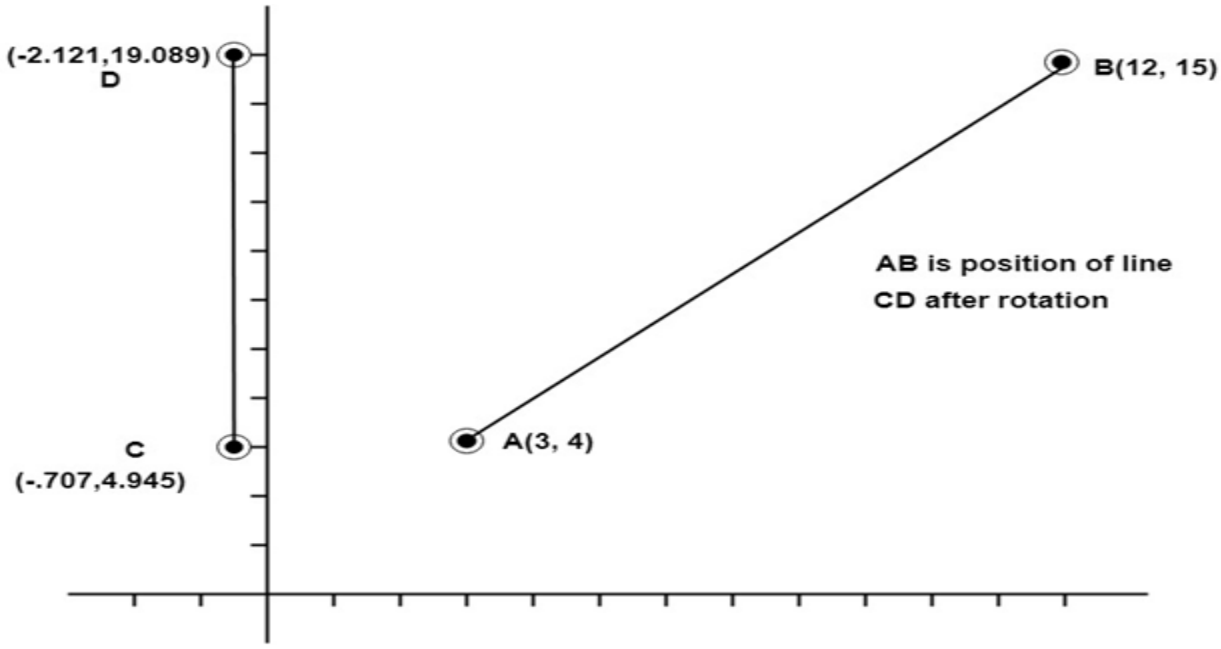
$$[x, y] = [12, 15] \begin{bmatrix} 0.707 & 0.707 \\ -0.707 & -0.707 \end{bmatrix}$$

$$= [12 * 0.707 + 15 * 0.707 \qquad 12 * 0.707 + 15 * 0.707]$$

$$= [(8.484\text{-}10.605) \; (8.484 + 10.605)]$$

$$= [-2.121 \quad 19.089]$$

So line AB after rotation at 45°become [.707, 4.945] and [-2.121, 19.089]




❖ Reflection:

It is a transformation which produces a mirror image of an object. The mirror image can be either about x-axis or y-axis. The object is rotated by180°.

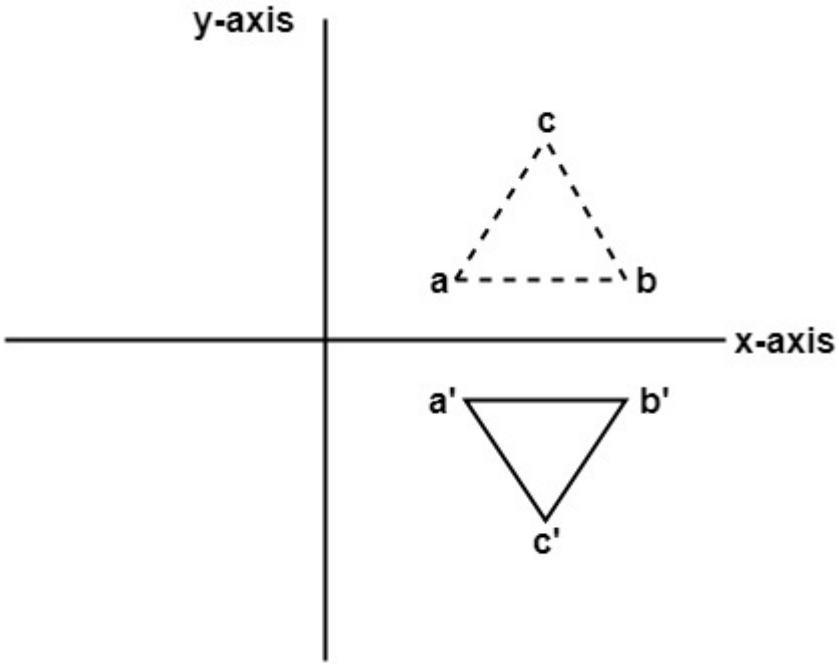
Types of Reflection:

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

**1. Reflection about x-axis:** The object can be reflected about x-axis with the help of the following matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In this transformation value of x will remain same whereas the value of y will become negative. Following figures shows the reflection of the object axis. The object will lie another side of the x-axis.

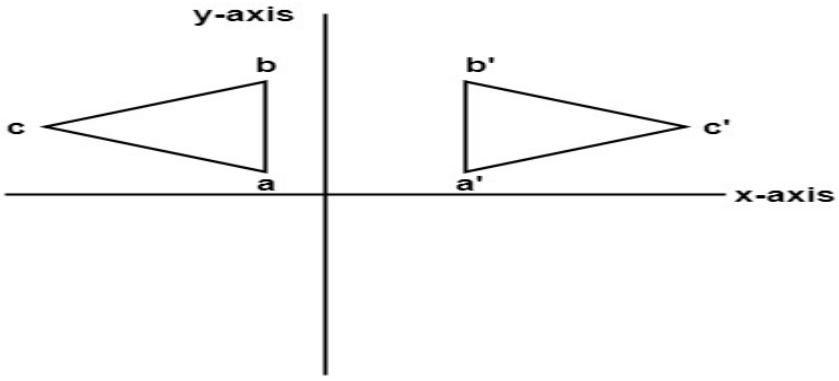



**2. Reflection about y-axis:** The object can be reflected about y-axis with the help of following transformation matrix

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

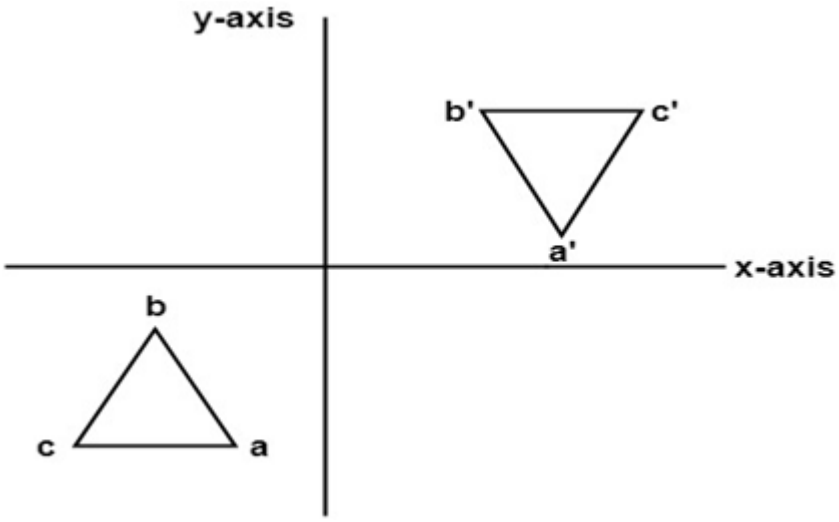
Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.

The following figure shows the reflection about the y-axis



**3. Reflection about an axis perpendicular to xy plane and passing through origin:** In the matrix of this transformation is given below

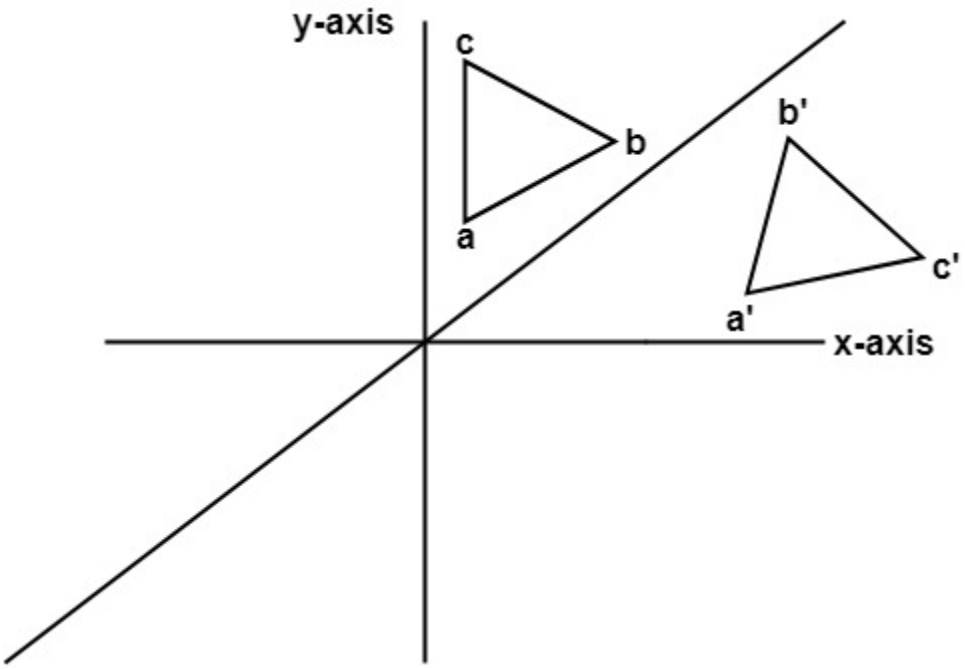
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$




In this value of x and y both will be reversed. This is also called as half revolution about the origin.

**4. Reflection about line y=x:** The object may be reflected about line y = x with the help of following transformation matrix

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



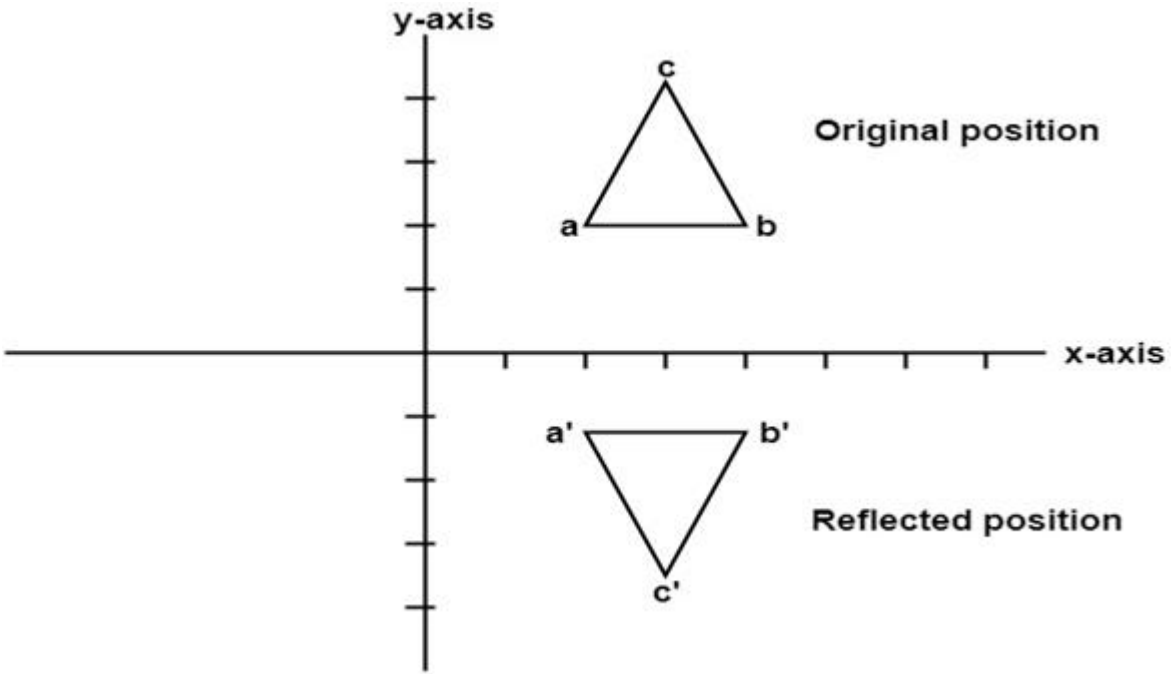
First of all, the object is rotated at 45°. The direction of rotation is clockwise. After it reflection is done concerning x-axis. The last step is the rotation of y=x back to its original position that is counterclockwise at 45°.

**Example:** A triangle ABC is given. The coordinates of A, B, C are given as

A (3 4)
B (6 4)
C (4 8)

Find reflected position of triangle i.e., to the x-axis.

**Solution:**




The matrix for reflection about x axis $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

The a point coordinates after reflection

$$(x, y) = (3, 4)\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$(x, y) = [3, -4]$

The b point coordinates after reflection

$$(x, y) = (6, 4)\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$(x, y) = [6, -4]$$

The coordinate of point c after reflection

$$(x, y) = (4, 8)\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$(x, y) = [4, -8]$$

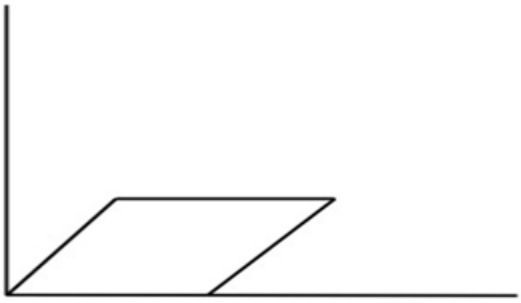
- Shearing:

It is transformation which changes the shape of object. The sliding of layers of object occur. The shear can be in one direction or in two directions.

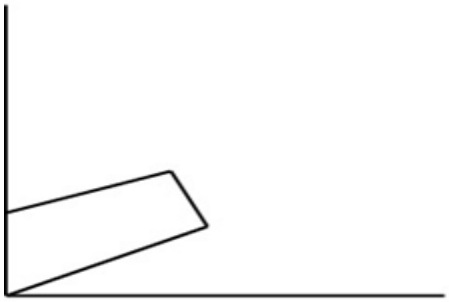
**Shearing in the X-direction:** In this horizontal shearing sliding of layers occur. The homogeneous matrix for shearing in the x-direction is shown below:

$$\begin{bmatrix}1 & 0 & 0\\ Sh_x & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

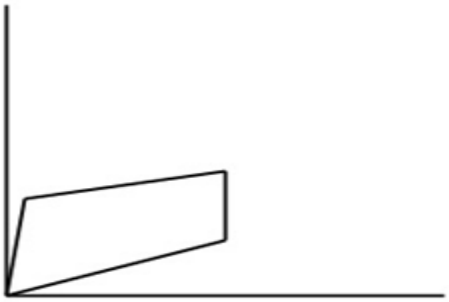
Original Object



Shear in X direction



Shear in Y direction



Shear in both directions

**Shearing in the Y-direction:** Here shearing is done by sliding along vertical or y-axis.

$$\begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Shearing in X-Y directions:** Here layers will be slided in both x as well as y direction. The sliding will be in horizontal as well as vertical direction. The shape of the object will be distorted. The matrix of shear in both directions is given by:

$$\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

## Matrix Representation of 2D Transformation

**1.** Scaling $\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

**2.** Rotation (clockwise) $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

**3.** Rotation (anti-clock) $\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$

**4.** Translation $\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ t_x & t_y \end{bmatrix}$

**5.** Reflection $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

(about x axis)

**6.** Reflection $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

(about y axis)

**7.** Reflection $\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$

(about origin)

**8.** Reflection about Y=X $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

**9.** Reflection about Y= –X $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$

**10.** Shearing in X direction $\begin{bmatrix} 1 & 0 \\ Sh_x & 1 \end{bmatrix}$

**11.** Shearing in Y direction $\begin{bmatrix} 1 & Sh_y \\ 0 & 1 \end{bmatrix}$

**12.** Shearing in both x and y direction $\begin{bmatrix} 1 & Sh_y \\ Sh_x & 1 \end{bmatrix}$

### ❖ Homogeneous Coordinates

The rotation of a point, straight line or an entire image on the screen, about a point other than origin, is achieved by first moving the image until the point of rotation occupies the origin, then performing rotation, then finally moving the image to its original position.

The moving of an image from one place to another in a straight line is called a translation. A translation may be done by adding or subtracting to each point, the amount, by which picture is required to be shifted.

Translation of point by the change of coordinate cannot be combined with other transformation by using simple matrix application. Such a combination is essential if we wish to rotate an image about a point other than origin by translation, rotation again translation.

To combine these three transformations into a single transformation, homogeneous coordinates are used. In homogeneous coordinate system, two-dimensional coordinate positions (x, y) are represented by triple-coordinates.

Homogeneous coordinates are generally used in design and construction applications. Here we perform translations, rotations, scaling to fit the picture into proper position.

**Example of representing coordinates into a homogeneous coordinate system:** For two-dimensional geometric transformation, we can choose homogeneous parameter h to any non-zero value. For our convenience take it as one. Each two-dimensional position is then represented with homogeneous coordinates (x, y, 1).

**Following are matrix for two-dimensional transformation in homogeneous coordinate:**

**1.** Translation $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$ or $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

**2.** Scaling $\begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**3.** Rotation (clockwise) $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**4.** Rotation (anti-clock) $\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**5.** Reflection against X axis $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**6.** Reflection against Y axis $\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**7.** Reflection against origin $\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**8.** Reflection against line Y=X $\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**9.** Reflection against Y= −X $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**10.** Shearing in X direction $\begin{bmatrix} 1 & 0 & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**11.** Shearing in Y direction $\begin{bmatrix} 1 & Sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**12.** Shearing in both x and y direction $\begin{bmatrix} 1 & Sh_y & 0 \\ Sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

**The viewing pipeline:** A world-coordinate area selected for display is called a window. An area on adisplay device to which a window is mapped is called a viewport. The window defines what is to be viewed; the viewport defines where it is to be displayed. Often, windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process. In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation. Sometimes the two-dimensional viewing transformation is simply referred to as the window-to-viewport transformation or the windowing transformation. But, in general, viewing involves more than just the transformation from the window to the viewport. Figure 6-1 illustrates the mapping of a picture section that falls within a rectangular window onto a designated &angular viewport.

**Viewing transformation or window to viewport transformation or windowing transformation:** The mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation etc.

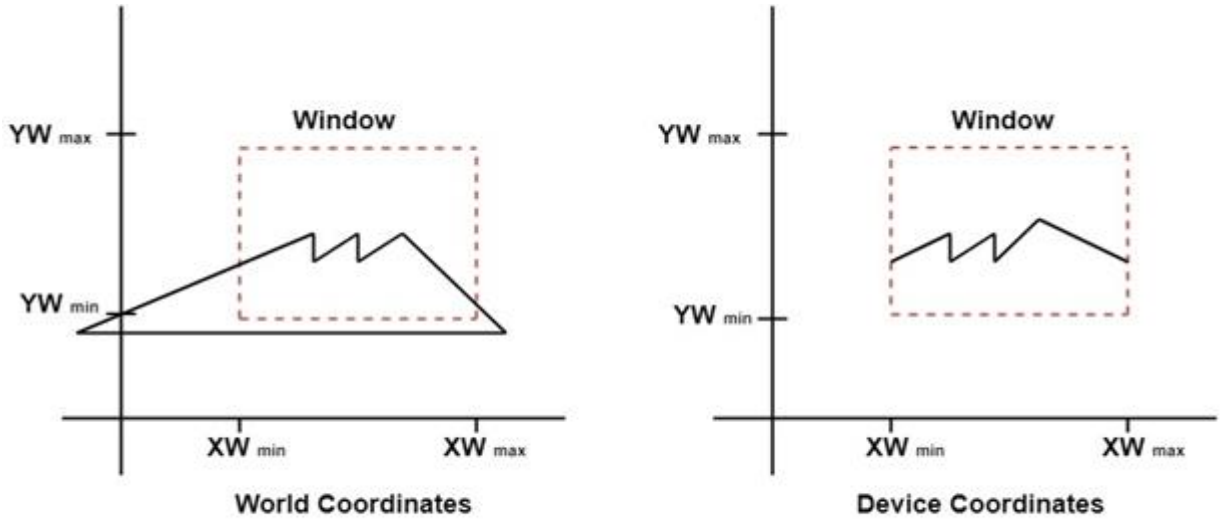



Fig: A viewing transformation using standard rectangles for the window and viewport.

**Viewing transformation in several steps:**

First, we construct the scene in world coordinate using the output primitives and attributes.

To obtain a particular orientation, we can set up a 2-D viewing coordinate system in the window coordinate plane and define a window in viewing coordinates system.

Once the viewing frame is established, are then transform description in world coordinates to viewing coordinates.

Then, we define viewport in normalized coordinates (range from 0 to 1) and map the viewing coordinates description of the scene to normalized coordinates.

At the final step, all parts of the picture that (i.e., outside the viewport are dipped, and the contents are transferred to device coordinates).

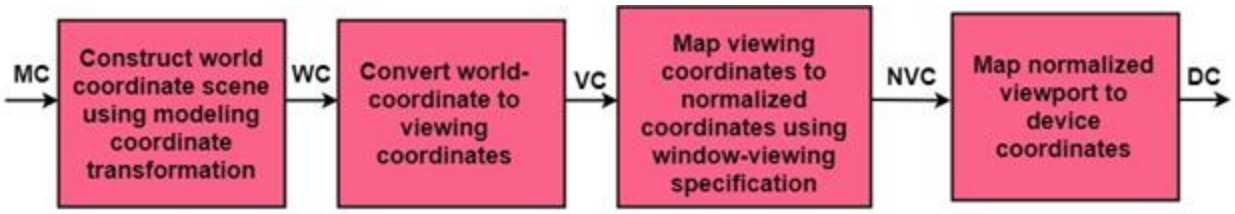



Fig: The two-dimensional viewing-transformation.

**By changing the position of the viewport:**We can view objects at different locations on the display area of an output device as shown in fig:

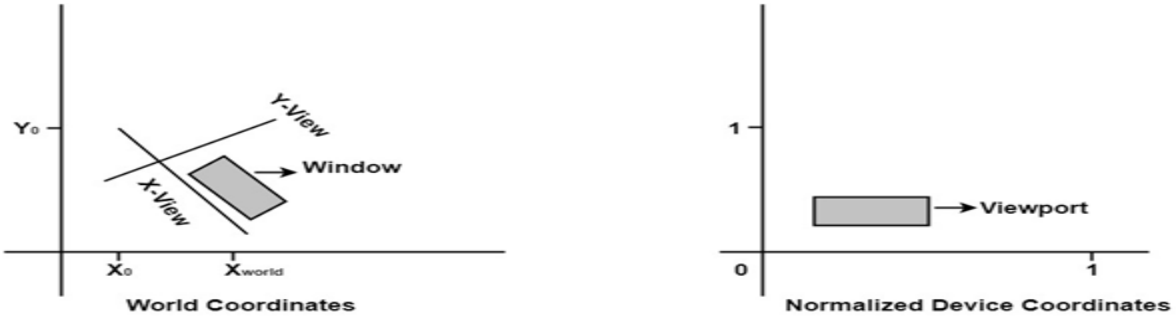



Fig:Setting up a rotated world window and corresponding normalized coordinate viewport.

**By varying the size of viewports:** We can change the size and proportions of displayed objects. We can achieve zooming effects by successively mapping different-sized windows on a fixed-size viewport.

As the windows are made smaller, we zoom in on some part of a scene to view details that are not shown with larger windows.

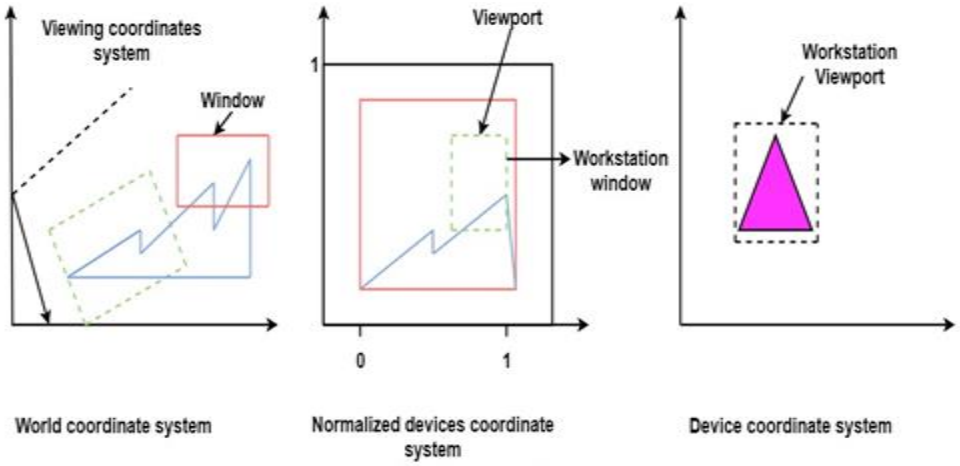



Fig: Zooming effects by mapping different-sized windows on a fixed-size viewport.

## Window to Viewport Co-ordinate Transformation:

Once object description has been transmitted to the viewing reference frame, we choose the window extends in viewing coordinates and selects the viewport limits in normalized coordinates.

Object descriptions are then transferred to normalized device coordinates:

We do this thing using a transformation that maintains the same relative placement of an object in normalized space as they had in viewing coordinates.

It will display at the center of the viewport.

Fig shows the window to viewport mapping. A point at position (xw, yw) in window mapped into position (xv, yv) in the associated viewport.

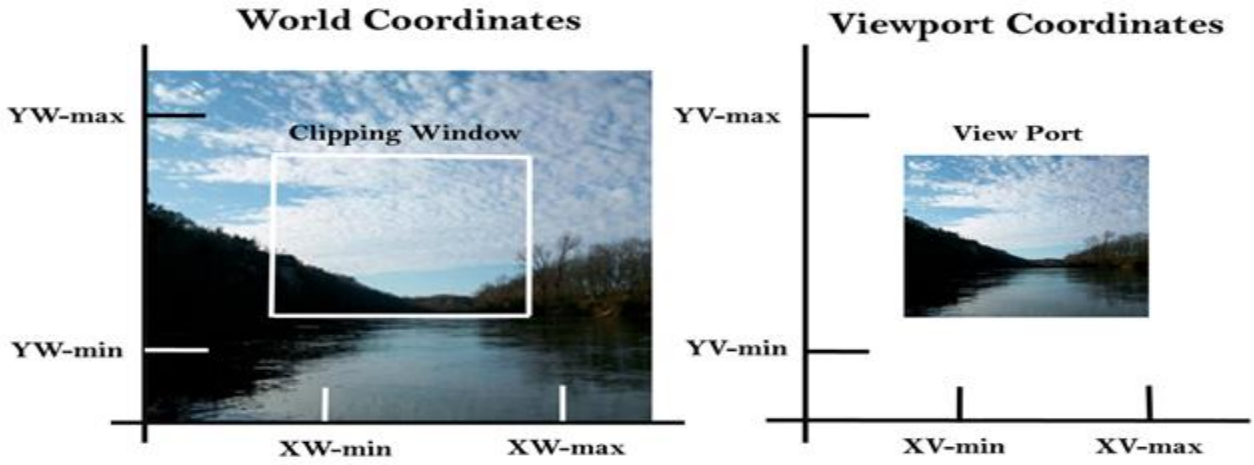



**Fig: Window to viewport mapping**

In order to maintain the same relative placement of the point in the viewport as in the window, we require:

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \quad \text{.....equation 1}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Solving these impressions for the viewport position (xv, yv), we have

$xv=xv_{min}+(xw-xw_{min})sx$
$yv=yv_{min}+(yw-yw_{min})sy$ .....equation 2

Where scaling factors are

$$sx = \frac{xv_{max}-xv_{min}}{xw_{max}-xw_{min}}$$

$$sy = \frac{yv_{max}-yv_{min}}{yw_{max}-yw_{min}}$$

Equation (1) and Equation (2) can also be derived with a set of transformation that converts the window or world coordinate area into the viewport or screen coordinate area. This conversation is performed with the following sequence of transformations:

1. Perform a scaling transformation using a fixed point position ($xw_{min}$,$yw_{min}$) that scales the window area to the size of the viewport.
2. Translate the scaled window area to the position of the viewport. Relative proportions of objects are maintained if the scaling factors are the same (sx=sy).

From normalized coordinates, object descriptions are mapped to the various display devices.

Any number of output devices can we open in a particular app, and three windows to viewport transformation can be performed for each open output device.

This mapping called workstation transformation (It is accomplished by selecting a window area in normalized space and a viewport area in the coordinates of the display device).

As in fig, workstation transformation to partition a view so that different parts of normalized space can be displayed on various output devices).

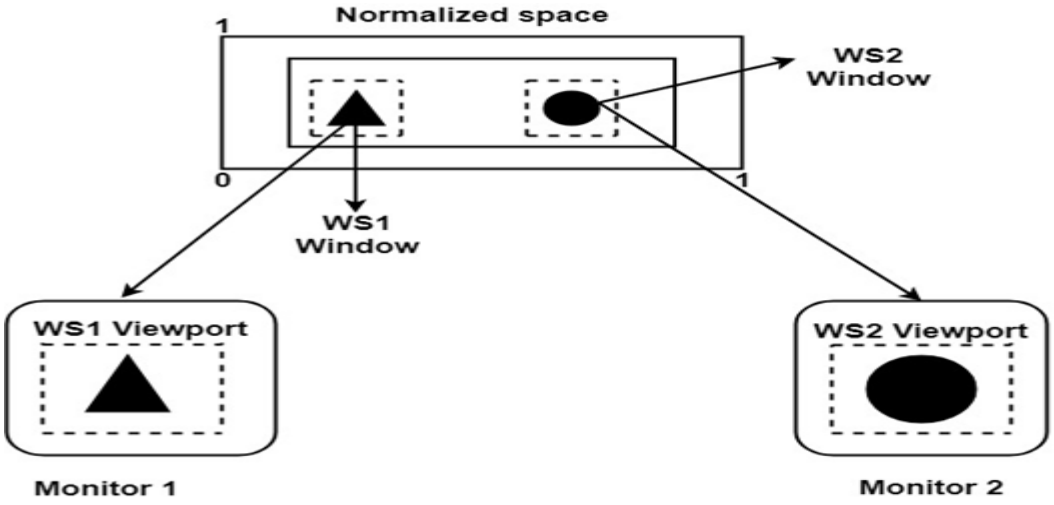
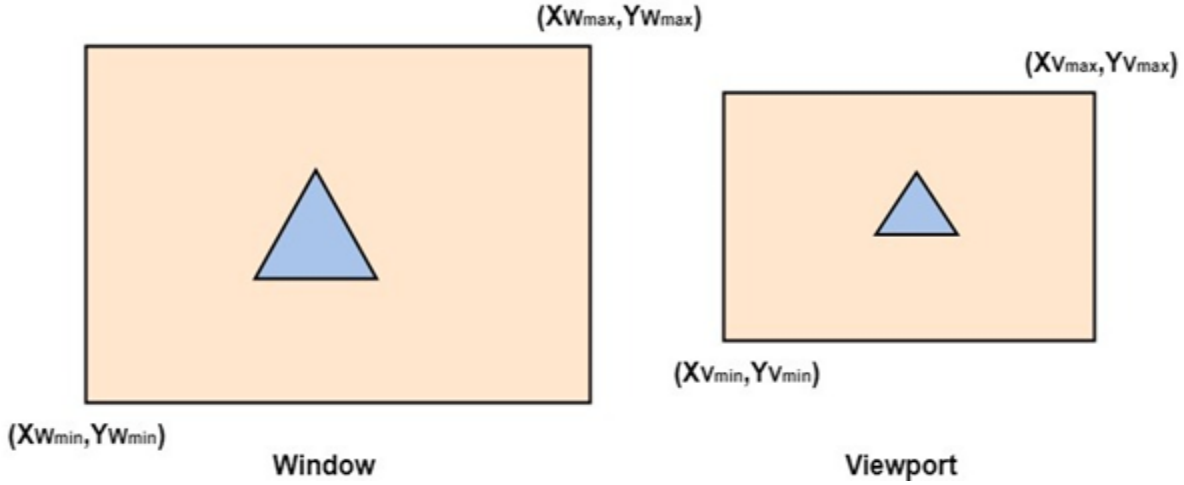



**Fig:Mapping selected parts of a scene in normalized coordinates to different video monitors with workstation transformation.**

Matrix Representation of the above three steps of Transformation:




**Step1:** Translate window to origin 1
$T_x$=-$Xw_{min}$ $T_y$=-$Yw_{min}$

**Step2:** Scaling of the window to match its size to the viewport
$S_x$=($Xy_{max}$-$Xv_{min}$)/($Xw_{max}$-$Xw_{min}$)
$S_y$=($Yv_{max}$-$Yv_{min}$)/($Yw_{max}$-$Yw_{min}$)

**Step3:** Again translate viewport to its correct position on screen.
$T_x$=$Xv_{min}$
$T_y$=$Yv_{min}$

Above three steps can be represented in matrix form:
VT=T * S * $T_1$

T = Translate window to the origin

S=Scaling of the window to viewport size

$T_1$=Translating viewport on screen.

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -Xw_{min} & -Yw_{min} & 1 \end{vmatrix}$$

$$S = \begin{vmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

$$T_1 = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Xv_{min} & Yv_{min} & 1 \end{vmatrix}$$

Viewing Transformation= T * S * $T_1$

**Two dimension viewing function:**

**evaluateViewOrientationMatrix(x0, y0, xv, yverror, viewMatrix)**

Where parameters x0 and y0 are the coordinates of the viewing origin, and parameters xv and yv are the world-coordinate positions for the view up vector. An integer error code is generated if the input parameters are in error; otherwise, the view matrix for the world-to-viewing transformation is calculated. Any number of viewing transformation matrices can be defined in an application. To set up the elements of a window-to-viewport mapping matrix, we invoke the function

**evaluateviewMappinyMatrix (xwmin, xwmax, ywmin, ywmax, xvmin, yvmax, error viewMappingMatrix)**

Here, the window limits in viewing coordinates are chosen with parameters xwmin, xwmax, ywmin, and ywmax; and the viewport limits are set with the normalized coordinate positions **xvmin, xvmax, yvmin ,yvmax**. As with the viewing-transformation matrix, we can construct several window-viewport pairs and use them for projecting various parts of the scene to different areas of the unit square. Next, we can store combinations of viewing and window-viewport mappings for various workstations in a viewing table with

**setVlewRepresentation (ws, viewIndex, viewMatrlx, viewMappingMatrix, xclipmin, xclipmax, yclipmin, yclipmax, clipxy)**

Where parameter wsdesignate the output device (workstation), and parameter viewIndex sets an integer identifier for this particular window-viewport pair. The matrices viewMatrix and viewMappingMatrix can be concatenated and referenced by the viewIndex. Additional clipping limits can also be specified here, but they are usually set to coincide with the viewport boundaries. And parameter clipxy is assigned either the value nochp or the value clip. This allows us to turn off clipping if we want to view the port of the scene outside the viewport. We can also select rloclip to speed up processing when we know that the entire scene is included within the viewport limits. The function

**setviewIndex (viewIndex)**

selects a particular set of options from the viewing table. This view-index selection is then applied to subsequently specified output primitives and associated attributes and generates a display on each of the active workstations.

**setWorkstationWindow (WS, xwswindmir.. xwswixlmax, ywswindrnin. ywswindmax)**

**setworksrationviewport (wsxwsVPortmin, xwsVPortmax, ywsVPortmin, ywsVPortmax)**

where parameter ws gives the workstation number. Window coordinate extents are specified in the range from 0 to 1 (normalized space), and viewport limits are in integer devicecoordinates. If a workstation viewport is not specified, the unit square of the normalized reference frame is mapped onto the largest square area possible on an output device. The coordinate origin of normalized space is mapped to the origin of device coordinates, and the aspect ratio is retained by transforming the unit square onto a square area on the output device.

**Cohen Sutherland line clipping algorithm:**

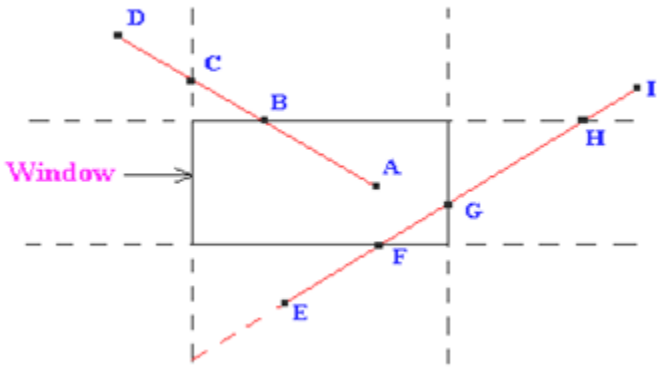
The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping. On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

**Algorithm:** The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's endpoints are tested to see if the line can be trivally accepted or rejected. If the line cannot be trivally accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted.

1. To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.
2. Given a line segment with endpoint P1=(x1,y1) and P2=(x2,y2)
3. Compute the 4-bit codes for each endpoint. If both codes are 0000,(bitwise OR of the codes yields 0000 ) line lies completely inside the window: pass the endpoints to the draw routine. If both codes have a 1 in the same bit position (bitwise AND of the codes is not 0000), the line lies outside the window. It can be trivially rejected.
4. If a line cannot be trivially accepted or rejected, at least one of the two endpoints must lie outside the window and the line segment crosses a window edge. This line must be clipped at the window edge before being passed to the drawing routine. Examine one of the endpoints, say P1=(x1,y1) . Read P1 's 4-bit code in order: Left-to-Right, Bottom-to-Top.
5. When a set bit (1) is found, compute the intersection I of the corresponding window edge with the line from P1 to P2. Replace P2 with I and repeat the algorithm.

**Illustration of Line Clipping**

**Before Clipping**

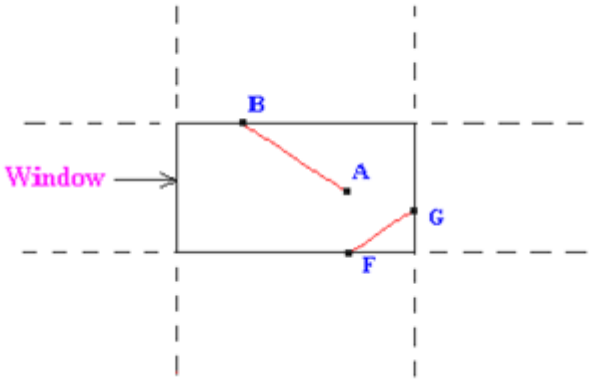



**1. Consider the line segment AD:** Point A has an outcode of 0000 and point D has an outcode of 1001. The logical AND of these outcodes is zero; therefore, the line cannot be trivally rejected. Also, the logical OR of the outcodes is not zero; therefore, the line cannot be trivally accepted. The algorithm then chooses D as the outside point (its outcode contains 1's). By our testing order, we first use the top edge to clip AD

at B. The algorithm then recomputes B's outcode as 0000. With the next iteration of the algorithm, AB is tested and is trivially accepted and displayed.

**2. Consider the line segment EI :**Point E has an outcode of 0100, while point I's outcode is 1010. The results of the trivial tests show that the line can neither be trivally rejected or accepted. Point E is determined to be an outside point, so the algorithm clips the line against the bottom edge of the window. Now line EI has been clipped to be line FI. Line FI is tested and cannot be trivially accepted or rejected. Point F has an outcode of 0000, so the algorithm chooses point I as an outside point since itsoutcode is1010. The line FI is clipped against the window's top edge, yielding a new line FH. Line FH cannot be trivally accepted or rejected. Since H's outcode is 0010, the next iteration of the algorthm clips against the window's right edge, yielding line FG. The next iteration of the algorithm tests FG, and it is trivially accepted and display.

**After Clipping:** After clipping the segments AD and EI, the result is that only the line segment AB and FG can be seen in the window.




**Cyrus Beck Line Clipping Algorithm:**

Cyrus Beck is a line clipping algorithm that is made for convex polygons. It allows line clipping for non-rectangular windows, unlike Cohen Sutherland or **Nicholl Le Nicholl**. It also removes the repeated clipping needed in Cohen Sutherland.

**Input:**

1. **Convex area of interest**
   which is defined by a set of coordinates
   given in a clockwise fashion.
2. **vertices** which are an array of coordinates:
   consisting of pairs (x, y)
3. **n** which is the number of vertices
4. A **line** to be clipped
   given by a set of coordinates.
5. **line** which is an array of coordinates:
   consisting of two pairs, (x0, y0) and (x1, y1)

**Output:**

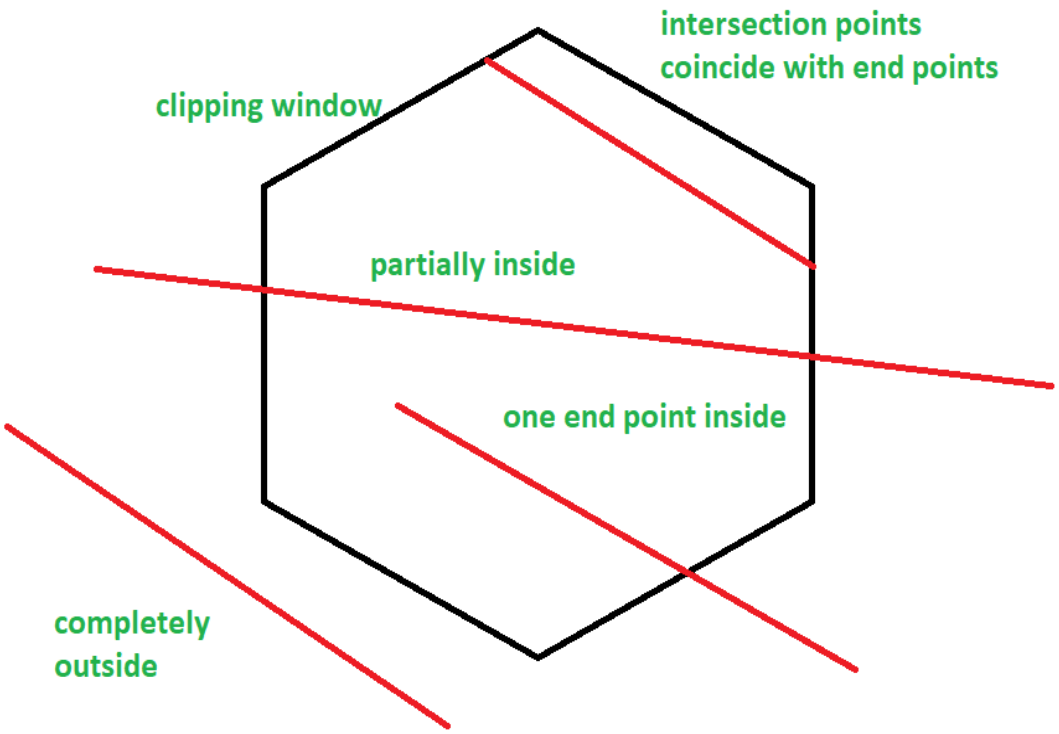
1. Coordinates of line clipping which is the **Accepted clipping**
2. Coordinates (-1, -1) which is the **Rejected clipping**

**Algorithm:**

- Normals of every edge is calculated.

- Vector for the clipping line is calculated.
- Dot product between the difference of one vertex per edge and one selected end point of the clipping line and the normal of the edge is calculated (for all edges).
- Dot product between the vector of the clipping line and the normal of edge (for all edges) is calculated.
- The former dot product is divided by the latter dot product and multiplied by -1. This is 't'.
- The values of 't' are classified as entering or exiting (from all edges) by observing their denominators (latter dot product).
- One value of 't' is chosen from each group, and put into the parametric form of a line to calculate the coordinates.
- If the entering 't' value is greater than the exiting 't' value, then the clipping line is rejected.

**Cases:**




1. **Case 1:** The **line is partially inside** the clipping window:
2. $0 < t_E < t_L < 1$
3. 
4. where $t_E$ is 't' value for entering intersection point
5. $t_L$ is 't' value for exiting intersection point
6. **Case 2:** The **line has one point inside or both sides inside** the window **or the intersection points are on the end points of the line**:
   $0 \leq t_E \leq t_L \leq 1$
7. **Case 3:** The line is **completely outside** the window:
   $t_L < t_E$

**Sutherland –hodgeman polygon clipping algo:**

We can correctly clip a polygon by processing the polygon bound jry as a whole against each window edge. This could be accomplished by processing all polygon vertices against each clip rectangle boundary in turn. Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices. The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper, as in Fig. 6-19. At each step, a new sequence of output vertices is generated and passed to the next window boundary clipper.

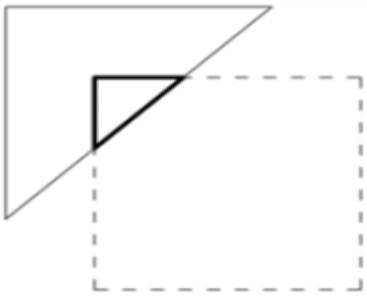


*Figure 1*

It is often necessary, particularly in graphics applications, to "clip" a given polygon with another. Figure 1 shows an example. In the figure, the clipping polygon is drawn with a dashed line, the clipped polygon with a regular line, and the resulting polygon is drawn with a heavy line. In this article we'll look at the particular case where the clipping polygon is a rectangle which is oriented parallel with the axes. For this case, the Sutherland-Hodgeman algorithm is often employed. This is the algorithm that we will explore.

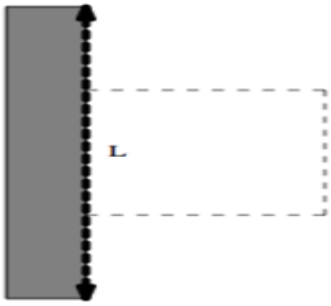



*Figure 2*

**The Sutherland-Hodgeman Algorithm**

The Sutherland-Hodgeman polygon clipping algorithm is relatively straightforward and is easily implemented in C. It does have some limitations, which we'll explore later. First, let's see how it works. For each of the four sides of the clipping rectangle, consider the line *L* through the two points which define that side. For each side, this line creates two planes, one which includes the clipping rectangle and one which does not. We'll call the one which does not the "clipping plane" (Figure 2).

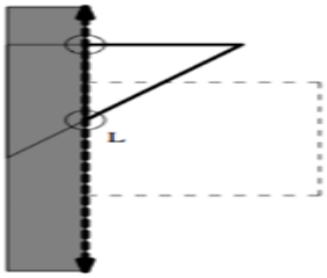



*Figure 3*

For each of the four clipping planes, we must remove any vertices of the clipped polygon which lie inside the plane and create new points where the segments associated with these vertices cross the lineL (Figure 3).

**After clipping each of the four planes, we are left with the clipped polygon.**

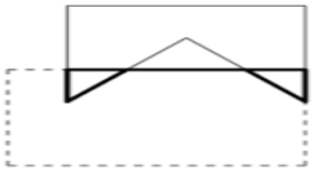


*Figure 4*

**Limitations :**

This algorithm always produces a single output polygon, even if the clipped polygon is concave and arranged in such a way that multiple output polygons might reasonably be expected. Instead, the polygons will be linked with overlapping segments along the edge of the clipping rectangle (Figure 4). This may or may not be the desired result, depending on your application

**Polygon Surface:**

Polygon and quadric surfaces provide precise descriptions for simple Euclidean objects such as polyhedrons and ellipsoids; spline surfaces end construction techniques are useful for designing aircraft wings, gears, and other engineering structures with curved surfaces; procedural methods, such as fractal constructions and particle systems, allow us to give accurate representations for clouds, clumps of grass, and other natural objects; physically based modeling methods using systems of interacting forces can be used to describe the non-rigid behavior of a piece of cloth or a glob of jello; octree encodings are used to represent internal features of objects, such as those obtained from medical CT images; and isosurface displays, volume renderings, and other visualization techniques are applied to three-dimensional discrete data sets to obtain visual representations of the data.

Representation schemes for solid objects are often divided into two broad categories, although not all representations fall neatly into one or the other of these two categories. Boundary representations (B-reps) describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. Typical examples of boundary representations are polygon facets and spline patches. Space-partitioning representations are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes). A common space-partitioning description for a three-dimensional object is an octree representation. In this chapter, we consider the features of the various representation schemes and how they are used in applications.

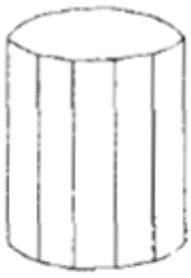


*Figure 10-1*
Wireframe representation of a cylinder with back (hidden) lines removed.

The most commonly used boundary representation for a three-dimensional graphics 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 simplifies and speeds up the surface rendering and display of objects, since all surfaces are described with linear equations. For this reason, polygon descriptions are often referred to as "standard graphics objects." In some cases, a polygonal representation is the only one available, but many packages allow objects to be described with other schemes, such as spline surfaces, that are then converted to polygonal representations for processing.

A polygon representation for a polyhedron precisely defines the surface features of the object. But for other objects, surfaces are tessellated (or tiled) to produce the polygon-mesh approximation. In Fig. 10-1, the surface of a cylinder is represented as a polygon mesh. Such representations are common in design

and solid- 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 eliminateor reduce the presence of polygon edge boundaries. And the polygon-mesh approximation to a curved surface can be improved by dividing the surface into smaller polygon facets.

**Quadric Surface:**

A frequently used class of objects is the quadric surfaces, which are described with second-degree equations (quadratics). They include spheres, ellipsoids, tori,paraboloids, and hyperboloids. Quadric surfaces, particularly spheres and ellipsoids, are common elements of graphics scenes, and they are often available in graphics packages as primitives horn which more complex objects can be constructed.

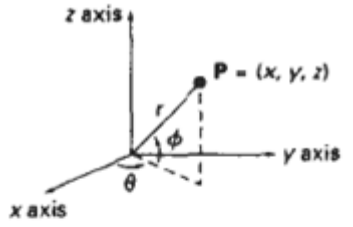



*Figure 10-8*
Parametric coordinate position $(r, \theta, \phi)$ on the surface of a sphere with radius $r$.

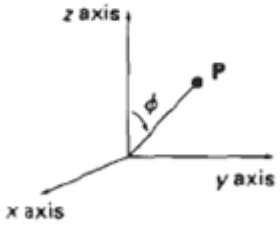



*Figure 10-9*
Spherical coordinate parameters $(r, \theta, \phi)$, using colatitude for angle $\phi$.

**Sphere:**

In Cartesian coordinates, a spherical surface with radius r centered on the coordinate origin is defined as the set of points (x, y, z) that satisfy the equation

$X^2\ y^2\ z^2 = r^2$

We can also describe the spherical surface in parametric form, using latitude and longitude angles (Fig. 10-8):

$$x = r\cos\phi\cos\theta, \qquad -\pi/2 \le \phi \le \pi/2$$
$$y = r\cos\phi\sin\theta, \qquad -\pi \le \theta \le \pi$$
$$z = r\sin\phi$$

The parametric representation inEqs. 10-8 provides a symmetric range for the angular parameters **θ**and φ. alternatively, we could write the parametric colatitude (Fig. 10-9). Then, φis defined over the range 0≤φ≤ π and **θ**is often taken in the range 0 ≤**θ**≤ **2**π. We could also set up the representation using parameters u and v, defined over the range from 0 to 1 by substituting φ= πu and **θ**= 2πv.

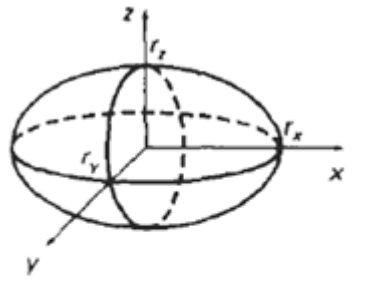



*Figure 10-10*
An ellipsoid with radii $r_x$, $r_y$, and $r_z$ centered on the coordinate origin.

**Ellipsoid:** An ellipsoidal surface can be described as an extension of a spherical surface, where the radii in three mutually perpendicular directions can have different values (Fig. 10-10). The Cartesian representation for points over the surface of an ellipsoid centered on the origin is

$$\left(\frac{x}{r_x}\right)^2 + \left(\frac{y}{r_y}\right)^2 + \left(\frac{z}{r_z}\right)^2 = 1$$

And a parametric representation for the ellipsoid in terms of the latitude angle 4 and the longitude angle 0 in Fig. 10-8 is

$$x = r_x\cos\phi\cos\theta, \qquad -\pi/2 \le \phi \le \pi/2$$
$$y = r_y\cos\phi\sin\theta, \qquad -\pi \le \theta \le \pi$$
$$z = r_z\sin\phi$$

**Spline Representation:**
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 it in position on

the drafting table as the curve is drawn. The term spline curve originally referred to a curve drawn in this manner. We can mathematically describe such a curve with a piecewise cubic

*Figure 10-19*
A set of six control points interpolated with piecewise continuous polynomial sections.

*Figure 10-20*
A set of six control points approximated with piecewise continuous polynomial sections

Polynomial function whose first and second derivatives are continuous across the various curve sections. In computer graphics, the term spline curve now refers to any composite curve formed with polynomial sections satisfying specified continuity conditions at the boundary of the pieces. A spline surface can be described with two sets of orthogonal spline curves. There are several different kinds of spline specifications that are used in graphics applications. Each individual specification simply refers to a particular type of polynomial with certain specified boundary conditions.

Splines are used, n graphics applications to design curve and surface shapes, to digitize drawings for computer storage, and to specify animation paths for the objects or the camera in a scene. Typical CAD applications for splines include the design of automobile bodies, aircraft and spacecraft surfaces, and ship hulls.

**Interpolation and approximation Splines**

We specify a spline curve by givinga set of coordinate positions, called control points, which indicates the general shape of the curve. These control points are then fitted pricewise continues paranormal functions in one of two ways. When polynomial sections are fitted so that the curve passes through each control point, as in Fig. 10-19, the resulting curve is said to interpolate the set of control points. On the other hand, 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 (Fig. 14-20),

Interpolation curves are commonly used to digitize drawings or to specify animation paths. Approximation curves are primarily used as design tools to structure object surfaces Figure 10-21 shows an approximation spline surface credited for a design applications. Straight lines connect the control-point positions above the surface.

A spline curve 1s defined, modified, and manipulated with operations on the control points. By interactively selecting spatial positions for the control points, a designer can set up an initial curve. After the polynomial fit is displayed for a given set of control points, the designer can then reposition some or all of the control points to restructure the shape of the curve. In addition, the curve can be translated, rotated, or scaled with transformations applied to the control points. CAD packages can also insert extra control points to aid a designer in adjusting the curve shapes.

The convex polygon boundary that encloses a set of control points is called the convex hull. One way to envision the shape of a convex hull is to imagine a rubber band stretched around the positions of the control points so that each control point is either on the perimeter of the hull or inside it (Fig. 10-22). Convex hulls provide a measure for the deviation of a curve or surface from the region bounding the control points. Some splines are bounded by the convex hull, thus ensuring that the polynomials smoothly follow the control points without erratic oscillations. Also the polygon region inside the convex hull is useful in some algorithms as a clipping region.

A polyline connecting the sequence of control points for an approximation spline is usually displayed to remind a designer of the control-point ordering. This set of connected line segments is often referred to as the control graph of the curve. Other names for the series of straight-line sections connecting the control points in the order specified are control polygon and characteristic polygon.

**Hermite curve**

A Hermite curve is a spline where every piece is a third degree polynomial defined in Hermite form: that is, by its values and initial derivatives at the end points of the equivalent domain interval. Cubic Hermite splines are normally used for interpolation of numeric values defined at certain dispute values x1,x2,x3, …..xn,to achieve a smooth continuous function. The data should have the preferred function value and derivative at each Xk. The Hermite formula is used to every interval (Xk, Xk+1) individually. The resulting spline become continuous and will have first derivative.

Cubic polynomial splines are specially used in computer geometric modeling to attain curves that pass via defined points of the plane in 3D space. In these purposes, each coordinate of the plane is individually interpolated by a cubic spline function of a divided parameter‘t’.

Cubic splines can be completed to functions of different parameters, in several ways. Bicubic splines are frequently used to interpolate data on a common rectangular grid, such as pixel values in a digital picture. Bicubic surface patches, described by three bicubic splines, are an necessary tool in computer graphics. Hermite curves are simple to calculate but also more powerful. They are used to well interpolate between key points.

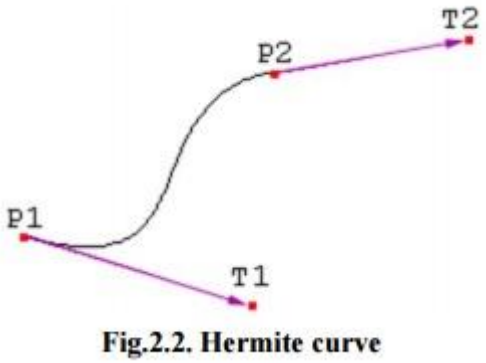


Fig.2.2. Hermite curve

**Fig.2.2. Hermite curve**

The following vectors needs to compute a Hermite curve:

- P1: the start point of the Hermite curve
- T1: the tangent to the start point
- P2: the endpoint of the Hermite curve
- T2: the tangent to the endpoint

These four vectors are basically multiplied with four Hermite basis functions h1(s), h2(s), h3(s) and,h4(s) and added together.

h1(s) = 2s3 - 3s2 + 1 h2(s) = -2s3 + 3s2

h3(s) = s3 - 2s2 + s h4(s) = s3 - s2

Figure 2.3 shows the functions of Hermite Curve of the 4 functions (from left to right: h1, h2, h3, h4).

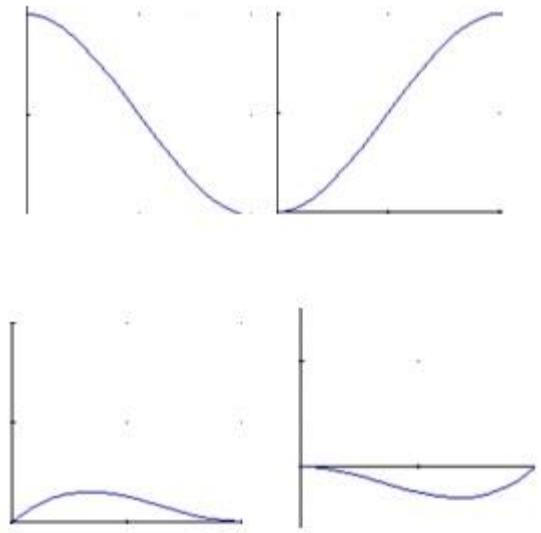


Fig.2.3. Functions of Hermite curve

**Fig.2.3. Functions of Hermite curve**

A closer view at functions ‘h1’and ‘h2’,the result shows that function ‘h1’starts at one and goes slowly to zero and function ‘h2’starts at zero and goes slowly to one.

At the moment, multiply the start point with function ‘h1’and the endpoint with function ‘h2’. Let s varies from zero to one to interpolate between start and endpoint of Hermite Curve. Function

‘h3’and function ‘h4’are used to the tangents in the similar way. They make confident that the Hermite curve bends in the desired direction at the start and endpoint.

**Bezier curve**

Bezier curves are extensively applied in CAD to model smooth curves. As the curve is totally limited in the convex hull of its control points P0, P1,P2 & P3, the points can be graphically represented and applied to manipulate the curve logically. The control points P0 and P3 of the polygon lie on the curve (Fig.2.4.). The other two vertices described the order, derivatives and curve shape. The Bezier curve is commonly tangent to first and last vertices.

Cubic Bezier curves and Quadratic Bezier curves are very common. Higher degree Bezier curves are highly computational to evaluate. When more complex shapes are required, Bezier curves in low order are patched together to produce a composite Bezier curve. A composite Bezier curve is usually described to as a ‘path’in vector graphics standards and programs. For smoothness assurance, the control point at which two curves meet should be on the line between the two control points on both sides.

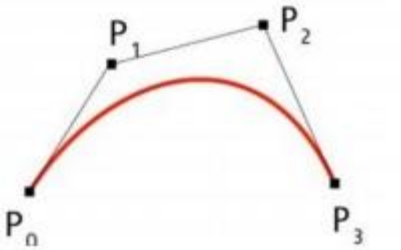


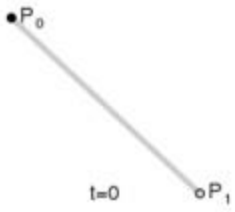

Fig.2.4. Bezier curve

**Fig.2.4. Bezier curve**

A general adaptive method is recursive subdivision, in which a curve's control points are verified to view if the curve approximates a line segment to within a low tolerance. If not, the curve is further divided parametrically into two segments, $0 \leq t \leq 0.5$ and $0.5 \leq \leq t1$, and the same process is used recursively to each half. There are future promote differencing techniques, but more care must be taken to analyze error transmission.

Analytical methods where a Bezier is intersected with every scan line engage finding roots of cubic polynomials and having with multiple roots, so they are not often applied in practice. A Bezier curve is described by a set of control points P0 through Pn, where 'n'is order of curve. The initial and end control points are commonly the end points of the curve; but, the intermediate control points normally do not lie on the curve.

**(i) Linear Bezier curves**




2.5. Linear Bezier curve

$$B(t) = P_0 + t(P_1 - P_0) = (1-t)P_0 + tP_1 \ , t \in [0,1]$$

**2.5. Linear Bezier curve**

As shown in the figure 2.5, the given points P0 and P1, a linear Bezier curve is merely a straight line between those two points. The Bezier curve is represented by And it is similar to linear interpolation.

**(ii) Quadratic Bezier curves**

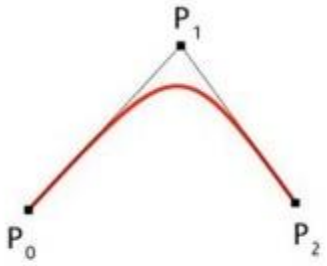



Fig.2.6. Quadratic Bezier curve

**Fig.2.6. Quadratic Bezier curve**

As shown in the figure 2.6, a quadratic Bezier curve is the path defined by the function B(*t*), given points P0, P1, and P2,

$$\mathbf{B}(t) = (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \ , t \in [0,1],$$

This can be interpreted as the linear interpolate of respective points on the linear Bezier curves from P0 to P1 and from P1 to P2 respectively. Reshuffle the preceding equation gives:

$$\mathbf{B}(t) = (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \ , t \in [0,1].$$

The derivative of the Bezier curve with respect to the value 't'is

$$\mathbf{B}'(t) = 2(1-t)(\mathbf{P}_1 - \mathbf{P}_0) + 2t(\mathbf{P}_2 - \mathbf{P}_1).$$

From which it can be finished that the tangents to the curve at P0 and P2 intersect at P1. While 't'increases from zero to one, the curve departs from P0 in the direction of P1, then turns to land at P2 from the direction of P1.

The following equation is a second derivative of the Bezier curve with respect to 't':

$$\mathbf{B}''(t) = 2(\mathbf{P}_2 - 2\mathbf{P}_1 + \mathbf{P}_0).$$

A quadratic Bezier curve is represent a parabolic segment. Since a parabola curve is a conic section, a few sources refer to quadratic Beziers as 'conic arcs'.

(iii) Cubic Bezier curves

As shown in figure 2.7, four control points P0, P1, P2 and P3 in the higher-dimensional space describe as a Cubic Bezier curve. The curve begins at P0 going on the way to P1 and reaches at P3 coming from the direction of P2. Typically, it will not pass through control points P1 / P2, these points are only there to

give directional data. The distance between P0 and P1 determines 'how fast'and 'how far'the curve travels towards P1 before turning towards P2.

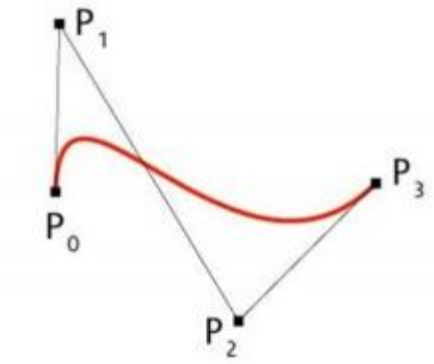



Fig.2.7. Cubic Bezier curve

**Fig.2.7. Cubic Bezier curve**

The function **B Pi**, **P**j, **P**k (t) for the quadratic Bezier curve written by points **P**i, **P**j, and **P**k, the cubic Bezier curve can be described as a linear blending of two quadratic Bezier curves:

$$B(t) = (1-t)B_{P_0,P_1,P_2}(t) + tB_{P_1,P_2,P_3}(t)\ ,\ t \in [0,1].$$

The open form of the curve is:

$$B(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3\ ,\ t \in [0,1].$$

For several choices of P1 and P2 the Bezier curve may meet itself.

Any sequence of any four dissimilar points can be changed to a cubic Bezier curve that goes via all four points in order. Given the beginning and ending point of a few cubic Bezier curve, and the points beside the curve equivalent to t = 1/3 and t = 2/3, the control points for the original Bezier curve can be improved.

The following equation represent first derivative of the cubic Bezier curve with respect to t:

$$B'(t) = 3(1-t)^2(P_1 - P_0) + 6(1-t)t(P_2 - P_1) + 3t^2(P_3 - P_2).$$

The following equation represent second derivative of the Bezier curve with respect to t:

**1. Properties Bezier curve**

· The Bezier curve starts at P0 and ends at P*n*; this is known as 'endpoint interpolation'property.

- The Bezier curve is a straight line when all the control points of a cure are collinear.

- The beginning of the Bezier curve is tangent to the first portion of the Bezier polygon.

- A Bezier curve can be divided at any point into two sub curves, each of which is also a Bezier curve.

- A few curves that look like simple, such as the circle, cannot be expressed accurately by a Bezier; via four piece cubic Bezier curve can similar a circle, with a maximum radial error of less than one part in a thousand (Fig.2.8).

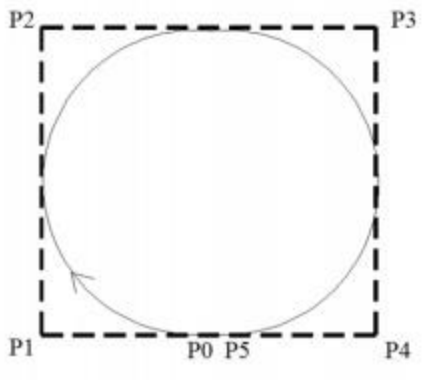



Fig.2.8. Circular Bezier curve

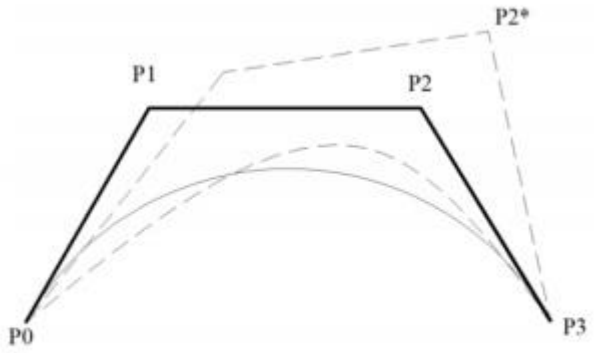
**Fig.2.8. Circular Bezier curve**

- Each quadratic Bezier curve is become a cubic Bezier curve, and more commonly, each degree ‘n’Bezier curve is also a degree ‘m’curve for any $m > n$.

Bezier curves have the different diminishing property. A Bezier curves does not ‘ripple’more than the polygon of its control points, and may actually ‘ripple’lss than that.

- Bezier curve is similar with respect to t and (1-t). This represents that the sequence of control points defining the curve can be changes without modify of the curve shape.

- Bezier curve shape can be edited by either modifying one or more vertices of its polygon or by keeping the polygon unchanged or simplifying multiple coincident points at a vertex (Fig .2.19).




**2.9. Bezier curve shape**

## 2. Construction of Bezier curves

(i) Linear curves:

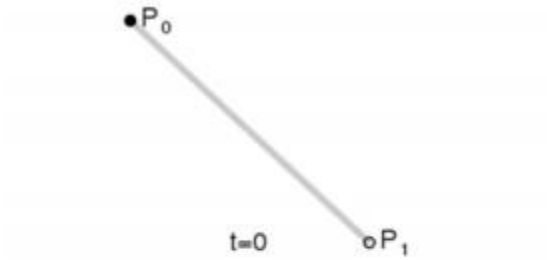



**Fig.2.10. Construction of linear Bezier curve**

**Fig.2.10. Construction of linear Bezier curve**

The figure 2.10 shows the function for a linear Bezier curve can be via of as describing how far B(*t*) is from P0 to P1 with respect to 't'. When t equals to 0.25, B(*t*) is one quarter of the way from point P0 to P1. As 't'varies from 0 to 1, B(*t*) shows a straight line from P0 to P1.

(ii) Quadratic curves

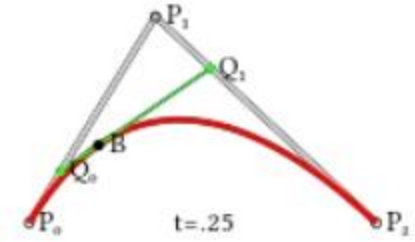



**Fig.2.11. Construction of linear Quadratic curve**

As shown in figure 2.11, a quadratic Bezier curves one can develop by intermediate points Q0 and Q1 such that as 't'varies from 0 to 1:

· Point Q0 (t) modifying from P0 to P1 and expresses a linear Bezier curve.

· Point Q1 (t) modifying from P1 to P2 and expresses a linear Bezier curve.

· Point B (t) is interpolated linearly between Q0(t) to Q1(t) and expresses a quadratic Bezier curve.

(iii) Higher-order curves

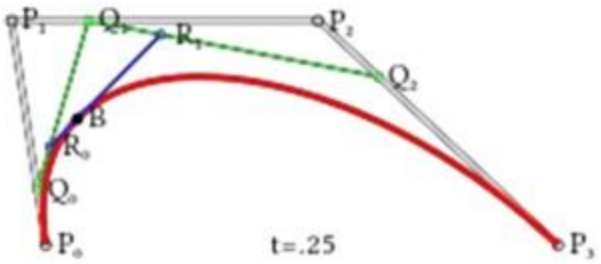



**Fig.2.12. Construction of Higher-order curve**

**Fig.2.12. Construction of Higher-order curve**

As shown in figure 2.12, a higher-order curves one requires correspondingly higher intermediate points. For create cubic curves, intermediate points Q0, Q1, and Q2 that express as linear

Bezier curves, and points R0 and R1 that express as quadratic Bezier curves.

**3. Rational Bezier curve**

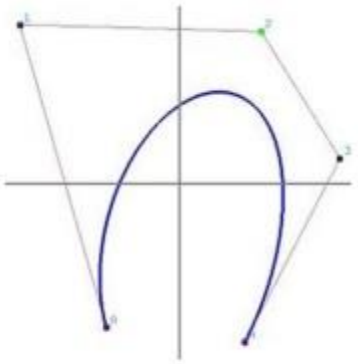


**Fig.2.13. Rational Bezier Curve**

**Fig.2.13. Rational Bezier Curve**

The rational Bezier curve includes variable weights (w) to provide closer approximations to arbitrary shapes. For Rational Bezier Curve, the numerator is a weighted Bernstein form Bezier and the denominator is a weighted sum of Bernstein polynomials. Rational Bezier curves can be used to signify segments of conic sections accurately, including circular arcs (Fig.2.13).

**B-Spline curves:**

These are the most widely used class of approximating splines. B-splines have two advantages over B6zier splines: (1) the degree of a B-spline polynomial can be set independently of the number of control points (with certain limitations), and (2) B-splines allow local control over the shape of a spline curve or surface The trade-off is that &splines are more complex than Bezier splines.

**B-Spline Curves**: We can write a general expression for the calculation of coordinate positions along a B-spline curve in a blending-function formulation as

$$\mathbf{P}(u) = \sum_{k=0}^{n} \mathbf{p}_k B_{k,d}(u), \qquad u_{min} \le u \le u_{max}, \qquad 2 \le d \le n+1$$

where the pk are an input set of n 1 control points. There are several differences between this B-spline formulation and that for Bezier splines. The range of parameter u now depends on how we choose the Bspline parameters. And the B spline blending functions Bbd are polynomials of degree d - 1, where parameter d can be chosen to be any integer value in the range from 2 up to the number of control points, n 1. (Actually, we can also set the value of d at 1, but then our "curve" is just a point plot of the control points.) Local control for B splines is achieved by defining the blending functions over subintervals of the total range of u.

Blending functions for B-spline curves are defined by the Cox-deBoor recursion formulas:

$$B_{k,1}(u) = \begin{cases} 1, & \text{if } u_k \le u < u_{k+1} \\ 0, & \text{otherwise} \end{cases}$$

$$B_{k,d}(u) = \frac{u - u_k}{u_{k+d-1} - u_k} B_{k,d-1}(u) + \frac{u_{k+d} - u}{u_{k+d} - u_{k+1}} B_{k+1,d-1}(u)$$

where each blending function is defined over d subintervals of the total range of u. The selected set of subinterval endpoints u, is referred to as a knot vector. We can choose any values for the subinterval endpoints satisfying the relation u1 ≤ uj 1,.Values for umax, and umin, then depend on the number of control points we select, the value we choose for parameter d, and how we set up the subintervals (knot vector). Since it is possible to choose the elements of the knot vector so that the denominators in the previous calculations can have a value of 0, this formulation assumes that any terms evaluated as 0/0 are to be assigned the value 0.

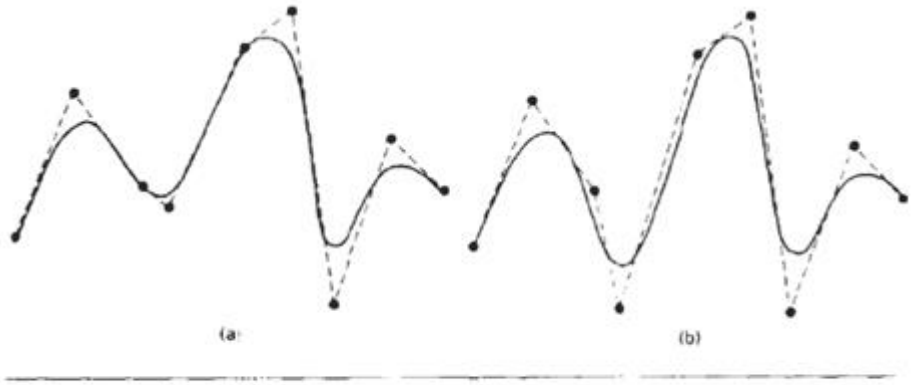



*Figure 10-41*
Local modification of a B-spline curve. Changing one of the control points in (a) produces curve (b), which is modified only in the neighborhood of the altered control point.

Figure 10-41 demonstrates the local-control characteristics of B splines. In addition to local control, B-splines allow us to vary the number of control points used to design a curve without hanging the degree of the polynomial. Also, any number of control points can be added or modified to manipulate curve shapes. Similarly, we can increase the number of values in the knot vector to aid in curve design. When we do this, however, we also need to add control points since the size of the knot vector depends on parameter n.

**B-spline curves have the following properties.**

1. The polynomial curve has degree d - 1 and $C^{d-2}$ Continuity over the range of u.

2. For n 1 control points, the curve is described with n 1 blending functions.

3. Each blending function $B_{k,d}$is defined over d subintervals of the total range of u, starting at knot value uk.
4. The range of parameter u 1s divided into n d subintervals by the n d 1 values specified in the knot vector.

5. With knot values labeled as [u1 u1 . . . . . $u_{n\ 1}$,] the resulting B-spline curve is defined only in the interval from knot value $u_{d-1}$ , up to knot value $u_{n-1.}$

6. Each section of the spline curve (between two successive knot values) is influenced by d control points.

7. Any one control point can affect the shape of at most d curve sections.

In addition, a B-spline curve lies within the convex hull of at most d 1 control points, so that B-splines are tightly bound to the input positions. For any value of u in the interval from knot value $u_{d-1}$ to $u_{n\ 1}$the sum over all basis functions is 1:

Given the control-point positions and the value of parameter d, we then need to specify the knot values to obtain the blending functions using the recurrence relations 10-55.

**B-Spline and Bezier Curves:**

**1. Spline :**
A spline curve is a mathematical representation for which it is easy to build an interface that will allow a user to design and control the shape of complex curves and surfaces.

**2. B-Spline :**
B-Spline is a basis function that contains a set of control points. The B-Spline curves are specified by Bernstein basis function that has limited flexibility.

**3. Bezier :**
These curves are specified with boundary conditions, with a characterizing matrix or with blending function. A Bezier curve section can be filled by any number of control points. The number of control points to be approximated and their relative position determine the degree of Bezier polynomial.

**Difference between Spline, B-Spline and Bezier Curves :**

| Spline | B-Spline | Bezier |
|---|---|---|
| A spline curve can be specified by giving a specified set of coordinate positions, called control points which indicate the general shape of the curve. | The B-Spline curves are specified by Bernstein basis function that has limited flexibiity. | The Bezier curves can be specified with boundary conditions, with a characterizing matrix or with blending function. |
| It follows the general shape of the curve. | These curves are a result of the use of open uniform basis function. | The curve generally follows the shape of a defining polygon. |
| Typical CAD application for spline include the design of automobile bodies, aircraft and spacecraft surfaces and ship hulls. | These curves can be used to construct blending curves. | These are found in painting and drawing packages as well as in CAD applications. |
| It possess a high degree of smoothness at the places where the polynomial pieces connect. | The B-Spline allows the order of the basis function and hence the degree of the resulting curve is independent of number of vertices. | The degree of the polynomial defining the curve segment is one less than the number of defining polygon point. |
| A spline curve is a mathematical representation for which it is easy to build an interface that will allow a user to design and control the shape of complex curves and surfaces. | In B-Spline, there is local control over the curve surface and the shape of the curve is | |

**Basic Illumination Models:**

The empirical models described in this section provide simple and fast methods for calculating surface intensity at a given point, and they produce reasonably good results for most scenes. Lighting calculations are based on the optical properties of surfaces, the background lighting conditions, and the light-source specifications. Optical parameters are used to set surface properties, such as glossy, matte, opaque, and transparent. This controls the amount of reflection and absorption of incident light. All light sources are considered to be point sources, specified with a coordinate position and an intensity value (color).

**Ambient Light**: A surface that is not exposed directly to a light source still will be visible it nearby objects are illuminated. In our basic illumination model, we can set a general level of brightness for a scene. This is a simple way to model the combination of light reflections from various surfaces to produce a uniform illumination called the ambient light, or background light. Ambient light has no spatial or directional characteristics. The amount of ambient light incident on each object is a constant for all surfaces and over all directions.

We can set the level for the ambient light in a scene with parameter $I_a$, and each surface is then illuminated with this constant value. The resulting reflected light is a constant for each surface, independent of the viewing direction and the spatial orientation of the surface. But the intensity of the reflected light for each surface depends on the optical properties of the surface; that is, how much of the incident energy is to be reflected and how much absorbed.

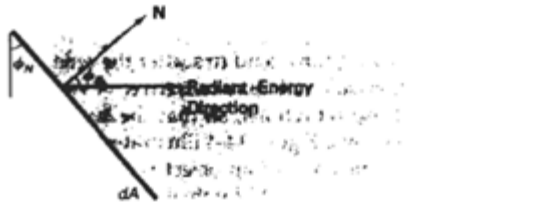



*Figure 14-6*
Radiant energy from a surface area $dA$ in direction $\phi_N$ relative to the surface normal direction.

**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 the viewing direction. The fractional mount of the incident light that is diffusely reflected can be set for each surface with parameter $k_d$, the diffuse-reflection coefficient, or diffuse reflectivity. Parameter $k_d$ is assigned a constant value in the interval 0 to 1, according to the reflecting properties we want the surface to have. If we want a highly reflective surface, we set the value of $k_d$ near 1. This produces a bright surface with the intensity of the reflected light near that of the incident light. To simulate a surface that absorbs most of the incident light, we set the reflectivity to a value near 0. Actually, parameter $k_d$is a function of surface color, but for the time being we will assume $k_d$ is a constant.

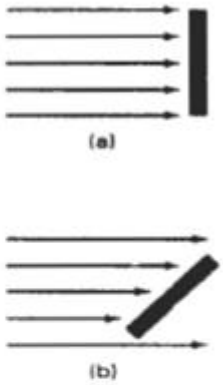



*Figure 14-7*
A surface perpendicular to the direction of the incident light (a) is more illuminated than an equal-sized surface at an oblique angle (b) to the incoming light direction.

If a surface is exposed only to ambient light, we can express the intensity of the diffuse reflection at any point on the surface as

$$I_{\text{ambdiff}} = k_d I_a$$

Since ambient light produces a flat uninteresting shading for each surface (Fig. 14-19(b)), 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.

We can model the diffuse reflections of illumination from a point source in a similar way. That is, we assume that the diffuse reflections from the surface are s c a t t e d with equal intensity in all directions, independent of the viewing dimension. Such surfaces are sometimes referred to as ideal diffuse reflectors. They are also called Lambertian reflectors, since radiated light energy from any point on the surface is governed by Iambert's cosine law. This law states that the radiant energy from any small surface area dA in any direction $\varphi_N$relative to the surface normal is proportional to $\cos\varphi_N$(Fig. 14-6). The light intensity, though, depends on the radiant energy per projected area perpendicular to direction $\varphi_N$, which is $dA\cos\varphi_N$. Thus, for Lambertian reflection, the intensity of light is the same over all viewing directions.

**Gouraud Shading**
Gouraud shading computes intensity for each vertex and then interpolates the computed intensities across the polygons. Gouraud shading performs a bi-linear interpolation of the intensities down and then across scan lines. It thus eliminates the sharp changes at polygon boundaries.

**The algorithm is as follows:**

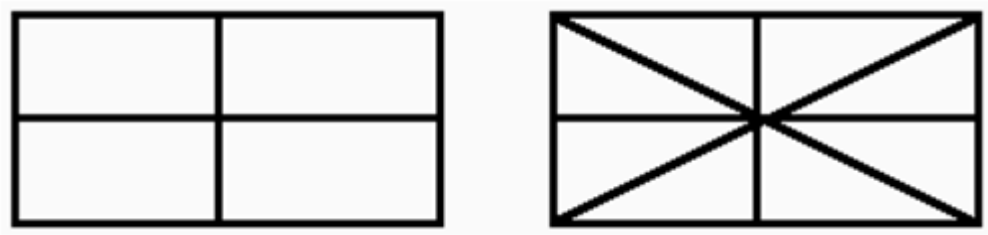
1. Compute a normal N for each vertex of the polygon.

2. From N compute intensity I for each vertex of the polygon.

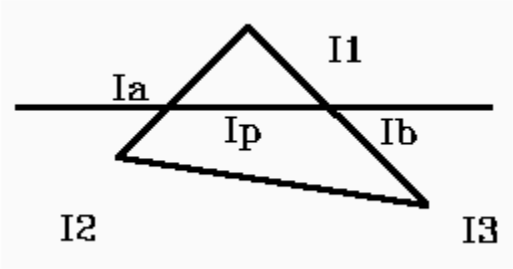
3. From bi-linear interpolation compute intensity Ii for each pixel.

4. Paint pixel to shade corresponding to Ii.

Let N = the average of the normal of the polygons which include the vertex. Note that 4 sided polygons have 4 neighbors and triangles have 6 neighbors.



We can find the neighbors for a particular vertex by searching the polygons and including any polygons which include the vertex. Now look at performing the bi-linear intensity interpolation. This is the same as the bi-linear interpolation for the z depth of a pixel (used with the z buffer visible surface algorithm).




**Advantages:**

Gouraud shading gives a much better image than faceted shading (the facets no longer visible). It is not too computationally expensive: one floating point addition (for each color) for each pixel. (the mapping to actual display registers requires a floating point multiplication)

**Disadvantages:**

It eliminates creases that you may want to preserve, e.g. in a cube. We can modify data structure to prevent this by storing each physical vertex 3 times, i.e. a different logical vertex for each polygon.

**Gouraud shading model**

A shading model is a method of applying a local illumination model to an object, usually an object modelled as a polygon mesh. There are four shading models we will consider: **Constant, Faceted, Gouraud, and Phong**. They give increasingly good images and are increasingly computationally expensive.

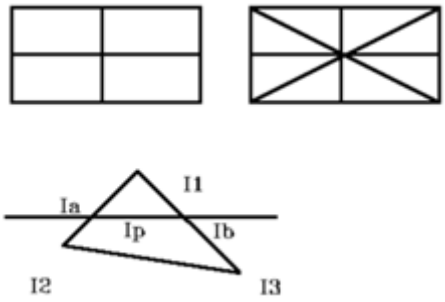
**Gouraud shading model.**: The second shading model, Gouraud shading, computes intensity for each vertex and then interpolates the computed intensities across the polygons. Gouraud shading performs a bi-linear interpolation of the intensities down and then across scan lines. It thus eliminates the sharp changes at polygon boundaries.The algorithm is as follows:

1. Compute a normal N for each vertex of the polygon.

2. From N compute intensity I for each vertex of the polygon.

3. From bi-linear interpolation compute intensity Ii for each pixel.

4. Paint pixel to shade corresponding to Ii.




How do we compute N for a vertex? Let N = the average of the normal of the polygons which include the vertex. Note that 4 sided polygons have 4 neighbors and triangles have 6 neighbors.

We can find the neighbors for a particular vertex by searching the polygons and including any polygons which include the vertex. Now look at performing the bi-linear intensity interpolation. This is the same as the bi-linear interpolation for the z depth of a pixel (used with the z buffer visible surface algorithm).

**Advantages of Gouraud shading:**

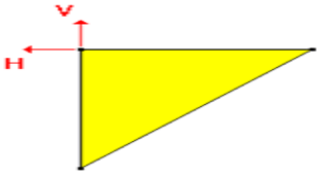
Gouraud shading gives a much better image than faceted shading (the facets no longer visible). It is not too computationally expensive: one floating point addition (for each color) for each pixel. (the mapping to actual display registers requires a floating point multiplication)

**Disadvantages to Gouraud shading:**

It eliminates creases that you may want to preserve, e.g. in a cube. We can modify data structure to prevent this by storing each physical vertex 3 times, i.e. a different logical vertex for each polygon. here is a data structure for a cube that will keep the edges:

**Phong shading method**

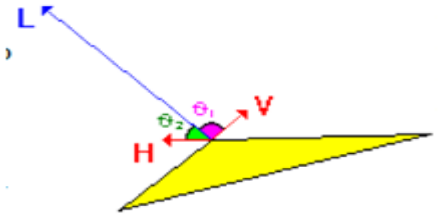
The basic Phong shading technique tends to be very slow. However, it can look really good, and so it has been the subject of much optimization. It's only possible to optimize the original algorithm so far. No matter how much Fixed Point math's and lookup tables you throw at it, it still runs too slowly to make it a realistic option for realtime graphics.



A document has been circulating for some time under the name of OTMPHONG.DOC. It suggests that instead of interpolating the normal vector across the polygon, you should interpolate the angle between the light source and the normal. This is a nice idea except for the fact that it doesn't really work. It is essentially a repeat of Gouraud shading, and of little use to anyone.

So, for each vertex of the polygon to be rendered, you will need to calculate the coordinates to the Phong Map

You have a polygon. For this polygon, define two vectors which are at right angles to each other and to the surface normal. Call them V and H. These two vectors represent the (u,v) coordinates of the phong map. Now we'll look at this polygon at an angle so the vector from the light source to the vertex can be seen. This is vector L. The aim is to calculate the coordinatesThe aim is to calculate the coordinates

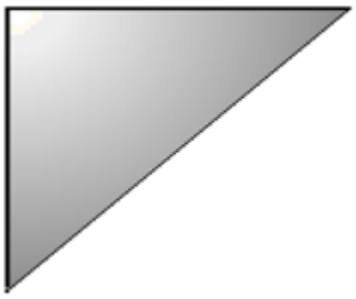



The algorithm is a very simple one. If the phong map is 256x256 and centered then:

**u = ( V . L ) * 128 127**

**v = ( H . L ) * 128 127**

Do this for each vertex, and then map the Phong map onto it, and there you have one nicely phong shaded polygon.



Since this can be slow, there are various ways you can speed it up if you don't mind a little loss of freedom. If you assume that the light source is at the same place as the camera, then you can ignore the V and H vectors altogether. Instead take the X and Y components of the normal vector, multiply by 128 and add 127 (assuming that is that the magnitude of the normal vector is 1). Alternatively, you can take the light source as being like an inverse camera. Transform the object as if the light source were the camera, and calculate the phong as in the previous paragraph.

## 3D Transformation Matrices For Translation, Scaling & Rotation

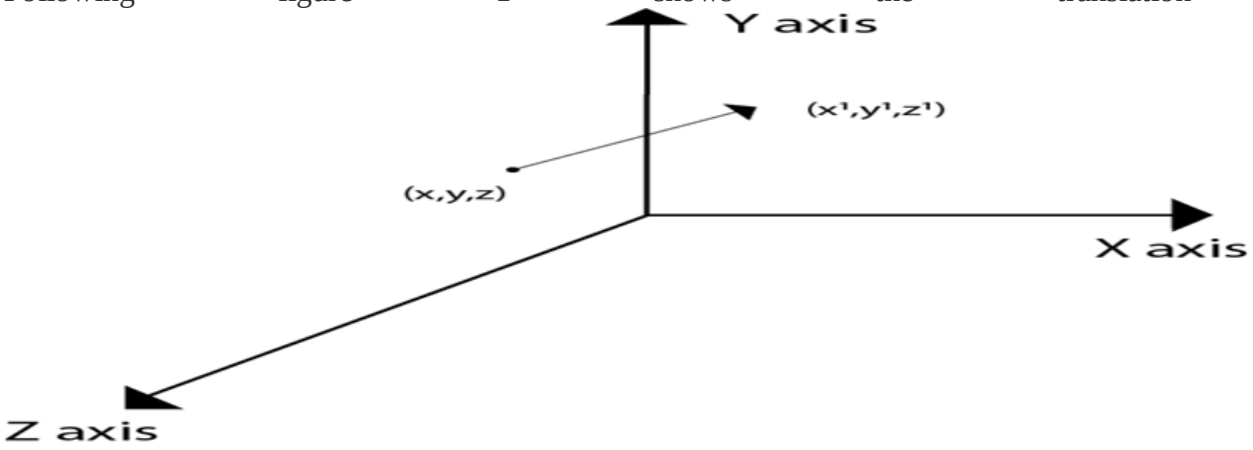
### Translation

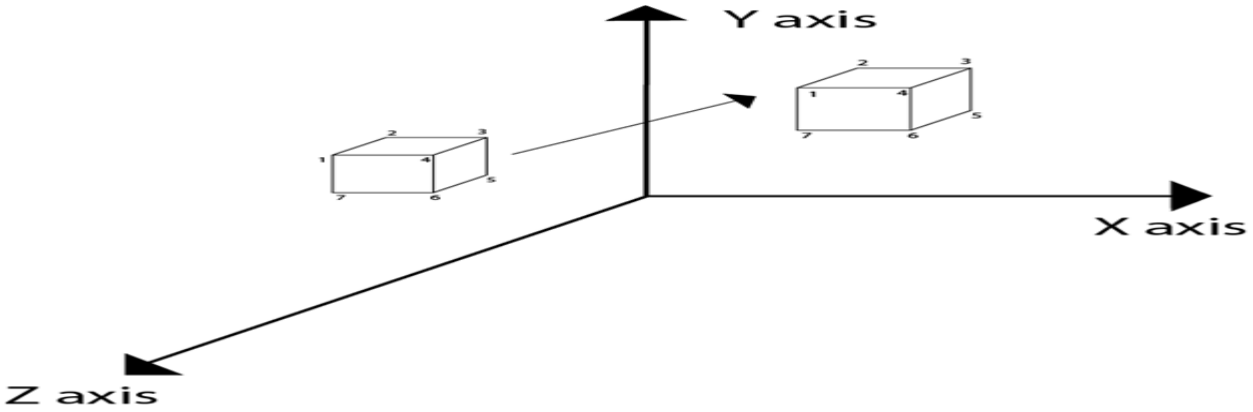
It is the movement of an object from one position to another position. Translation is done using translation vectors. There are three vectors in 3D instead of two. These vectors are in x, y, and z directions. Translation in the x-direction is represented using $T_x$. The translation is y-direction is represented using $T_y$. The translation in the z- direction is represented using $T_z$.

If P is a point having co-ordinates in three directions (x, y, z) is translated, then after translation its coordinates will be ($x^1$ $y^1$ $z^1$) after translation. $T_x$ $T_y$ $T_z$ are translation vectors in x, y, and z directions respectively.

$$x^1=x+T_x$$
$$y^1=y+T_y$$
$$z^1=z+T_z$$

Three-dimensional transformations are performed by transforming each vertex of the object. If an object has five corners, then the translation will be accomplished by translating all five points to new locations. Following figure 1 shows the translation of point



## Matrix for translation

$$\begin{Bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{Bmatrix} \text{ or } \begin{Bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$$

## Matrix representation of point translation

Point shown in fig is (x, y, z). It become ($x^1$,$y^1$,$z^1$) after translation. $T_x$ $T_y$ $T_z$ are translation vector.

$$\begin{bmatrix} x^1 \\ y^1 \\ z^1 \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

**Example:** A point has coordinates in the x, y, z direction i.e., (5, 6, 7). The translation is done in the x-direction by 3 coordinate and y direction. Three coordinates and in the z- direction by two coordinates. Shift the object. Find coordinates of the new position.

**Solution:** Co-ordinate of the point are (5, 6, 7)
Translation vector in x direction = 3
Translation vector in y direction = 3
Translation vector in z direction = 2
Translation matrix is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{pmatrix}$$

Multiply co-ordinates of point with translation matrix

$$(x^1y^1z^1) = (5,6,7,1)\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 3 & 2 & 1 \end{pmatrix}$$

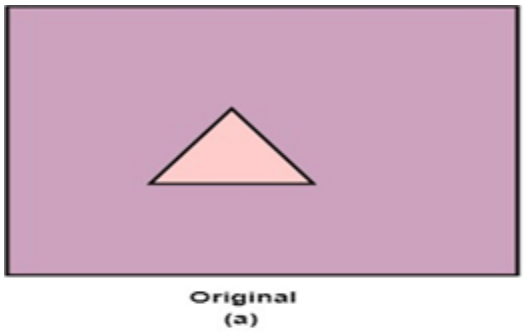
= [5+0+0+30+6+0+30+0+7+20+0+0+1] = [8991]

x becomes $x^1=8$
y becomes $y^1=9$
z becomes $z^1=9$

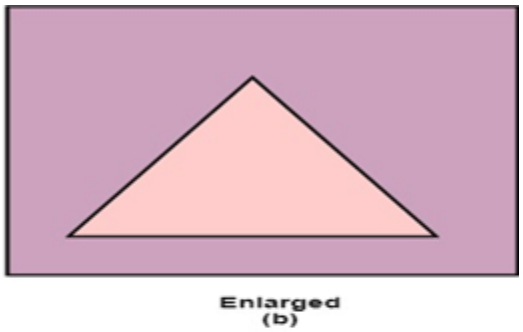
## Scaling

Scaling is used to change the size of an object. The size can be increased or decreased. The scaling three factors are required $S_x$ $S_y$ and $S_z$.

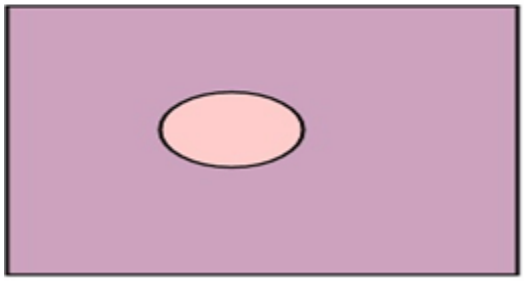
$S_x$=Scaling factor in x- direction
$S_y$=Scaling factor in y-direction
$S_z$=Scaling factor in z-direction



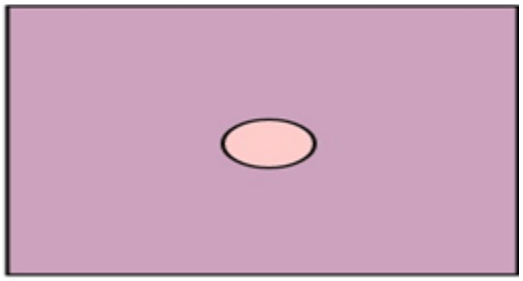
Original
(a)



Enlarged
(b)



Original
(a)



Reduced
(b)

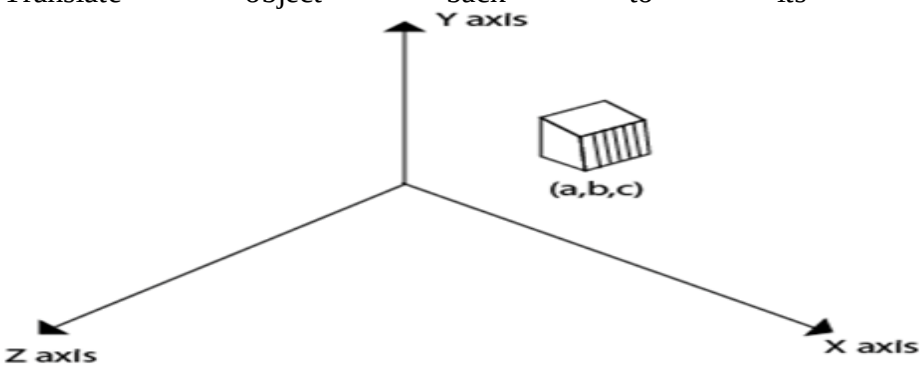
## Matrix for Scaling

$$\begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

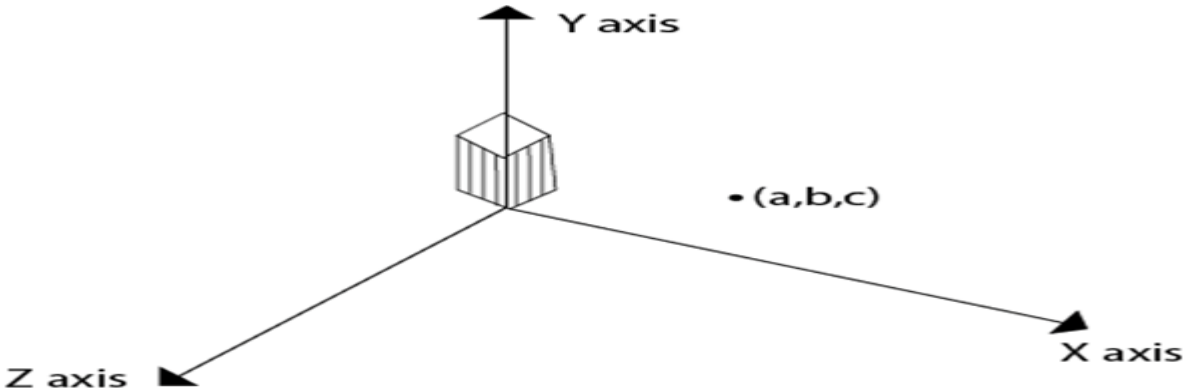
## Scaling of the object relative to a fixed point

Following are steps performed when scaling of objects with fixed point (a, b, c). It can be represented as below:

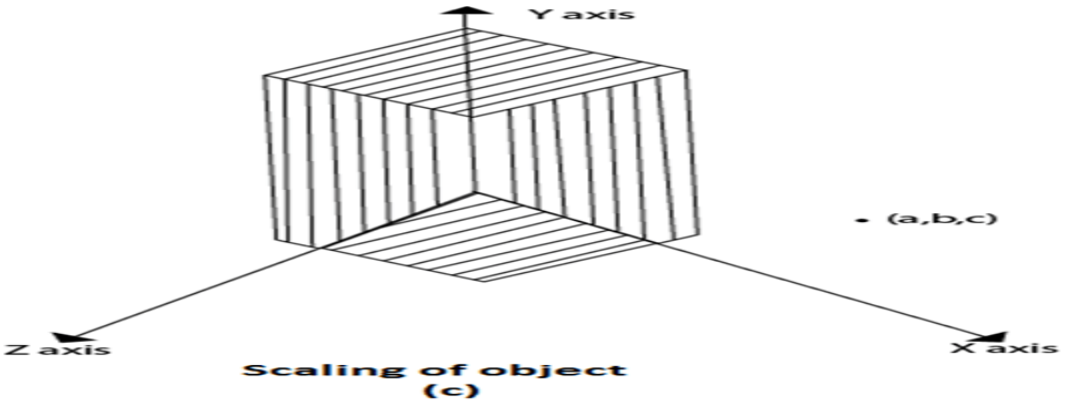
1. Translate fixed point to the origin
2. Scale the object relative to the origin
3. Translate object back to its original positioN



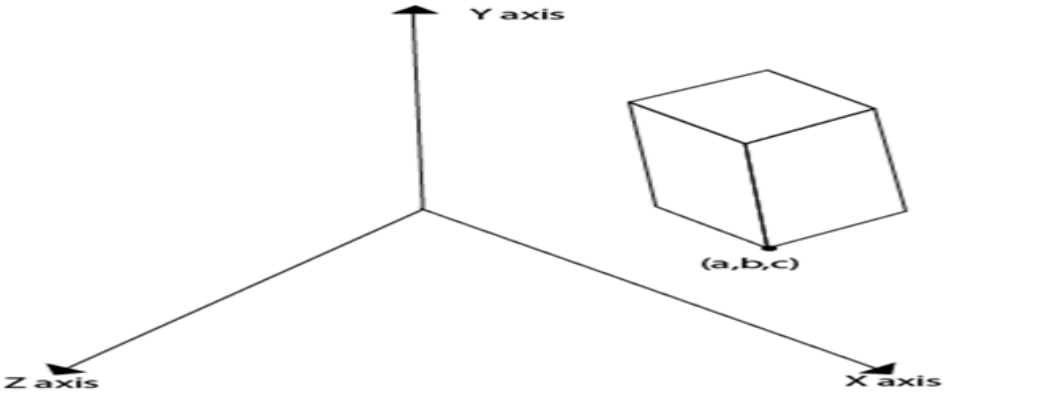

**Object & point for scaling**
**(a)**




**Object transfer at origin**
**(b)**




Scaling of object
(c)

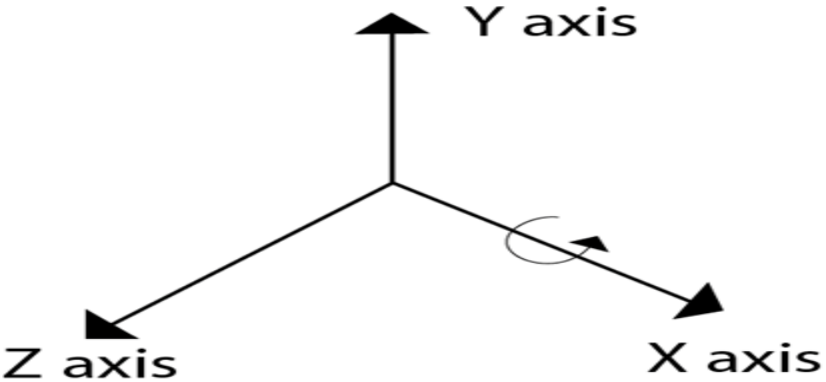



Object shifted to original position after scaling
(d)

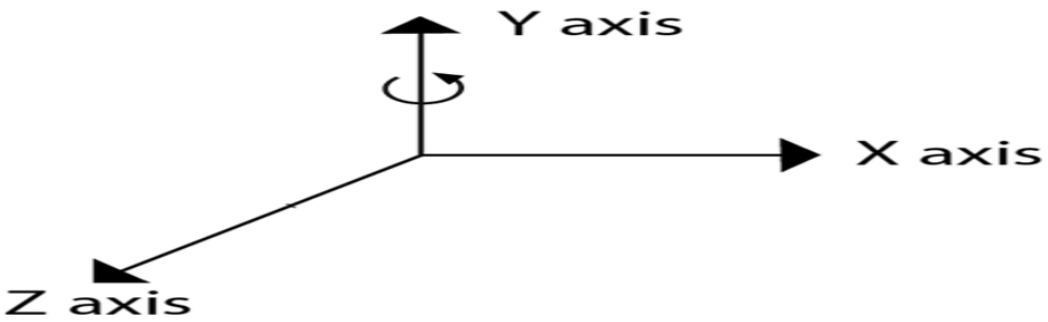
## Rotation

It is moving of an object about an angle. Movement can be anticlockwise or clockwise. 3D rotation is complex as compared to the 2D rotation. For 2D we describe the angle of rotation, but for a 3D angle of rotation and axis of rotation are required. The axis can be either x or y or z.

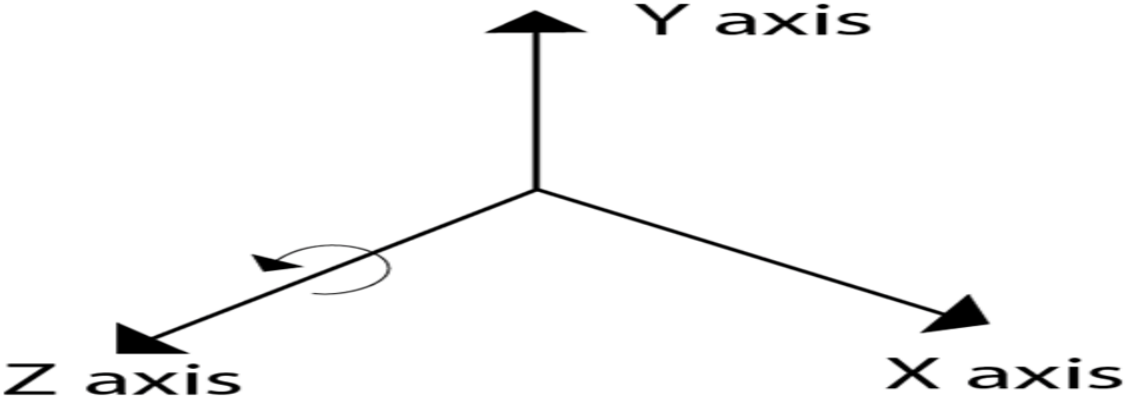
**Following figures shows rotation about x, y, z- axis**



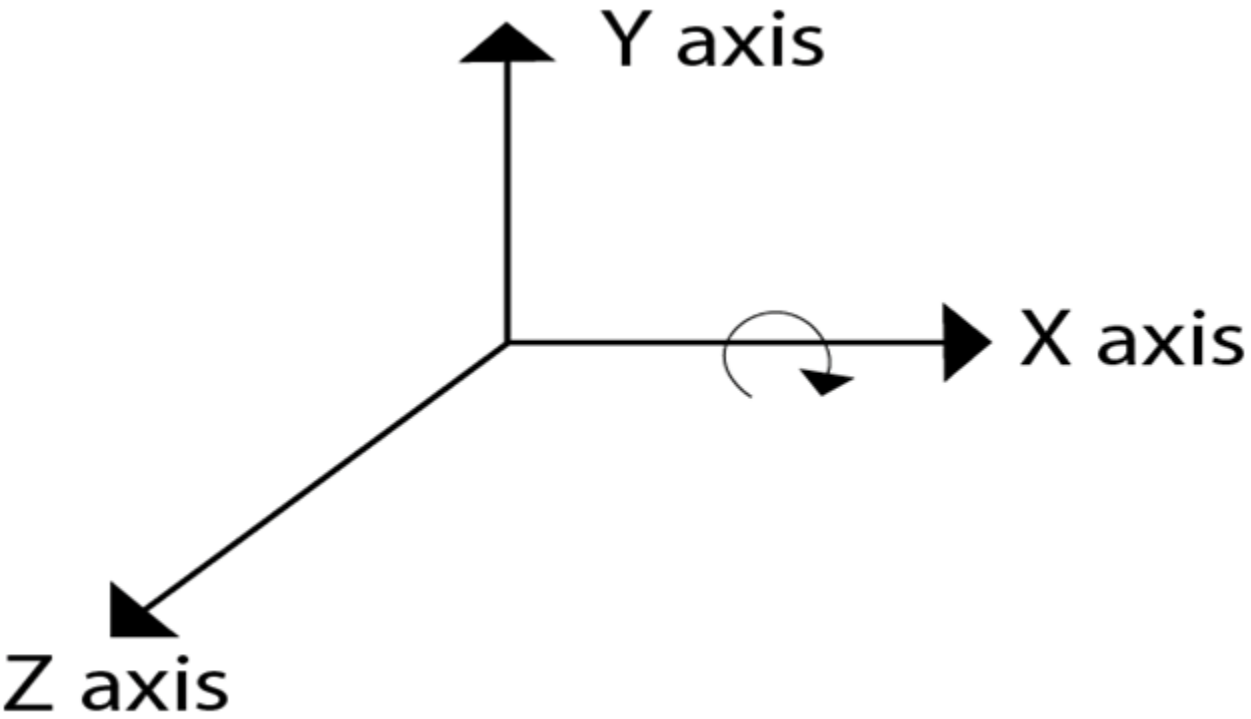

rotation object X axis anticlockwise




rotation object Y axis anticlockwise

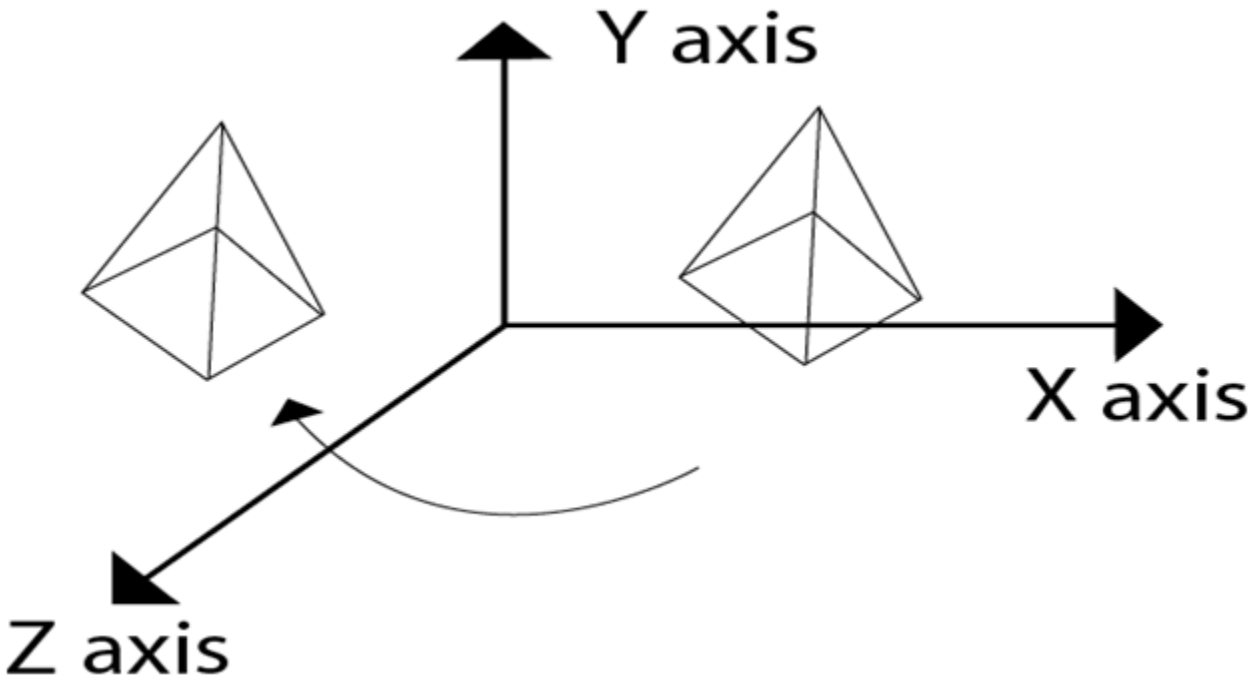



rotation object Z axis anticlockwise

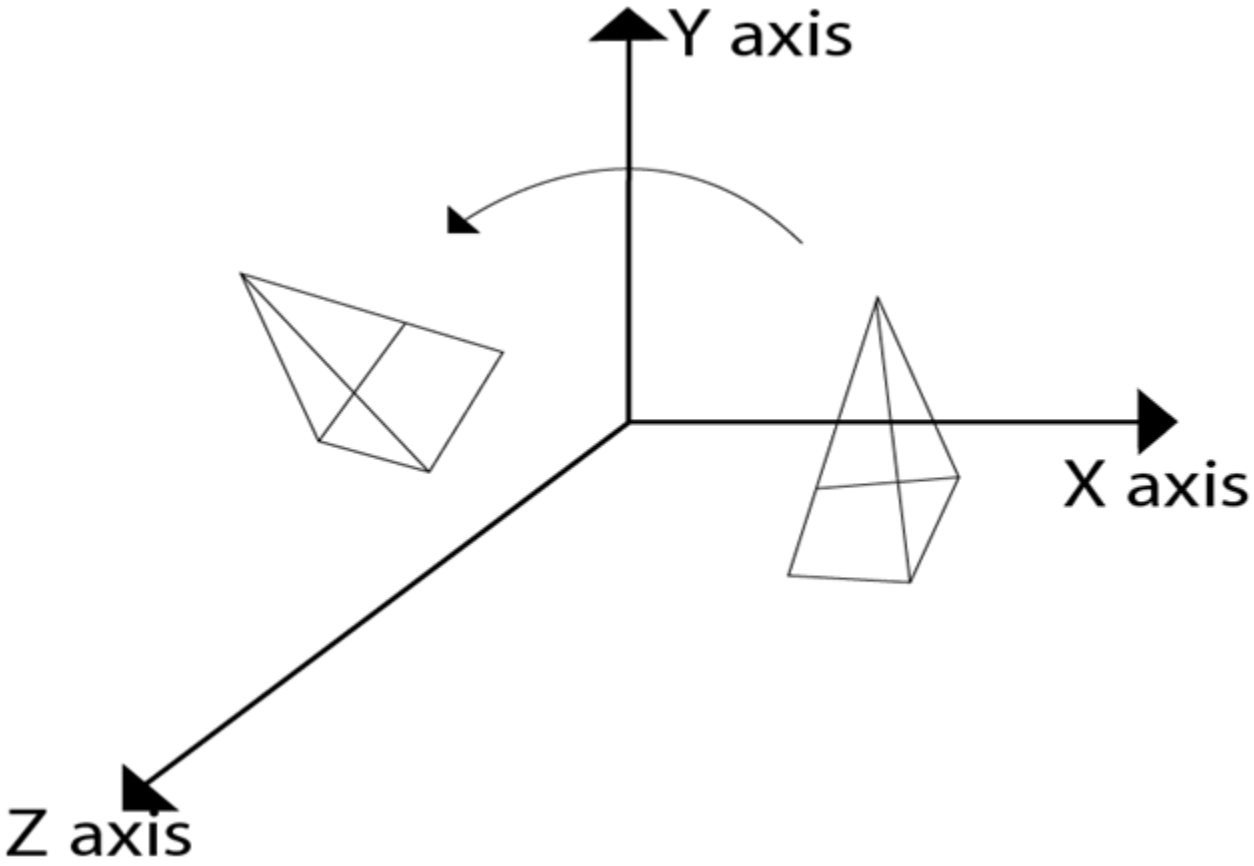



rotation about X axis clockwise

Following figure show rotation of the object about the Y axis



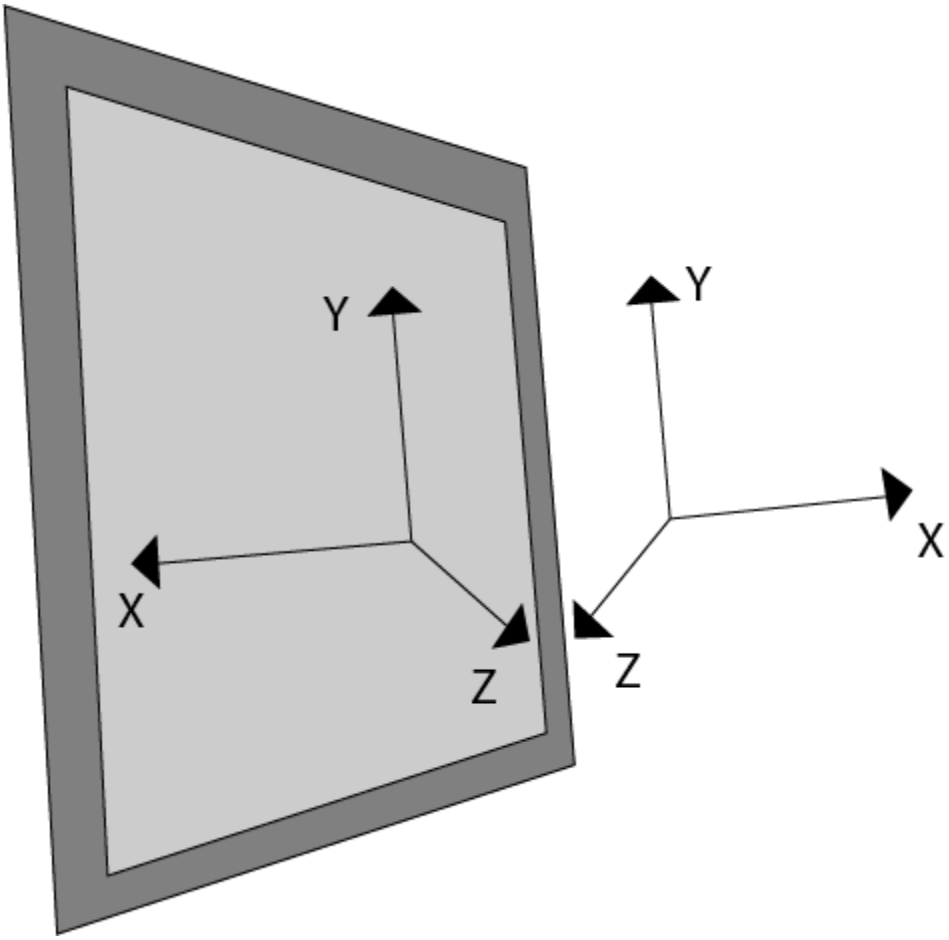

Following figure show rotation of the object about the Z axis



## Reflection

It is also called a mirror image of an object. For this reflection axis and reflection of plane is selected. Three-dimensional reflections are similar to two dimensions. Reflection is 180° about the given axis. For reflection, plane is selected (xy,xz or yz). Following matrices show reflection respect to all these three planes.

**Reflection relative to XY plane**




$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Reflection relative to YZ plane**

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

**Reflection relative to ZX plane**

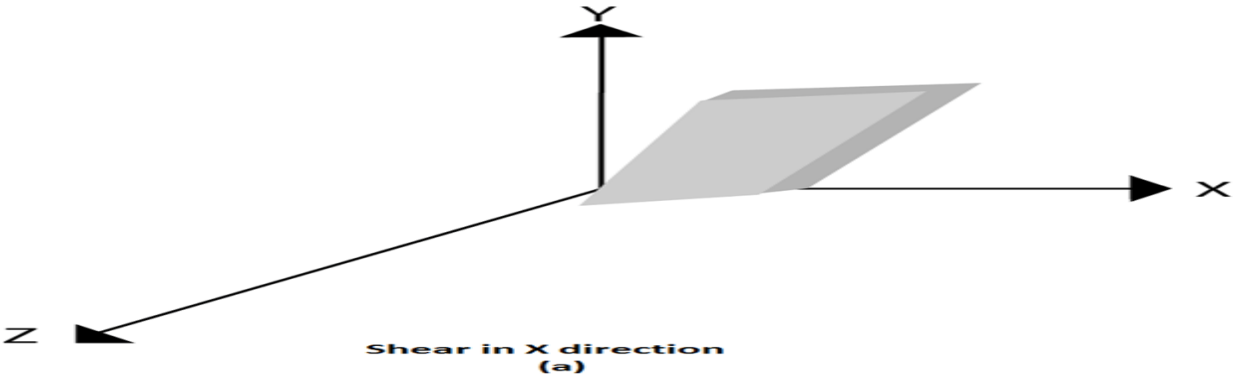
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## Shearing

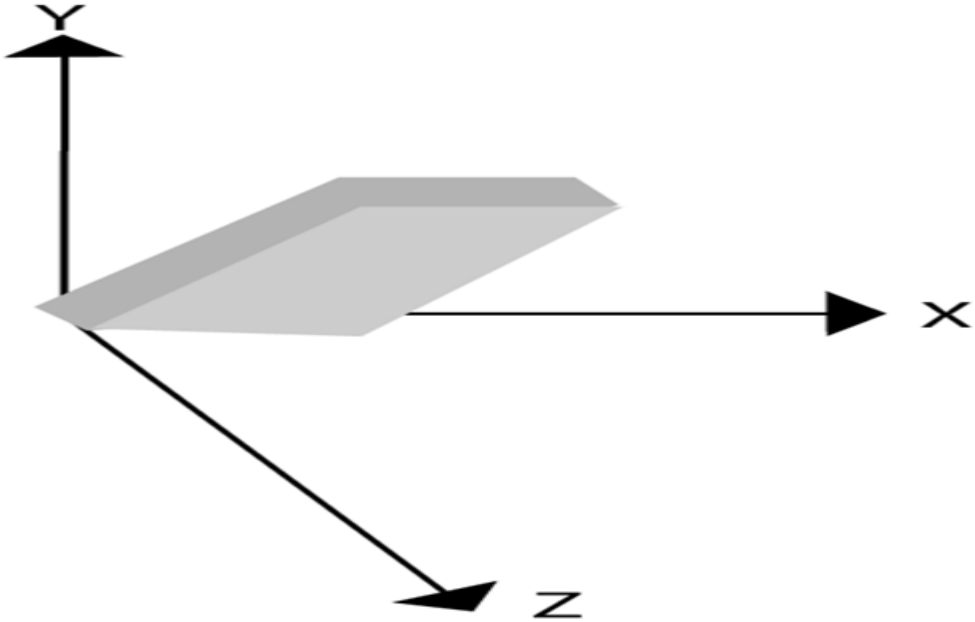
It is change in the shape of the object. It is also called as deformation. Change can be in the x -direction or y -direction or both directions in case of 2D. If shear occurs in both directions, the object will be distorted. But in 3D shear can occur in three directions.

**Matrix for shear**

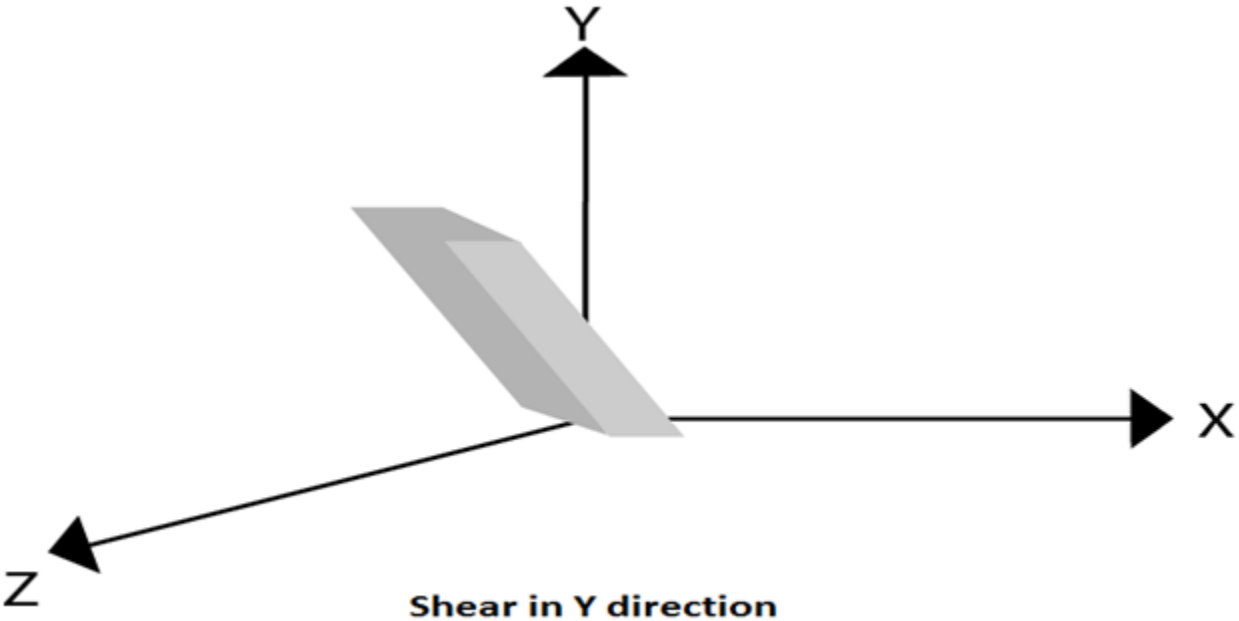
$$\begin{pmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$




Shear in X direction
(a)




Shear in X, Y direction
(b)




Shear in Y direction
(c)

**3d viewing pipeline**

In two-dimensional graphics applications, viewing operations transfer positions from the world-coordinate plane to pixel positions in the plane of the output device. Using the rectangular boundaries for the world-coordinate window and the device viewport, a two-dimensional package maps the world scene to device coordinates and clips the scene against the four boundaries of the viewport. For three-dimensional graphics applications, the situation is a bit more involved, since we now have more choices as to how views are to be generated. First of all, we can view an object from any spatial position: from the front, from above, or from the back.

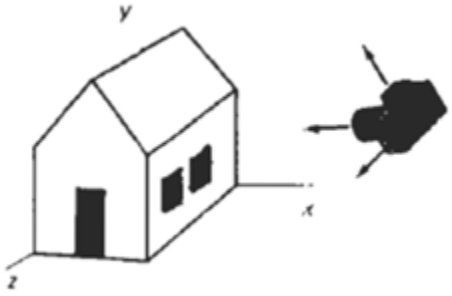



*Figure 12-1*
Photographing a scene involves selection of a camera position and orientation.

The steps for computer generation of a view of a three-dimensional scene are somewhat analogous to the processes involved in taking a photograph. To take a snapshot, we first need to position the camera at a particular point in space. Then we need to decide on the camera orientation (Fig. 12-1): Which way do we point the camera and how should we rotate it around the line of sight to set the up direction for the picture? Finally, when we snap the shutter, the scene is cropped to the size of the "window" (aperture) of the camera, and light from the visible surfaces is projected onto the camera film. We need to keep in mind, however, that the camera analogy can be carried only so far, since we have more flexibility and many more options for generating views of a scene with a graphics package than we do with a camera.

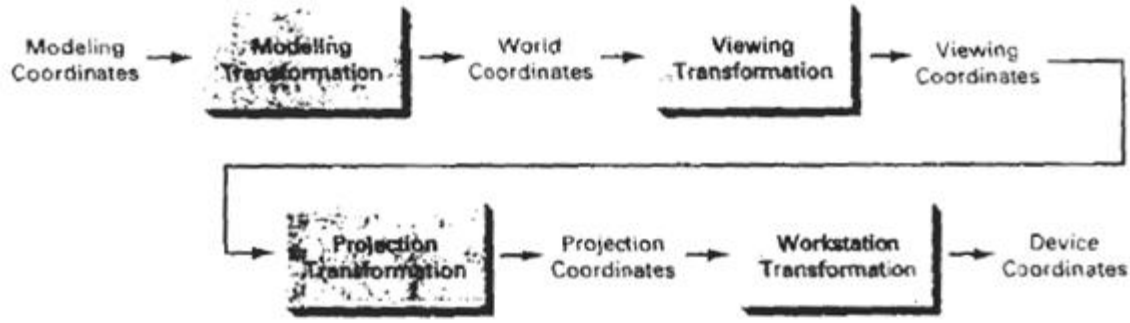



*Figure 12-2*
General three-dimensional transformation pipeline, from modeling coordinates to final device coordinates.

Figure 12-2 shows the general processing steps for modeling and converting a world-coordinate description of a scene to device coordinates. Once the scene has been modeled, world-coordinate positions are converted to viewing coordinates. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of in analogy with the camera film plane. Next, projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device. Objects outside the specified viewing limits are clipped h m further consideration, and the remaining objects are processed through visible-surface identification and surface-rendering procedures to produce the display within the device viewport.

**Viewing coordinate**

Generating a view of an object in three dimensions is similar to photographing the object. We can walk around and take its picture from any angle, at variousdistances, and with varying camera orientations. Whatever appears in the viewfinder is projected onto the flat film surface. The type and size of the camera lens determines which parts of the scene appear in the final picture. These ideas are incorporated

into threedimensional graphics packages so that views of a scene can be generated, given the spatial position, orientation, and aperture size of the "camera".

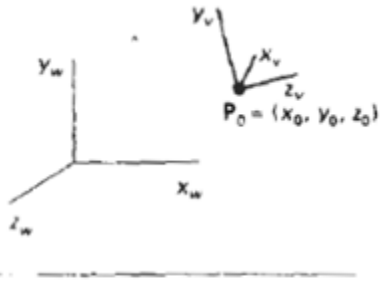



*Figure 12-3*
A right-handed viewing-coordinate system, with axes $x_v$, $y_v$, and $z_v$ relative to a world-coordinate scene.

**Specifying the View Plane:** We choose a particular view for a scene by first establishing the viewing-coordinate system, also called the view reference coordinate system, as shown in Fig. 12-3. A view plane, or projection plane, is then set up perpendicular to the viewing z, axis. We can think of the view plane as the film plane in a camera that has been positioned and oriented for a particular shot of the scene. World-coordinate positions in the scene are transformed to viewing coordinates, and then viewing coordinates are projected onto the view plane.

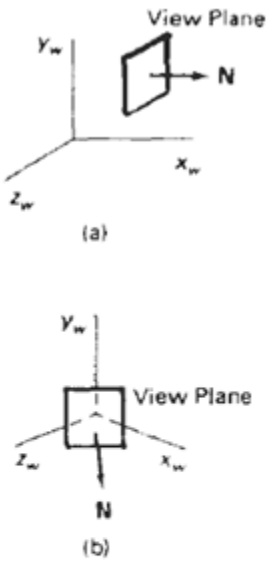



*Figure 12-4*
Orientations of the view plane for specified normal vector coordinates relative to the world origin. Position (1, 0, 0) orients the view plane as in (a), while (1, 0, 1) gives the orientation in (b).

To establish the viewing-coordinate reference frame, we first pick a world coordinate position called the view reference point. This point is the origin of our viewing-coordinate system. The view reference point is often chosen to be close to or on the surface of some object in a scene. But we could also choose a point that is at the cent4.r of an object, or at the center of a group of objects, or somewhere out in front of the scene to be displayed. If we choose a point that is near to or on some object, we can think of this point as the position where we might want to aim a camera to take a picture of the object. Alternatively, if we choose a point that is at some distance from a scene, we could think of this as the camera position.

Next, we select the positive direction for the viewing z, axis, and the orientation of the view plane, by specifying the view-plane normal vector, N. We choose a world-coordinate position, and this point establishes the direction for N relative either to the world origin or to the viewing-coordinate origin. Graphics packages such as GKS and PHIGS, for example, orient N relative to the world coordinate origin, as shown in Fig. 12-4. The view-plane normal N is then the directed line segment from the world origin to the selected coordinate position. In other words, N is simply specified as a world-coordinate vector. Some other packages (GL from silicongraphic for, instance) establish the direction for N using the selected coordinate position as a look-at poi111 relative to the view reference point (,viewing-coordinate origin). Figure 12-5 illustrates this method for defining the direction of N, which is from the look-at point to the view reference point. Another possibility is to set up a left-handed viewing system and take N and the positive xi, axis from the viewing origin to the look-at point. Only the direction of N is needed to establish the z, direction; the magnitude is irrelevant, because N will be normalized to a unit vector by the viewing calculations.

**view volume and general projection**

In the camera analogy, the type of lens used on the camera is one factor that determines how much of the scene 1s caught on film. A wide angle lens takes in more of the scene than a regular lens. In three-dimensional viewing, a rectangular view window, or projection window, in the view plane is used to the same effect. Edges of the view window are parallel to the x,y,, axes, and the window boundary positions are specified in viewing coordinates. The view window can be placed anywhere on the view plane.

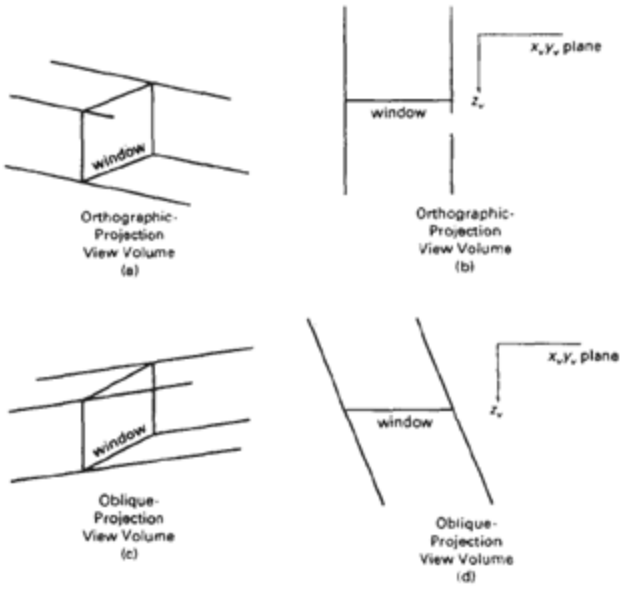



*Figure 12-28*
View volume for a parallel projection. In (a) and (b), the side and top views of the view volume for an orthographic projection are shown; and in (c) and (d), the side and top views of an oblique view volume are shown.

Given the specification of the view window, we can set up a view volume using the window boundaries. Only those objects within the view volume will appear in the generated display on an output device all others are clipped from the display. The size of the view volume depends on the size of the window, while the shape of the view volume depends on the type of projection to be used to generate the display. In any case, four sides of the volume are planes that pass through the edges of the window. For a parallel projection, these four sides of the view volume form an infinite parallelepiped, as in Fig. 12-28. For a perspective projection, the view volume is-a pyramid with apex at the projection reference point (Rg. 12-29).

A finite view volume is obtained by limiting the extent of the volume in the 2 direction. This is done by specifying positions for one or two additional boundary planes. These z,-boundary planes are referred to as the front plane and back plane, or the near plane and the far plane, of the viewing volume. The front and back planes are parallel to the view plane at specified-positions $z_{front}$ and $z_{back}$. Both planes must be on the same side of the projection reference point, and the back plane must be farther from the projection point than the front plane. Including the front and back planes produces a view volume bounded by six planes, as shown in Fig. 12-30 with an orthographic parallel projection; the six planes form a rectangular parallelepiped, while an oblique parallel projection produces an oblique parallelepiped view volume. With a perspective projection, thefront and back clipping planes truncate the infinite pyramidal view volume to form a frustum.

## Animation

Animation refers to the movement on the screen of the display device created by displaying a sequence of still images. Animation is the technique of designing, drawing, making layouts and preparation of photographic series which are integrated into the multimedia and gaming products. Animation connects the exploitation and management of still images to generate the illusion of movement. A person who creates animations is called animator. He/she use various computer technologies to capture the pictures and then to animate these in the desired sequence.

Animation includes all the visual changes on the screen of display devices. These are:

1. Change of shape as shown in fig:

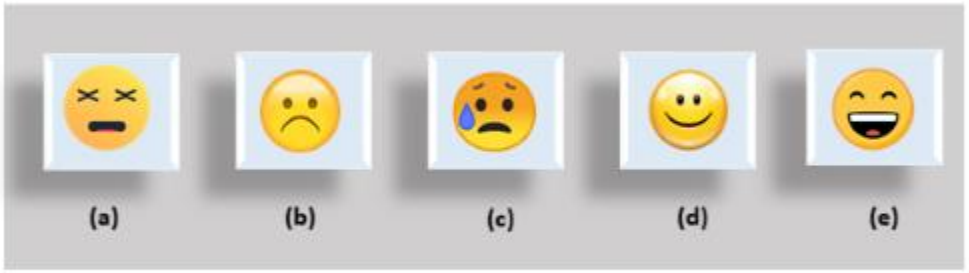



**Fig: Change in Shape**

2. Change in size as shown in fig:

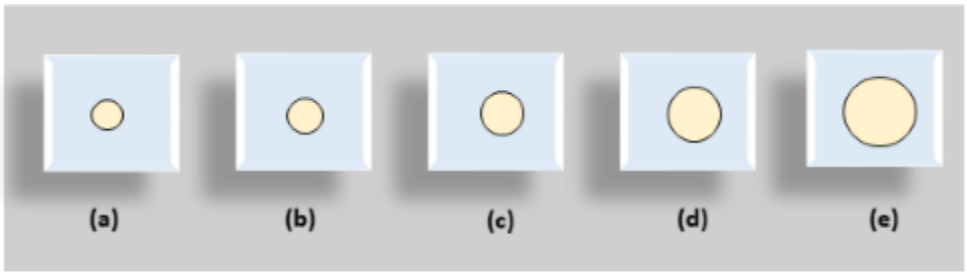



Fig: Change in Size

3. Change in color as shown in fig:

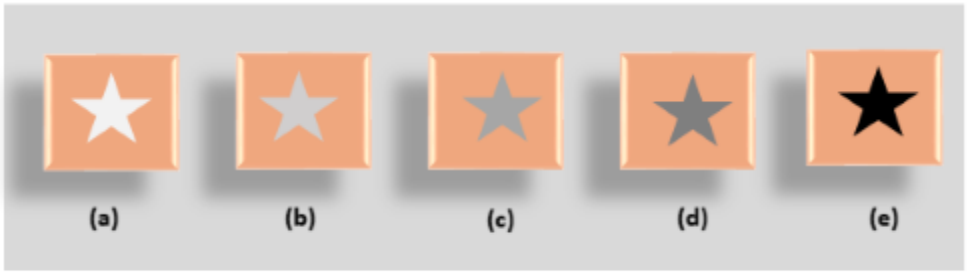



Fig: Change in Color

4. Change in structure as shown in fig:

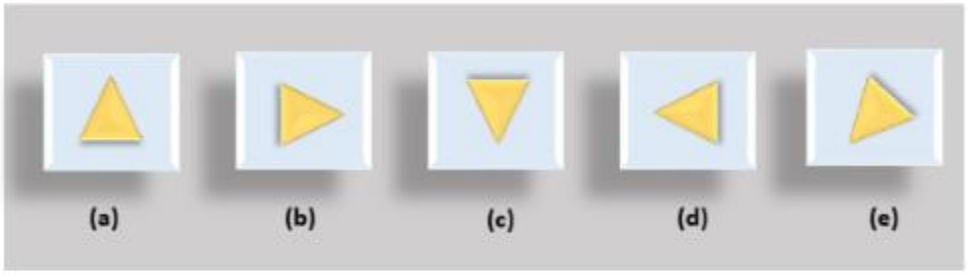



Fig: Change in Structure

5. Change in angle as shown in fig:

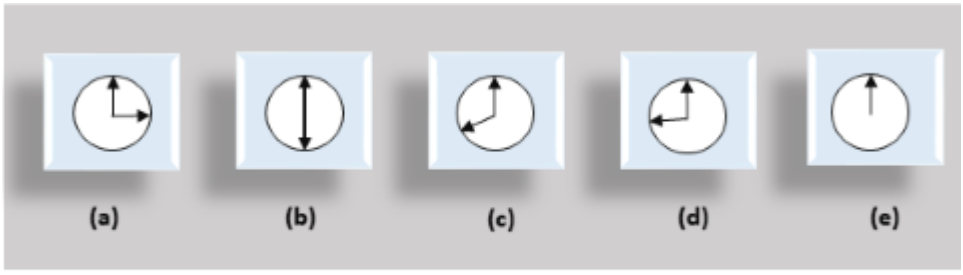



Fig: Change in angle

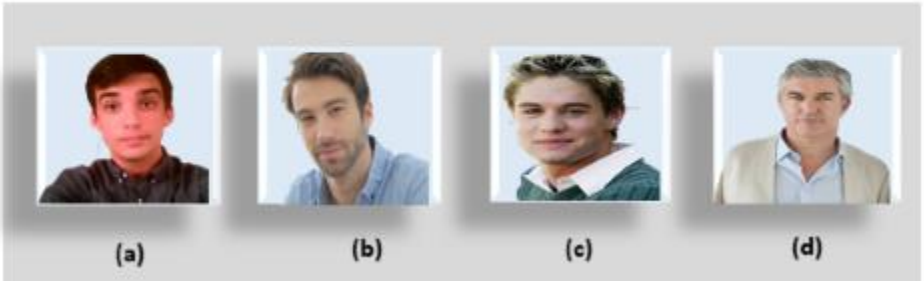
## Animation Functions

**1. Morphing:** Morphing is an animation function which is used to transform object shape from one form to another is called Morphing. It is one of the most complicated transformations. This function is commonly used in movies, cartoons, advertisement, and computer games.

**For Example:**

**1.Human Face is converted into animal face as shown in fig:**


**2. Face of Young person is converted into aged person as shown in fig:**




**The process of Morphing involves three steps:**

1. In the first step, one initial image and other final image are added to morphing application as shown in fig: I$^{st}$ & 4$^{th}$ object consider as key frames.
2. The second step involves the selection of key points on both the images for a smooth transition between two images as shown in 2$^{nd}$ object.

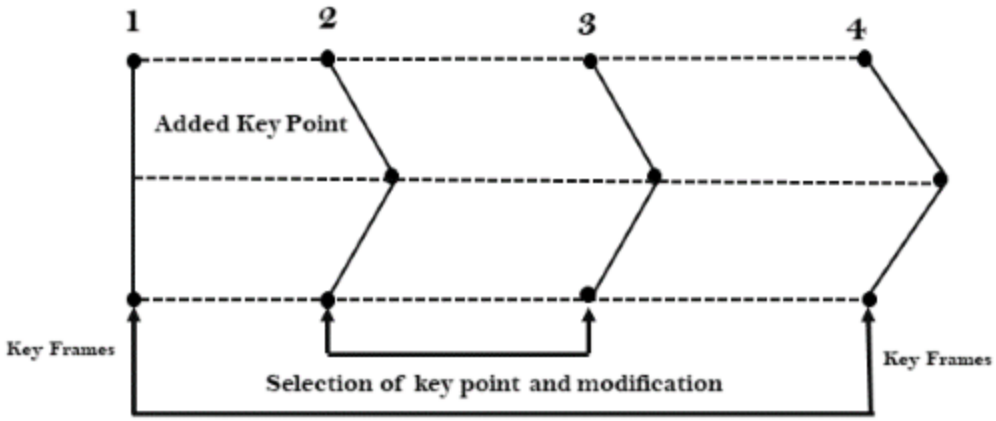



Fig: Process of Morphing

3. In the third step, the key point of the first image transforms to a corresponding key point of the second image as shown in 3rd object of the figure.

2. **Wrapping:** Wrapping function is similar to morphing function. It distorts only the initial images so that it matches with final images and no fade occurs in this function.

3. **Tweening:** Tweening is the short form of 'inbetweening.' Tweening is the process of generating intermediate frames between the initial & last final images. This function is popular in the film industry.

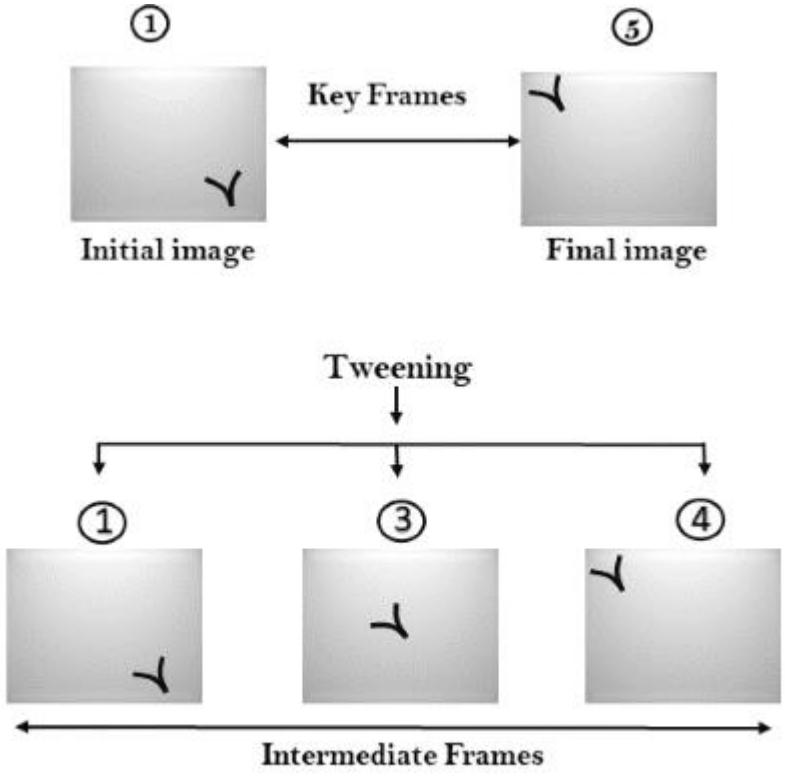



Fig: Tweening

4. **Panning:** Usually Panning refers to rotation of the camera in horizontal Plane. In computer graphics, Panning relates to the movement of fixed size window across the window object in a scene. In which direction the fixed sized window moves, the object appears to move in the opposite direction as shown in fig:

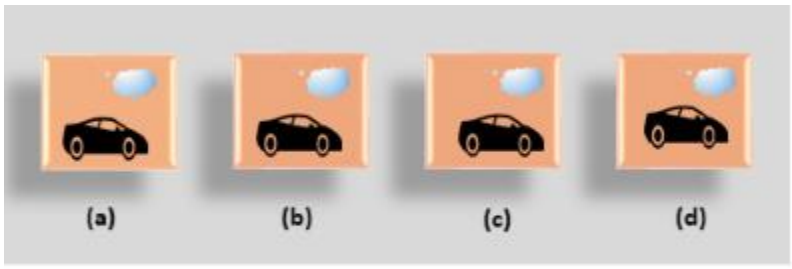



**Fig: Panning**

If the window moves in a backward direction, then the object appear to move in the forward direction and the window moves in forward direction then the object appear to move in a backward direction.

5. **Zooming:** In zooming, the window is fixed an object and change its size, the object also appear to change in size. When the window is made smaller about a fixed center, the object comes inside the window appear more enlarged. This feature is known as **Zooming In**.

When we increase the size of the window about the fixed center, the object comes inside the window appear small. This feature is known as **Zooming Out**.

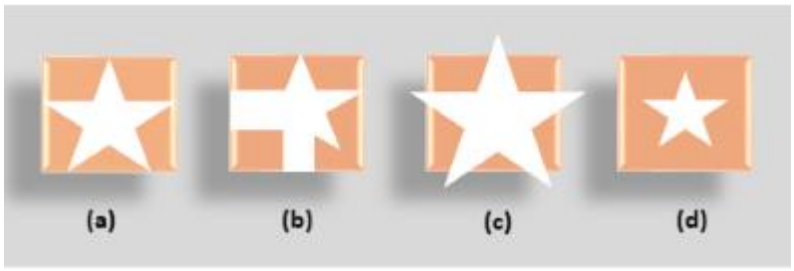



**Fig: Zooming in & Zooming Out**

6. **Fractals:** Fractal Function is used to generate a complex picture by using Iteration. Iteration means the repetition of a single formula again & again with slightly different value based on the previous iteration result. These results are displayed on the screen in the form of the display picture.

### Raster Methods for Computer Animation

- We can create simple animation sequences in our programs using real-time methods.
- We can produce an animation sequence on a raster-scan system one frame at a time, so that each completed frame could be saved in a file for later viewing.
- The animation can then be viewed by cycling through the completed frame sequence, or the frames could be transferred to film.
- If we want to generate an animation in real time, however, we need to produce the motion frames quickly enough so that a continuous motion sequence is displayed.

- Because the screen display is generated from successively modified pixel values in the refresh buffer, we can take advantage of some of the characteristics of the raster screenrefresh process to produce motion sequences quickly.

**Double Buffering**

- One method for producing a real-time animation with a raster system is to employ two refresh buffers.
- We create a frame for the animation in one of the buffers.
- Then, while the screen is being refreshed from that buffer, we construct the next frame in the other buffer.
- When that frame is complete, we switch the roles of the two buffers so that the refresh routines use the second buffer during the process of creating the next frame in the first buffer.
- When a call is made to switch two refresh buffers, the interchange could be performed at various times.
- The most straight forward implementation is to switch the two buffers at the end of the current refresh cycle, during the vertical retrace of the electron beam.
- If a program can complete the construction of a frame within the time of a refresh cycle, say 1/60 of a second, each motion sequence is displayed in synchronization with the screen refresh rate.
- If the time to construct a frame is longer than the refresh time, the current frame is displayed for two or more refresh cycles while the next animation frame is being generated.
- Similarly, if the frame construction time is 1/25 of a second, the animation frame rate is reduced to 20 frames per second because each frame is displayed three times.
- Irregular animation frame rates can occur with double buffering when the frame construction time is very nearly equal to an integer multiple of the screen refresh time the animation frame rate can change abruptly and erratically.
- One way to compensate for this effect is to add a small time delay to the program.
- Another possibility is to alter the motion or scene description to shorten the frame construction time.

**Generating Animations Using Raster Operations**

- We can also generate real-time raster animations for limited applications using block transfers of a rectangular array of pixel values.
- A simple method for translating an object from one location to another in the xy plane is to transfer the group of pixel values that define the shape of the object to the new location
- Sequences of raster operations can be executed to produce realtime animation for either two-dimensional or three-dimensional objects, so long as we restrict the animation to motions in the projection plane.
- Then no viewing or visible-surface algorithms need be invoked.
- We can also animate objects along two-dimensional motion paths using color table transformations.
- Here we predefine the object at successive positions along the motion path and set the successive blocks of pixel values to color-table entries.
- The pixels at the first position of the object are set to a foreground color, and the pixels at the other object positions are set to the background color .
- Then the animation is then accomplished by changing the color-table values so that the object color at successive positions along the animation path becomes the foreground color as the preceding position is set to the background color

**Computer Animation languages:**

There is several animation languages already develop. All of them can be categories under three groups:

**1. linear list notations language:**

It is the specially animation supporting language. Each event in the animation is described by start and ending frame number and an action that is to take place (event). The example of this type of languages is SCEFO (scene format). For example:
42, 53, B, rotate, "palm", 1, 30
Here,
42 => start frame no.
53 => ending frame no.
B => table.
Rotate => action.
Palm => object.
1 => start angle.
30 => end angle.

**2. General purpose languages:**

The high level computer languages which are developed for the normal application software development also have the animation supporting features along with graphics drawing, For example QBASIC, C, C++, java etc.

**3. Graphical language:**

it is also computer high level language and especially develop for graphics drawing and animation has been already develop for e.g. AutoCAD.

**Display of animation:**

To display the animation on video motivator as a series of horizontal scan lines from top to bottom in pixels (raster system). The animated object must be scan, converted into their pixmap image in the frame buffer. The conversion must be done at least ten times per second (preferably 15 to 20 times), for smooth animations effects. Hence a new image must be created in no more than 100 milliseconds, from these 100 milliseconds scan conversion should take only a small portion of time, so that redraw and erase of the object on the display can be done fast enough.

**Transmission of animation:**

The transmission of animation over computer network may be using any one of two methods:

***1. By using symbolic representation:***

The symbolization of animation object (for example ball) is transmitted along with the operation commands performed on the thing and at the receiver side the animation is displayed after scan converting into pixmap. during this case the UTC is brief because the symbolization of animated object is smaller in size then pixmap but display time at receiver takes longer thanks to scan converting operation

has got to be acting at receiver side.

***2. By pixmap representation:***

In this method the animated object is converted into pixmap and then transmitted to the receiver. In this case, the transmission time is longer in comparisons to symbolic representation, because the size of the pixmap is larger than symbolic but display time is shorter because no scan conversion should have to perform at receiver side.

**Key frame systems**

A keyframe is a frame where we define changes in animation. They create keyframes. Keyframes are important frames during which an object changes its size, direction, shape or other properties. The computer then figures out all the in-between frames and saves an extreme amount of time for the animator.