UNIT -1 INTRODUCTION TO COMPUTER GRAPHICS

UNIT STRUCTURE

1.0 OBJECTIVE

The objectives of this lesson are to make the student aware of the following concepts

• Describe computer graphics, its features and characteristics;

• Discuss applications of computer graphics in various fields

• Describe various types of hardware, required to work with graphic systems like

a) Display systems

b) Cathode ray tube

c) Random Scan

d) Raster Scan and

e) Display processor

f) Graphics Pipelining

- Discuss existing Graphic Softwares to assist in areas of Graphical processing

1.1 DEFINITION OF COMPUTER GRAPHICS

Computer Graphics is principally concerned with the generation of images, with wide ranging applications from entertainment to scientific visualisation. In other words, we can say that computer graphics is a rendering tool for the generation and manipulation of images. Thus, by using a computer as a rendering tool for the generation and manipulation of images is called computer graphics. Computer graphics is an art of drawing pictures on computer screens with the help of programming. It involves computations, creation, and manipulation of data. Graphics is a vast field that encompasses almost any graphical aspect like special effects, simulation and training, games, medical imagery, animations and much more. "*Computer Graphics refers to any sketch, drawing, special artwork or other material generated with the help of computer to pictorially depict an object or a process or convey information, as a supplement to or instead of written descriptions*".

It relies on an internal *model* of the scene, that is, a mathematical representation suitable for graphical computations. The model describes the 3D shapes, layout, projections to compute a 2D image from a given viewpoint and rendering which involves projecting the objects (perspective), handling visibility (which parts of objects are hidden) and computing their appearance and lighting interactions and materials of the scene.

1.2 HISTORY OF COMPUTER GRAPHICS

In the 1950's, graphics output were taken via teletypes, line printer, and Cathode Ray Tube (CRT). Using dark and light characters, a picture could be reproduced. In the 1960's, advent of modern interactive graphics, output were vector graphics and interactive graphics. One of the worst problems was the cost and inaccessibility of machines. In the early 1970's, output started using raster displays, graphics capability was still fairly chunky. In the 1980's output were built-in raster graphics, bitmap image and pixel. Personal computers costs decreased drastically; trackball and mouse become the standard interactive devices. In the 1990's, since the introduction of VGA and SVGA, personal computer could easily display photo-realistic images and movies. 3D image renderings became the main advances and it stimulated cinematic graphics applications. Table 1: gives a general history of computer graphics.

YEAR	DISCOVERY & FINDINGS
1950	Ben Laposky created the first graphic images, an Oscilloscope, generated by an electronic machine

1951	UNIVAC-I: the first general purpose commercial computer, crude hardcopy devices
	MIT – Whirlwind computer, the first to display real time video
1960	William Fetter coins the computer graphics to describe new design methods.
1961	Steve Russel developed Spacewars, the first video/computer game
1963	Douglas Englebart developed first mouse
	Ivan Sutherland developed Sketchpad, an interactive CG system
1964	William Fetter developed first computer model of a human figure
1965	Jack Bresenham designed line-drawing algorithm
1968	Tektronix – a special CRT, the direct-view storage tube, with keyboard and mouse
	Ivan Sutherland developed first head-mounted display
1969	John Warnock – area subdivision algorithm, hidden-surface algorithms
	Bell Labs – first framebuffer containing 3 bits per pixel
1972	Nolan Kay Bushnell – Pong, video arcade game
1973	John Whitney. Jr. and Gary Demos – "Westworld", first film with computer graphics
1974	Edwin Catmuff –texture mapping and Z-buffer hidden-surface algorithm
	James Blinn – curved surfaces, refinement of texture mapping
	Phone Bui-Toung – specular highlighting
1975	Martin Newell – famous CG teapot, using Bezier patches
	Benoit Mandelbrot – fractal/fractional dimension
1976	James Blinn – environment mapping
1977	Steve Wozniak -- Apple II, color graphics personal computer
1979	Roy Trubshaw and Richard Bartle – MUD, a multi-user dungeon/Zork
1982	Steven Lisberger – "Tron", first Disney movie which makes extensive use of 3-D graphics
	Tom Brighman – "Morphing", first film sequence plays a female character which deforms
	and transforms herself into the shape of a lynx.
	John Walkner and Dan Drake
	Jaron Lanier – "DataGlove", a virtual reality film.
1984	Wavefron tech. – Polhemus, first 3D graphics software
1985	Pixar Animation Studios – "Luxo Jr.", 1989, " Tin toy"
	NES – Nintendo home game system
1987	IBM – VGA, Video Graphics Array introduced
1989	Video Electronics Standards Association (VESA) – SVGA, Super VGA formed
1990	Hanrahan and Lawson – Renderman
1991	Disney and Pixar – "Beauty and the Beast", CGI was widely used, Renderman systems with high quality computer effects
1992	Silicon Graphics – OpenGL specification
1993	University of Illinois -- Mosaic, first graphic Web browser
	Steven Spielberg – "Jurassic Park" a successful CG fiction film.
1995	Buena Vista Pictures – "Toy Story", first full-length, computer-generated, feature film
2001	NVIDIA Corporation – GeForce 256, GeForce3
2003	ID Software – Doom3 graphics engine

Table 1 :- Development Lineage in the field of Computer Graphics

1.3 APPLICATIONS OF COMPUTER GRAPHICS

The core elements of computer graphics include the following :-

- **Modeling** :- Involves representation of choices, geometric processing
- **Rendering** :- Executes geometric transformation, visibility, implements simulation of light
- **Interaction** :- Synergy of input/output devices, tools
- **Animation** :- Simulation of lifelike characters, natural phenomena, their interactions, surrounding environments

Computer graphics can be broadly divided into the following classes:

- **Business Presentation Graphics**, which refers to graphics, such as bar-charts, histograms, pie-charts, pictograms, x-y charts, etc.
- **Scientific Graphics**, such as x-y plots, curve-fitting, contour plots, system or program flowcharts etc.
- **Cartography** (Scaled Drawings), such as architectural representations, drawings of buildings, bridges, and machines.
- **Satellite Imaging** − Geodesic images.
- **Photo Enhancement** − Sharpening blurred photos.
- **Medical imaging** − MRIs, CAT scans, etc. - Non-invasive internal examination.
- **Engineering drawings** − mechanical, electrical, civil, Replacing the blueprints of the past.
- **Typography** − The use of character images in publishing - replacing the hard type of the past.
- **Cartoons and artwork**, including advertisements.
- **Simulation and modeling** − Replacing physical modeling and enactments
- **Graphics User Interfaces (GUIs)** A graphic, mouse-oriented paradigm which allows the user to interact with a computer.The images that appear and are designed to help the user utilise the software without having to refer to manuals or read a lot of text on the monitor.

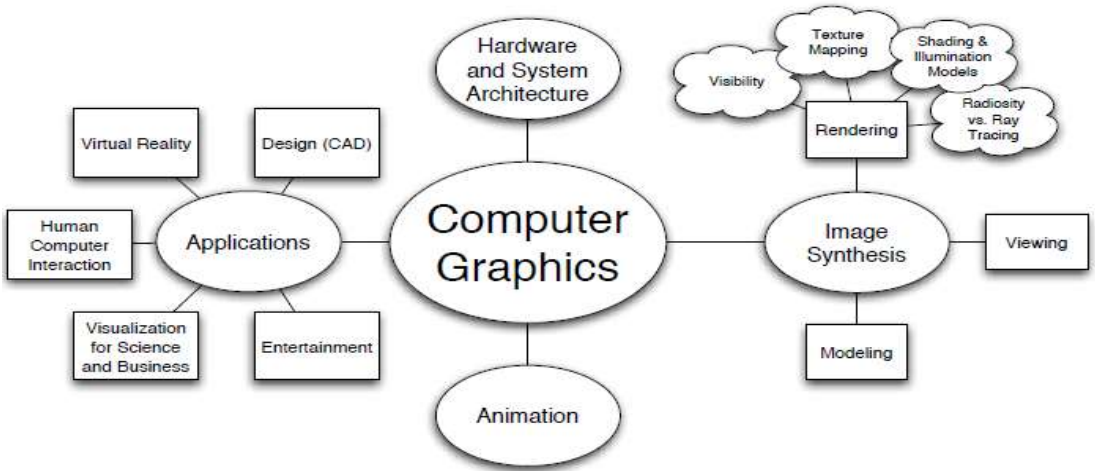


Fig :- Some Application Areas of Computer Graphics

Computer Graphics is used for a broad spectrum of applications, on a large number of different graphical devices and some of the main areas of its applications are :-

1.3.1 CAD and CAM processes

CAD is used to design, develop and optimize products, which can be goods used by end consumers or intermediate goods used in other products. CAD is also extensively used in the design of tools and machinery used in the manufacture of components, and in the drafting and design of all types of buildings, as it enables designers to layout and develop work on screen, print it out and save it for future editing, saving time on their drawings CAD is mainly used for detailed engineering of 3D models and/or 2D drawings of physical components, but it is also used throughout the engineering process from conceptual design and layout of products, through strength and dynamic analysis of assemblies to definition of manufacturing methods of components.

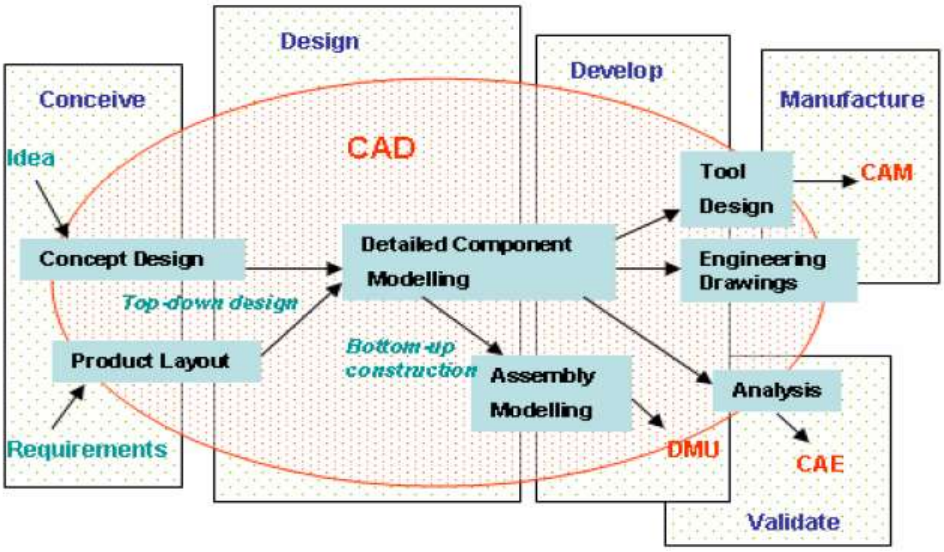


Fig:- The Process of Computer Aided Design

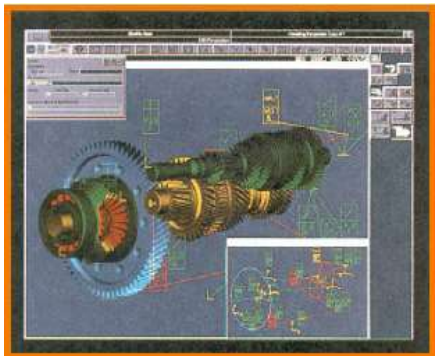
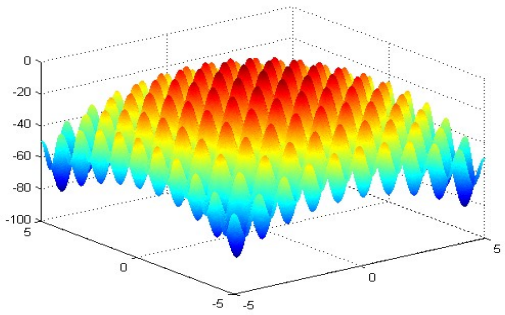


Fig :- Gear Shaft Design



Bench mark Bimodal Function

A CAM or Computer-Aided Manufacturing system usually seeks to assist and control the production process through varying degrees of automation. Because each of the many manufacturing processes in a CAM system is computer controlled, a high degree of precision can

be achieved that is not possible with a human interface. The CAM system, for example, sets the tool path and executes precision machine
operations based on the imported design. Another advantage of Computer Aided Manufacturing is that it can be used to facilitate mass customization: the process of creating small batches of products that are custom designed to suit each particular client. Without CAM, and the CAD process that precedes it, customization would be a time-consuming, manual and costly process. However, CAD software allows for easy customization and rapid design changes: the automatic controls of the CAM system make it possible to adjust the machinery automatically for each different order.

Fig :- The Computer Aided Manufacturing Process

1.3.2 Scientific Visualisation

It is difficult for the human brain to make sense out of the large volume of numbers produced by a scientific computation. Numerical and statistical methods are useful for solving this problem. Visualisation techniques are another approach for interpreting large data sets, providing insights that might be missed by statistical methods. As the volume of data accumulated from computations or from recorded measurements increases, it becomes more important that we be able to make sense out of such data quickly. Scientific visualisation, using computer graphics, is one way
to do this.

Scientific visualisation involve interdisciplinary research into robust and effective computer science and visualisation tools for solving problems in biology, aeronautics, medical imaging, and other disciplines. The profound impact of scientific computing upon virtually every area of science and engineering has been well established. The increasing complexity of the underlying mathematical models has also highlighted the critical role to be played by Scientific visualisation. Thus, Scientific visualisation is one of the most active and exciting areas of Mathematics and Computing Science, and indeed one which is only beginning to mature. Scientific visualisation is

a technology which helps to explore and understand scientific phenomena visually, objectively, quantitatively. Scientific visualization allow scientists to think about the unthinkable and visualise the unviable. This concept of scientific visualisation fits well with modeling and simulation.

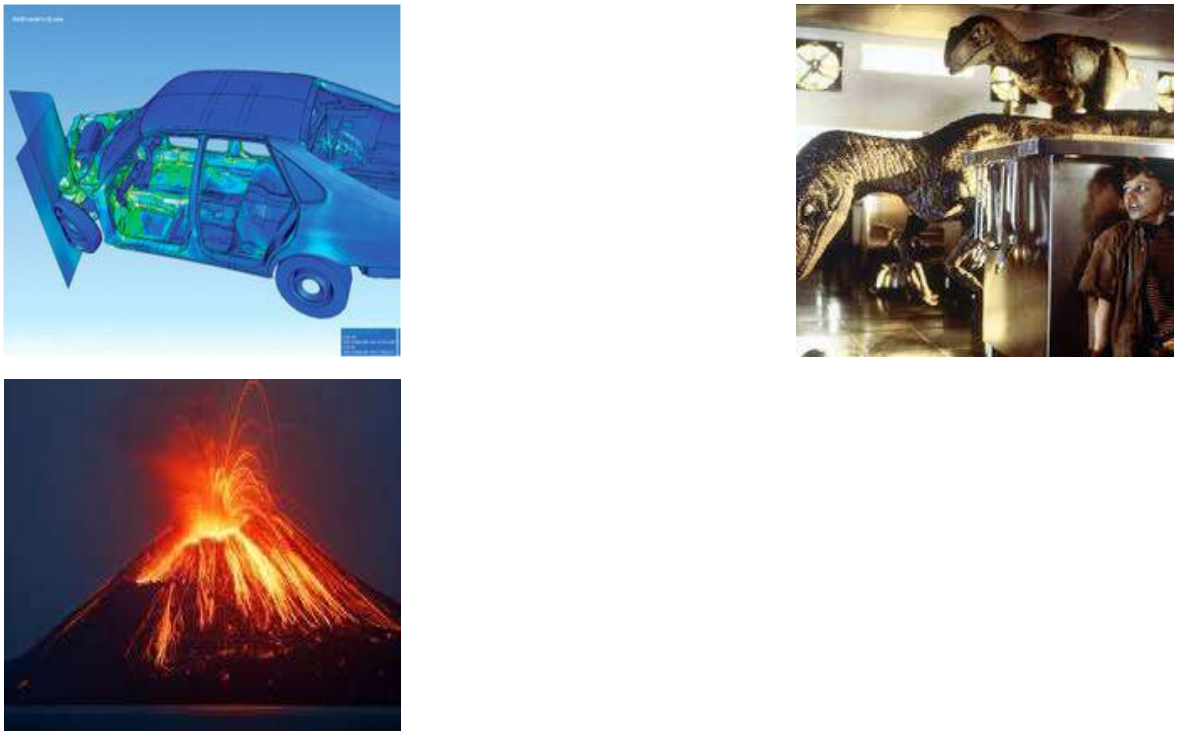


Fig : Visualization in areas of Car Crash, Jurassic Park, Volcano eruption

1.3.3 Entertainment and Photorealism

One of the main goals of today's special effects producers and animators is to create images with highest levels of photorealism. Volume graphics is the key technology to provide full immersion in upcoming virtual worlds e.g. movies or computer games.

Real world phenomena can be realized best with true physics based models and volume graphics is the tool to generate, visualize and even feel these models! Movies like Star Wars Episode I, Titanic and The Fifth Element already started employing true physics based effects.

1.3.4 Medical Content Creation

Medical content creation like virtual anatomical atlas on CD-ROM and DVD have been build on the base of the NIH Visible Human Project data set and different kind of simulation and training

software were build up using volume rendering techniques. Volume Graphics' products like the VGStudio software are dedicated to the used in the field of medical content creation. VGStudio provides powerful tools to manipulate and edit volume data. An easy to use keyframer tool allows to generate animations.

Fig : Graphic Representation of a foetus in the womb

1.4 GRAPHICS PACKAGE

A computer graphics system is a computer system which must have all the components of a general-purpose computer system. Considering the high-level view of a graphics system, there are six major elements in the Graphics system:

1. Input devices
2. Central Processing Unit
3. Graphics Processing Unit
4. Memory
5. Frame buffer
6. Output devices

This model is general enough to include workstations and personal computers, interactive game systems, mobile phones, GPS systems, and sophisticated image generation systems. Although most of the components are present in a standard computer, it is the way each element is specialized for computer graphics that characterizes this diagram as a portrait of a graphics system.

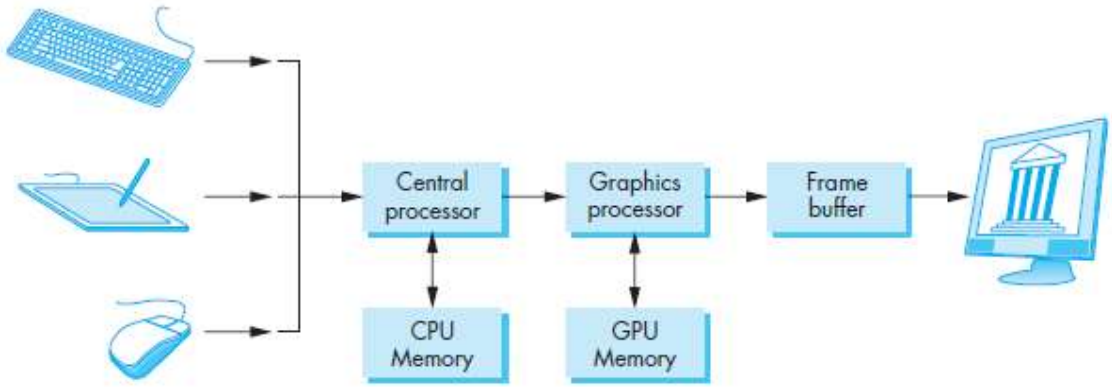


Fig : A Graphics System

1.4.1 Pixels and the Frame Buffer

Virtually all modern graphics systems are raster based. The image we see on the output device is an array—the **raster**—of picture elements, or **pixels**, produced by the graphics system.

An *image* that is presented on the computer screen is made up of **pixels**. The screen consists of a rectangular grid of pixels, arranged in rows and columns. The pixels are small enough that they are not easy to see individually. At a given time, each pixel can show only one color. Most screens these days use 24-bit color, where a color can be specified by three 8-bit numbers, giving the levels of red, green, and blue in the color. Any color that can be shown on the screen is made up of some combination of these three "primary" colors. Other formats are possible, such as *grayscale*, where each pixel is some shade of gray and the pixel color is given by one number that specifies the level of gray on a black-to-white scale. Typically, 256 shades of gray are used. Early computer screens used indexed color, where only a small set of colors, usually 16 or 256, could be displayed. The color values for all the pixels on the screen are stored in a large block of memory known as a ***frame buffer***. Changing the image on the screen requires changing color values that are stored in the frame buffer. The screen is redrawn many times per second, so that almost immediately after the color values are changed in the frame buffer, the colors of the pixels on the screen will be changed to match, and the displayed image will change. Its **resolution**—the number of pixels in the frame buffer—determines the detail that you can see in the image. The **depth**, or **precision**, of the frame buffer, defined as the number of bits that are used for each pixel, determines properties such as how many colors can be represented on a given system. For example, a 1-bit-deep frame buffer allows only two colors, whereas an 8-bit-deep frame buffer allows 28 (256) colors. In **full-color** systems, there are 24 (or more) bits per pixel. Such systems can display sufficient colors to represent most images realistically. They are also called **true-color** systems, or **RGB-color** systems, because individual groups of bits in each pixel are assigned to each of the three primary colors—red, green, and blue—used in most displays. **High dynamic range** (HDR) systems use

or more bits for each color component. Until recently, frame buffers stored colors in integer formats. Recent frame buffers use floating point and thus support HDR colors more easily.

Resolution	Number of Pixels	Aspect Ratio
320*200	64000	8:5
640*480	307200	4:3
800*600	480000	4:3
1024*768	786432	4:3
1280*1024	1310720	5:4
1600*1200	1920000	4:3

Fig :- The Relation between Resolution, Pixels and Aspect Ratio of a Graphical Device

1.4.2 Input Devices

Most graphics systems provide a keyboard and at least one other input device. The most common input devices are the mouse, the joystick, and the data tablet. Each provides positional information to the system, and each usually is equipped with one or more buttons to provide signals to the processor. Often called **pointing devices**, these devices allow a user to indicate a particular location on the display. Some commercial input devices used in Graphics System are

- 2D mice
- Scanners
- Light Pens
- Digitizers
- Digital Camera
- Video Camera
- Touch-sensitive screens
- Joysticks
- Trackballs
- Thumb wheels
- Microphones (voice data entry)
- Touch Panels

The **keyboard device** is a device that returns character codes. We use the American Standard Code for Information Interchange (ASCII) in our examples. ASCII assigns a single unsigned byte to each character. However in Internet applications, multiple bytes were used for each character, thus allowing for a much richer set of supported characters.

Touch Panels allow displayed object or screen positions to be selected with the touch of the finger and is also known as Touch Sensitive Screens (TSS). A typical application of touch panels is for

the selection of processing options that are represented with graphical icons. Touch input can be recorded using optical electrical or acoustical methods.

Optical touch panels employ a line of intra red LEDS (light emitting diodes) along one vertical edge and along one horizontal edge of frame. The opposite vertical and horizontal edges contain light detection which are used to record the beams that may have been interrupted when the panel was touched.

An electrical touch panel is constructed with two transparent plates separated by a short distance. One of the plates is coated with a conducting material and the other is resistive material. When the outer plate is touched, it is forced into contact with the inner plate. The contact creates a voltage drop that is converted to a coordinate value of the selected screen position. They are not too reliable or accurate, but are easy to use.

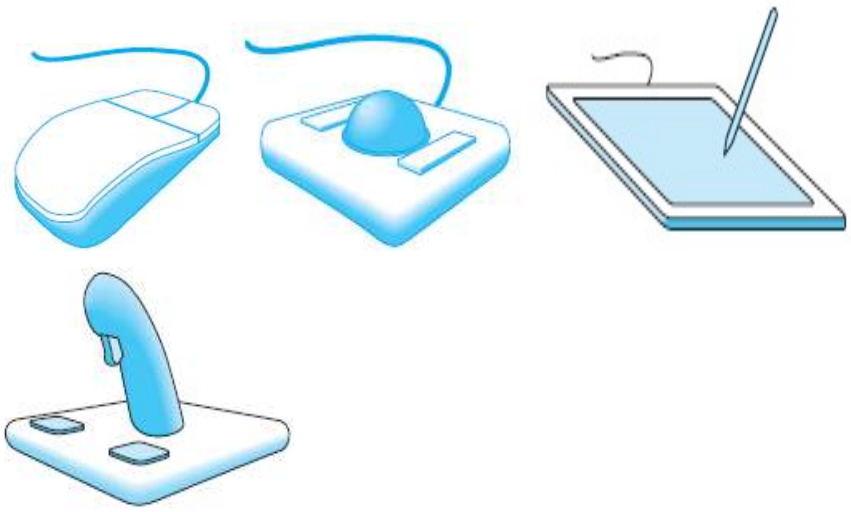


Fig : Some Input Devices in a Graphics System (Mouse, Trackball, Data tablet, Joystick)

Data tablets provide absolute positioning with rows and columns of wires embedded under its surface. The position of the stylus is determined through electromagnetic interactions between signals traveling through the wires and sensors in the stylus. Touch-sensitive transparent screens that can be placed over the face of a CRT have many of the same properties as the data tablet.

Small, rectangular, pressure-sensitive touchpads are embedded in the keyboards of many portable computers. One other device, the **joystick** regulates its motion of the stick in two orthogonal directions by encoding and interpreting as two velocities, and integrated to identify a screen location. The integration implies that if the stick is left in its resting position, there is no change in the cursor position and that the farther the stick is moved from its resting position, the faster the screen location changes. Thus, the joystick is a variable-sensitivity device.

1.4.3 Output Devices

One of the common physical output devices for display (or **monitor**) was the **cathode-ray tube** (**CRT**). A simplified picture of a CRT is shown in figure below which shows the working principle of CRT. When electrons strike the phosphor coating on the tube, light is emitted. The direction of the beam is controlled by two pairs of deflection plates. The output of the computer is converted, by digital to-analog converters, to voltages across the *x* and *y* deflection plates. Light appears on the surface of the CRT when a sufficiently intense beam of electrons is directed at the phosphor.

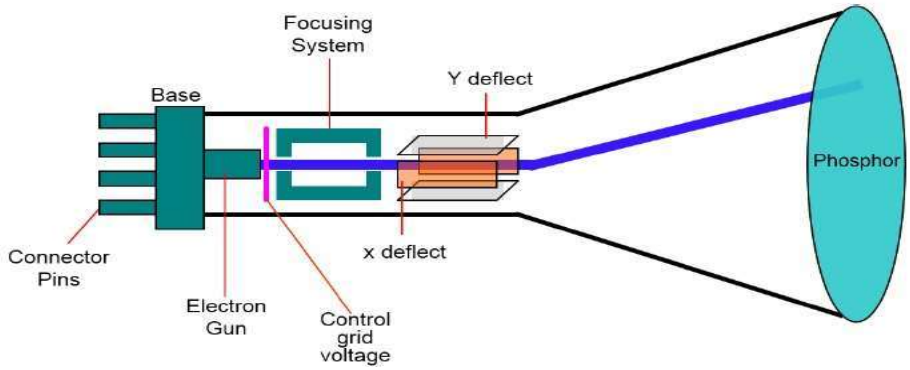


Fig : The Working Principle of a Cathode Ray Tube

A computer screen used in this way is the basic model of **raster graphics**. The term "raster" technically refers to the mechanism used on older vacuum tube computer monitors: An electron beam would move along the rows of pixels, making them glow. The beam was moved across the screen by powerful magnets that would deflect the path of the electrons. The stronger the beam, the brighter the glow of the pixel, so the brightness of the pixels could be controlled by modulating the intensity of the electron beam. The color values stored in the frame buffer were used to determine the intensity of the electron beam. The idea of an image consisting of a grid of pixels, with numerical color values for each pixel, defines raster graphics in a modern flat screen too.

The performance parameters of a monitor are:

 Luminance, measured in candelas per square metre (cd/m²).

 Size, measured diagonally. For CRT the viewable size is one inch (25 mm) smaller than the tube itself.

 Dot pitch,describes the distance between pixels of the same color in millimetres. In general, the lower the dot pitch (e.g. 0.24 mm, which is also 240 micrometres), the sharper the picture.

 Response time. The amount of time a pixel in an LCD monitor takes to go from active (black) to inactive (white) and back to active (black) again. It is measured in milliseconds (ms). Lower numbers mean faster transitions and therefore fewer visible image artifacts.

- **Refresh rate**. The number of times in a second that a display is illuminated.
- **Power consumption**, measured in watts (W).

- **Aspect ratio**, which is the horizontal size compared to the vertical size, e.g. 4:3 is the standard aspect ratio, so that a screen with a width of 1024 pixels will have a height of 768 pixels.
- **Display resolution**. The number of distinct pixels in each dimension that can be displayed.

The formula to calculate the video memory required at a given resolution and bit-depth is :-
Memory (in MB) = (X-resolution * Y-resolution * Bit per pixel) / (8*1024*1024)

Display systems use either random or raster scan:
Random scan displays, often termed **vector displays**, came first and are still used in some applications. Here the electron gun of a CRT illuminates points and/or straight lines in any order. The display processor repeatedly reads a variable 'display file' defining a sequence of X,Y coordinate pairs and brightness or colour values, and converts these to voltages controlling the electron gun. In a Random Scan System, the Display buffer stores the picture information.
Further, the device is capable of producing pictures made up of lines but not of curves. Thus, it is also known as "Vector display device or Line display device or Calligraphic display device".

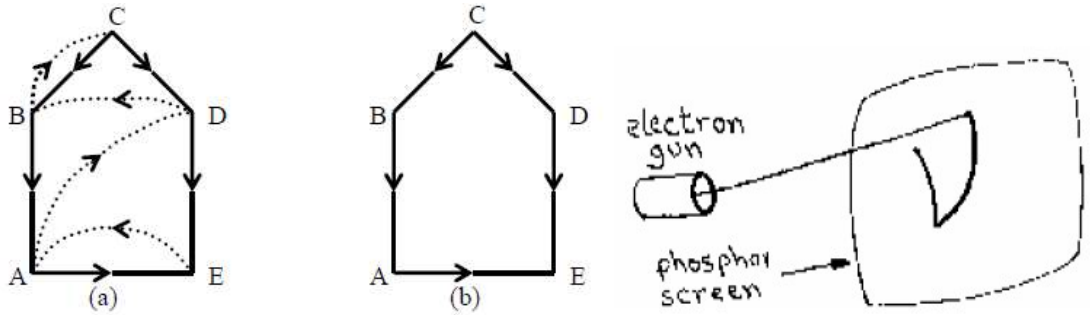


Fig : A Random Scan Display (……. Beam off;____ Beam on)

Raster scan displays, also known as bit-mapped or raster displays, are somewhat less relaxed. Their whole display area is updated many times a second from image data held in raster memory. The rest of this handout concerns hardware and software aspects of raster displays. In a raster scan, an image is cut up into successive samples called pixels, or picture elements, along *scan lines*. Each scan line can be transmitted as it is read from the detector, as in television systems, or can be stored as a row of pixel values in an array in a computer system. Each complete sweep from top left to bottom right of the screen is one complete cycle, called the **Refresh Cycle**.
Refreshing on raster-scan displays is carried out at the rate of 60 to 80 frames per second, although some systems are designed for higher refresh rates. Sometimes, refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame. Using these units, we would describe a refresh rate of ***60*** frames per second as simply 60 Hz. At the end of each

scan line, the electron beam returns to the left side of the screen to begin displaving the next scan line. The return to the left of the screen, after refreshing each scan line, is called the *horizontal retrace* of the electron beam. And at the end of each frame (displayed in 1/80th to 1/60th of a second), the electron beam returns (*vertical retrace*) to the top left comer of the screen to begin the next frame. On some raster-scan systems (and in **TV** sets), each frame is displayed in two passes using an interlaced refresh procedure. In the first pass, the beam sweeps across every other scan line from top to bottom. Then after the vertical retrace, the beam sweeps out the remaining scan lines. *Interlacing* of the scan lines in this way allows us to see the entire image displayed in one-half the time it would have taken to sweep all the lines at once from top to bottom.

Interlacing is primarily used with slower refreshing rates.

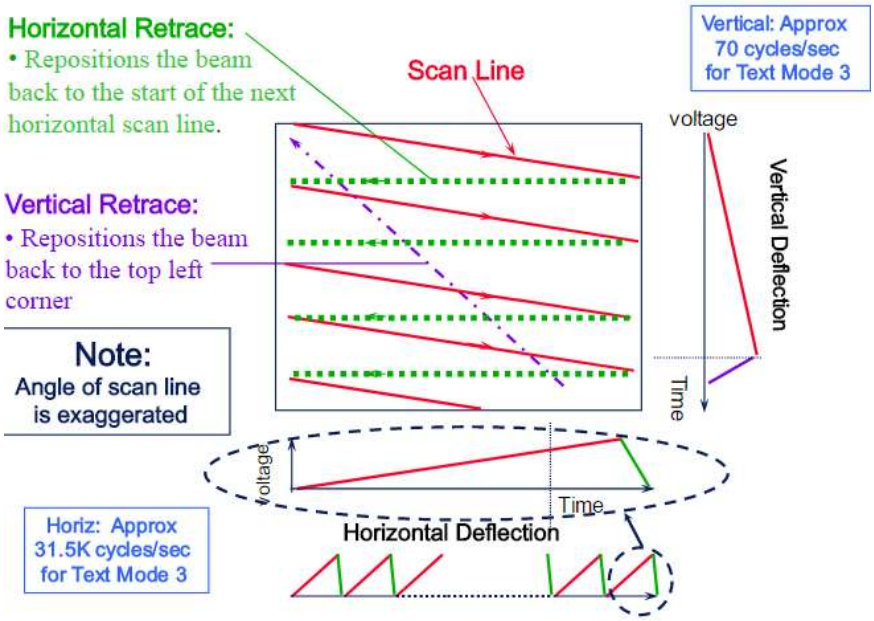


Fig :- Horizontal and Vertical Retrace in a non-Interlaced Raster Display

Color CRTs have three different colored phosphors (red, green, and blue), arranged in small groups like in triangular groups called **triads**, each triad consisting of three phosphors. In the shadow-mask CRT , a metal screen with small holes—the **shadow mask**—ensures that an electron beam excites only phosphors of the proper color.

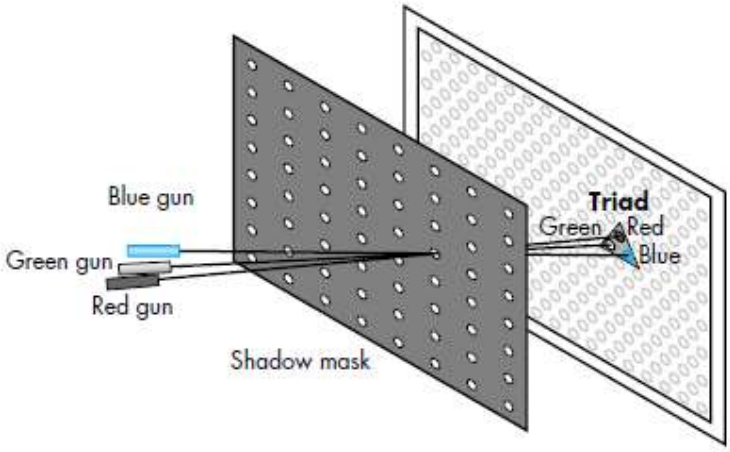


Fig :- A Shadow-Mask CRT

Although CRTs are still common display devices, they are rapidly being replaced by flat-screen technologies. Flat-panel monitors are inherently raster based. Although there are multiple technologies available, including light-emitting diodes (LEDs), liquid-crystal displays (LCDs), and plasma panels, all use a two-dimensional grid to address individual light-emitting elements. The two outside plates each contain parallel grids of wires that are oriented perpendicular to each other. By sending electrical signals to the proper wire in each grid, the electrical field at a location, determined by the intersection of two wires, can be made strong enough to control the corresponding element in the middle plate. The middle plate in an LED panel contains light-emitting diodes that can be turned on and off by the electrical signals sent to the grid. In an LCD display, the electrical field controls the polarization of the liquid crystals in the middle panel, thus turning on and off the light passing through the panel. A plasma panel uses the voltages on the grids to energize gases embedded between the glass panels holding the grids. The energized gas becomes a glowing plasma.

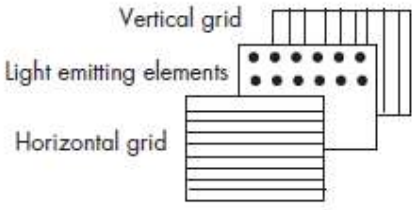


Fig :- A Generic Flat-Panel Display

Plotter: A plotter is a vector graphics-printing device that connects to a computer. Now-a-days, we use the plotter right from the field of engineering, to media and advertising. Even in our day-to-day lives we see a large number of computer designed hoardings and kiosks as publicity material. This fine output is achieved byusing plotters with computers.

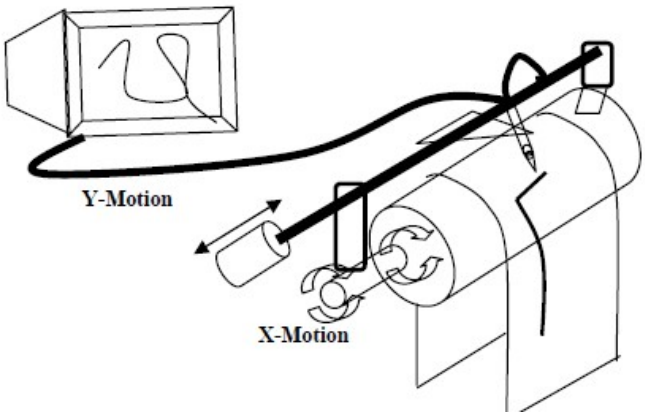


Fig : A Drum Plotter

1.4.4 The CPU and the GPU

In a simple system, there may be only one processor, the **central processing unit** (**CPU**) of the system, which must do both the normal processing and the graphical processing. The main graphical function of the processor is to take specifications of graphical primitives (such as lines, circles, and polygons) generated by application programs and to assign values to the pixels in the frame buffer that best represent these entities. For example, a triangle is specified by its three vertices, but to display its outline by the three line segments connecting the vertices, the graphics system must generate a set of pixels that appear as line segments to the viewer. The conversion of geometric entities to pixel colors and locations in the frame buffer is known as **rasterization**, or **scan conversion**. In early graphics systems, the frame buffer was part of the standard memory that could be directly addressed by the CPU. Today, virtually all graphics systems are characterized by special-purpose **graphics processing units** (**GPUs**), to carry out specific graphics functions. The GPU can be either on the mother board of the system or on a graphics card. The frame buffer
is accessed through the graphics processing unit and usually is on the same circuit board as the GPU. GPUs are characterized by both special-purpose modules geared toward graphical operations and a high degree of parallelism—recent GPUs contain over 100 processing
units, each of which is user programmable. GPUs are so powerful that they can often be used as mini supercomputers for general purpose computing.

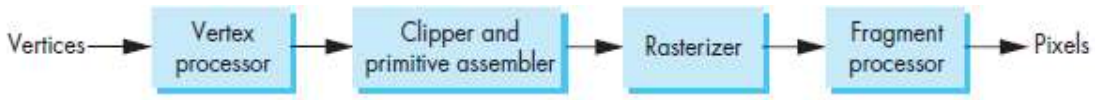


Fig : The Graphic Processor Architecture

1.4.5 The Graphics Pipeline

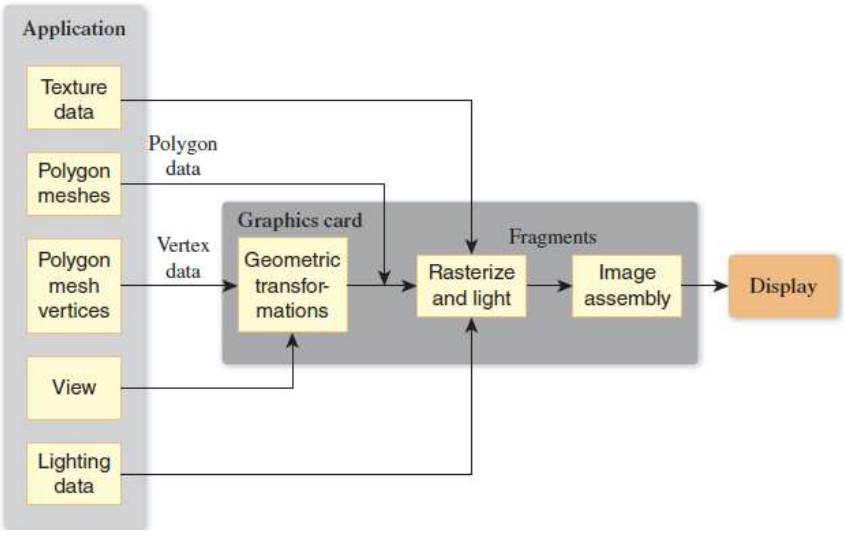


Fig :- The :Pipeline Architecture of Graphics Package

The functioning of a standard graphics system is typically described by an abstraction called the **graphics pipeline.** The term "pipeline" is used because the transformation from mathematical model to pixels on the screen involves multiple steps, and in a typical architecture, these are performed in sequence; the results of one stage are pushed on to the next stage so that the first stage can begin processing the next polygon immediately.

Each object comprises a set of graphical primitives. Each primitive comprises a set of vertices. We can think of the collection of primitive types and vertices as defining the **geometry** of the scene. In a complex scene, there may be thousands—even millions—of vertices that define the objects. We must process all these vertices in a similar manner to form an image in the frame buffer. If we think in terms of processing the geometry of our objects to obtain an image, we can employ the four major steps in the imaging process:

1. Vertex processing
2. Clipping and primitive assembly
3. Rasterization
4. Fragment processing

No matter with which advance graphic software you are working with, if your output device or Graphics hardware is not good, or hardware handling that software is not good, then ultimate result will be not good, as discussed in the previous section to work with graphic packages.

1.5 GRAPHICS SOFTWARE IMPLEMENTS

The programmer who sets out to write a graphics program has a wide choice of starting points. Because graphics cards—the hardware that generates data to be displayed on a screen—or their equivalent chipsets vary widely from one machine to the next, it's typical to use some kind of software abstraction of the capabilities of the graphics card. This abstraction is known as an **application programming interface** or **API.** A graphics API could be as simple as a single function that lets you set the colors of individual pixels on the display, or it could be as complex as a system in which the programmer describes an illuminated scene with high-level objects and their properties. Often such high-level APIs are just a part of a larger system for application development, such as modern game engines.

1.5.1 Software Tools for Image Processing

The Mathematical tools for the processing of digital images include *convolution*, *Fourier analysis*, and *statistical* descriptions, and manipulative tools such as *chain codes* and *run codes*. But these tools are worked with at very core levels, in general we use some software to process the image with the help of computers. Some of the categories of image processing software with their respective examples and features are listed below:

1) ***Graphics Image Processing: Software used*** : Photoshop.
• Most common image processing software.
• Focuses on creating a pretty picture.
• Usually limited to popular graphics formats such as: TIFF, JPEG, GIF
• Best suited for working with RGB (3-band) images.
• Does not treat an image as a "map".

2) ***Geographic Information Systems (GIS): Software used*** : ArcMap
• Works within a geographic context.
• Great for overlaying multiple vector and raster layers.
• More common than remote sensing software.

3) ***Remote Sensing Packages: Software used***: ERDAS
• Best suited for satellite imagery.
• Uses geo-spatial information.
• Easily works with multi-spectral data.
• Provides analysis functions commonly used for remote sensing applications.

• Often easy to use but it helps to be familiar with remote sensing.

4) ***Numerical Analysis Packages: Software used***: Matlab.

• Focus usually on numeric processing.

• Programming or mathematical skills usually helpful.

• Used to build more user-friendly applications.

5) ***Web-based Services: Software used*** : Protected Area Archive.

• Image display, roam, zoom.

• Image enhancement.

• Simple image processing.

• Distance and area measurement.

• Comparison of old and new images.

• Image annotation (adding text, lines, etc).

• Overlaying vector layers.

6) Computer Aided Design and Drafting (CADD): *Software used:* AutoCad,SolidWorks

CAD/CAM software uses CAD drawing tools to describe geometries used by the CAM portion of the program to define a tool path that will direct the motion of a machine tool to machine the exact shape that is to be drawn on the computer. Now-a-days many new machine tools incorporate CNC technologies. These tools are used in every conceivable manufacturing sector, like CNC technology is related to Computer Integrated Manufacturing (CIM), Computer Aided Process Planning (CAPP) and other technologies such as Group Technology (GT) and Cellular Manufacturing. Flexible Manufacturing Systems (FMS) and Just-In- Time Production (JIT) are made possible by Numerically-Controlled Machines.

1.5 NUMERICAL EXAMPLES

1. Consider a raster system with resolution 1280 x 1024 and 2560 x 2048. What size frame buffer (in bytes) is needed for the system to store 12 bits/pixel? How much storage is required for each system if 24 bits per pixel are to be stored?

   Solution :

   Frame-buffer size for the system is

   1280 × 1024 × 12 bits ÷ 8 bits per byte = 1920 KB

   2560 × 2048 × 12 bits ÷ 8 bits per byte = 7680 KB

   For 24 bits of storage per pixel, each of the above values is doubled.

2. Consider a raster system with the resolution of 1024 x 768 pixels and the color palette calls for 65,536 colors. What is the minimum amount of video RAM that the computer must have to support the above-mentioned resolution and number of colors?

Solution :-

Recall that the color of each pixel on a display is represented with some number of bits. Hence, a display capable of showing up to 256 colors is using 8 bits per pixels (i.e. "8-bit color"). Notice first that the color palette calls for 65,536 colors. This number is but 216 , which implies that 16 bits are being used to represent the color of each pixel on the display. The display's resolution is 1024 by 768 pixels, which implies that there is a total of 786,432 (1024 × 768) pixels on the display. Hence, the total number of bits required to display any of 65,536 colors on each of the screen's 786,432 pixels is 12,582,912 (786,432 × 16). Dividing this value by 8 yields an answer of 1,572,864 bytes. Dividing that value by 1,024 yields an answer of 1,536 KB. Dividing that value by 1,024 yields an answer of 1.5 MB.

3. How many Kilobytes does a frame buffer need in a 600 x 400 pixel ?

Solution :

Resolution is 600 x 400 Suppose 1 pixel can store n bits Then, the size of frame buffer = Resolution X bits per pixel = (600 X 400) X n bits = 240000 n bits

= 240000 n/1024 X 8 = 29.30 n KB (as 1kb = 1024 bytes)

4. Find out the aspect ratio of the raster system using 8 x 10 inches screen and 100 pixel/inch.

Solution :-

Aspect ratio = Width : Height = (8 x 100) /(10 x 100) Aspect ratio = 4 : 5

5. How much time is spent scanning across each row of pixels during screen refresh on a raster system with resolution of 1280 X 1024 and a refresh rate of 60 frames per second?

Solution :-

Here, resolution = 1280 X 1024 that means system contains 1024 scan lines and each scan line contains 128 pixels refresh rate = 60 frame/sec. So, 1 frame takes = 1/60 sec. Since resolution = 1280 X 1024 1 frame buffer consist of 1024 scan lines It means then 1024 scan lines takes 1/60 sec Therefore, 1 scan line takes , 1 /(60 X 1024) = 0.058 sec

1.6 SUMMARY

In this chapter, we have set the stage for our top-down development of computer graphics. We have stressed that computer graphics is a discipline with multi-faceted Applications and have covered the following areas :-

a) Introduction to Computer Graphics

b) History of Computer Graphics

c) Applications of Computer Graphics

d) The need and use of Computer Graphics in the modern world

1.7 QUESTIONS FOR EXERCISE

1) Discuss about the application of computer graphics in entertainment.

2) Discuss about the application of computer graphics in visualization.

3) What do you mean by interactive computer Graphics?

4) Define persistence,Resolution, Aspect Ratio,Frame Buffer.

5) What do you mean by retracing? Define horizontal as well as vertical retracing?

6) Consider three different raster systems with resolutions of 640 x 480, 1280 x 1024 and 2560 x 2048. What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel? How many pixels could be accessed per second in each of these systems by a display controller that refreshes the screen at a rate of 60 frames per second?

Hint : 640 x 480 x 12 bits / 8 = 450KB ; (640 x 480) * 60 = 1.8432 x 107 pixels/second.

7) What is the size of a pixel on a 21-inch diagonal screen with physical aspect ratio 8:5 operating in 1152 x 800 mode?

1.8 SUGGESTED READINGS

William M. Newman, Robert F. Sproull, “Principles of InteractiveComputer Graphics”, Tata-McGraw Hill, 2000

Donald Hearn, M. Pauline Baker, “Computer Graphics – C Version”, Pearson Education, 2007

Chapter 1, 2, 3 of ISRD Group, “Computer Graphics”, McGraw Hill, 2006

J.D. Foley, A.Dam, S.K. Feiner, J.F. Hughes, “Computer Graphics – principles and practice”, Addison-Wesley, 1997

Andy Johnson's CS 488 Course Notes, Lecture 1

UNIT -2 GRAPHICS TECHNIQUES

UNIT STRUCTURE

2.0 OBJECTIVE

The unit describes the Line Generation,Circle generation Algorithms and Clipping ;

• To Apply Line Generation Algorithm to practical problems;

• Describe the different types of Line Generation Algorithm;

• Describe Circle Generation Algorithm;

• apply Circle Generation algorithm to practical problems;

• describe the different types of Circle Generation Algorithms;

• describe Polygon fill algorithm, and

• describe Scan Line Polygon fill algorithm.

2.1 GRAPHICS PRIMITIVES

Scan conversion is the process of converting basic, low level objects into their corresponding pixel map representations. This is often an approximation to the object, since the frame buffer is a discrete grid. Each pixel on the display surface has a finite size depending on the screen resolution and hence, a pixel cannot represent a single mathematical point.

However, we consider each pixel as a unit square area identified by the coordinate of its lower left corner, the origin of the reference coordinate system being located at the lower left corner of the display surface. Thus, each pixel is accessed by a non-negative integer coordinate pair (*x*, *y*). The x values start at the origin and increase from left to right along a scan line and the y values

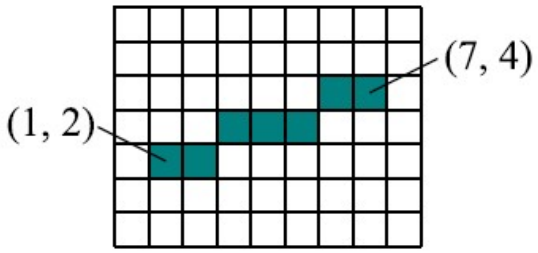


Fig :- Pixel representation of point primitive

Primitive operations involved in C or OpenGL are :

• setpixel(x, y, color) : Sets the pixel at position (x, y) to the given color.

• getpixel(x, y) :Gets the color at the pixel at position (x, y).

2.2 Line Drawing

A line connects two points.Line drawing is accomplished by calculating the intermediate point coordinates along the line path between two given end points. Since, screen pixels are referred with integer values, plotted positions may only approximate the calculated coordinates – i.e., pixels which are intensified are those which lie very close to the line path if not exactly on the line path which is in the case of perfectly horizontal, vertical or 45° lines only. Standard algorithms are available to determine which pixels provide the best approximation to the desired line.

The explicit equation for a line is **y = mx + b.**

The Basic Line Drawing Algorithm involves the following steps

Step 1 : Set the color of pixels to approximate the appearance of a line from (x_0, y_0) to (x_1, y_1).

Step 2 : Subtract y_0 from y_1 to solve for $m = (y1-y0)/(x1-x0)$ and $b = y_0 - mx_0$.

Step 3 : Substituting in the value for b, this equation can be written as $y = m(x - x_0) + y_0$.

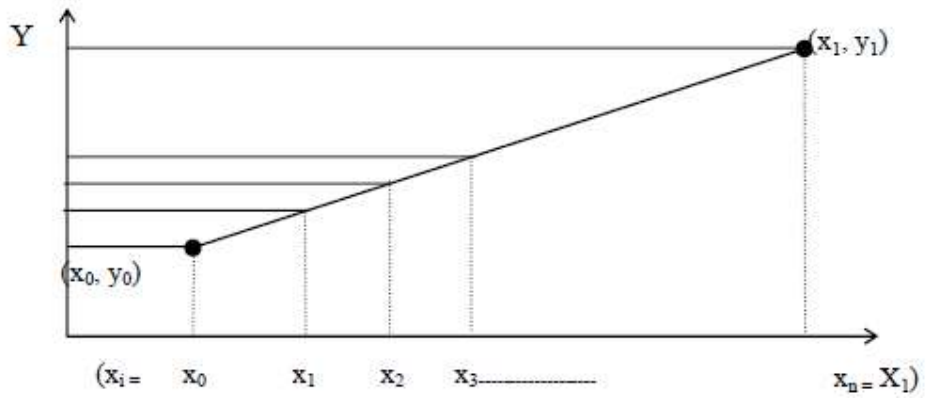


Fig : Slope m of the Straight line < 1

2.2.1 Digital Differential Analyzer

Digital Differential Analyzer (DDA) algorithm is the simple line generation algorithm which is explained step by step here.

Step 1 − Get the input of two end points (X_0,Y_0) and (X_1,Y_1).

Step 2 − Calculate the difference between two end points.

```
dx = X1 - X0
dy = Y1 - Y0
```

Step 3 − Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If dx > dy, then you need more steps in x coordinate; otherwise in y coordinate.

```
if (absolute(dx) > absolute(dy))
  Steps = absolute(dx);
else
  Steps = absolute(dy);
```

Step 4 − Calculate the increment in x coordinate and y coordinate.

```
Xincrement = dx / (float) steps;
Yincrement = dy / (float) steps;
```

Step 5 − Put the pixel by successfully incrementing x and y coordinates accordingly and complete the drawing of the line.

```
for(int v=0; v < Steps; v++)
{
```

```
  x = x + Xincrement;
  y = y + Yincrement;
  putpixel(Round(x), Round(y));
}
```

Example 1: Draw line segment from point (2, 4) to (9, 9) using DDA algorithm.

Solution: We know general equation of line is given by

y = mx+c where m =(y1 – y0) / (x1 – x0)

given (x0, y0) → **(2, 4)** ; (x1, y1) → (9, 9)

⇒ m = (y1 – y0)/(x1 – x0) = 9-4 / 9-2

= 5/7 i.e., 0 < m < 1

Thus updated pixel values are :- $\mathbf{x_{i+1} = x_i + 1}$ $\mathbf{y_{i+1} = y_i + m}$

given (x0, y0) = (2, 4)

1) x1 = x0 + 1 = 3 and y1 = y0 + m = 4 + 5/7 = 4.71

So, put pixel (x1, round(y1), colour)

i.e., put on **(3, 5)**

2) x2 = x1 + 1 = 3 + 1 = 4

y2 = y1 + m = (33/7) + 5/7 = 38/7=5.428

i.e. put on **(4, 5)**

Similarly go on till (9, 9) is reached.

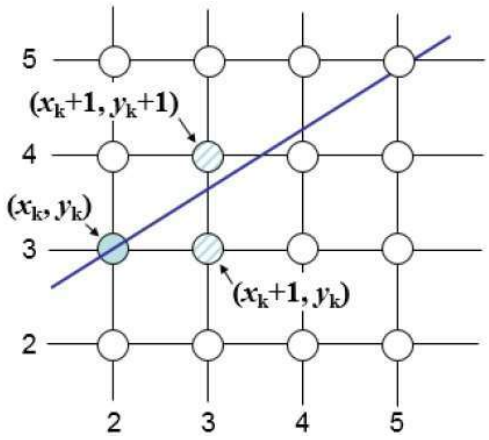
Drawbacks of DDA :

- Floating point values (m,y)
- Round operation
- Special cases m = 0 or infinity

2.2.2 Bresenham's Line Generation

The Bresenham algorithm is another incremental scan conversion algorithm. It is an accurate and efficient raster line generation algorithm. This algorithm scan converts lines using only incremental integer calculations which is a big advantage of this algorithm and these calculations can also be adopted to display circles and other curves moving across the x axis in unit intervals and at each step choose between two different y coordinates.

For example, as shown in the following illustration, from position (2, 3) you need to choose between (3, 3) and (3, 4). You would like the point that is closer to the original line.



At sample position X_{k+1},Y_{k+1}, the vertical separations from the mathematical line are made.

$d_{new} = F(x_p+2, y_p+1/2) = a(x_p+2) + b(y_p+1/2) + c$

but $d_{old} = a(x_p+1) + b(y_p+1/2) + c$

Now assuming we have to determine that the pixel at (x_k , y_k) is to be displayed, we next need to divide which pixel to plot in column xk+1. Our choices are the pixels at position **(x_k+1 , y_k) and (x_k+1 , y_k+1).**

At sampling position x_k+1, we label vertical pixel separations from the mathematical line path as d**1** and d**2**. . The y coordinate on the mathematical line at pixel column position x_k+1 is calculated as : $y = m(x_k + 1) + b$

Then $d1 = y - y_k = m(x_k + 1) + b - y_k$

and $d2 = (y_k + 1) - y = y_k + 1 - m(x_k + 1) - b$

The difference between these two separations is

$d1 - d2 = 2m(x_k+1) - 2y_k + 2b - 1$

A decision Parameter $\mathbf{P_k}$ for the k^{th} step in the line algorithm can be obtained by rearranging and by substituting m = Δy/Δx. where Δy & Δx are the vertical & horizontal separation of the endpoint positions & defining.

$P_k = \Delta x\,(d1 - d2) = 2\Delta y.\, x_k - 2\Delta x\, y_k + c$ _ _ _ _____(A)

The sign of P_k is same as the sign of d1 - d2.

This recursive calculation of decision Parameter is performed each integer x position, starting at left coordinate endpoint of the line. The first parameter P_0 is evaluated from equation (A) at starting pixel position (x_0, y_0) and with m evaluated as Δy/Δx.

$P_0 = 2\Delta y - \Delta x$ _ _ _ _____(B)

Since Δx > 0 for our example Parameter C is constant & has the value 2Δy + Δx (2b -1), which is independent of pixel position. If the pixel position at y_k is closer to line path than the pixel at y_k+1 (that is d1 < d2), then decision Parameter P_k is Negative. In that case we plot the lower pixel otherwise we plot the upper pixel.

i.e. $\mathbf{P_{k+1}=P_K + 2* \Delta y;}$

else when Decision Parameter $\mathbf{P_k > 0}$

$\mathbf{y_{k+1}=y_k+1;}$ $\mathbf{x_{k+1} = x_k+1}$

and $\mathbf{P_{k+1}=P_K+ 2*(\Delta y - \Delta x);}$

Coordinate changes along the line owner in unit steps in either the x or directions. Therefore we can obtain the values of successive decision Parameter using incremental integer calculations.

Example 2: Digitize the line with end points (20, 10) & (30, 18) using Bresenham's Line Drawing Algorithm

Hint Solution :

$\Delta x = 10$, $\Delta y = 8$

Initial decision parameter has the value

$P_0 = 2\Delta y - \Delta x = 2x8 – 10 = 6$

Since P0 > 0, so next point is (xk + 1, yk + 1) (21, 11)

Now k = 0, Pk+1 = Pk + 2Δy - 2Δx

P1 = P0 + 2 Δ y - 2 Δ x
= 6 + (-4)
= 2
Since P1 > 0, Thus , Next point is (22, 12)
Now k = 1, Pk+1 = Pk + 2 Δ y - 2 Δ x
P2 = 2 + (- 4)
= - 2
Since P2 < 0, Thus, Next point is (23, 12)
Now k = 2 Pk+1 = Pk + 2 Δ y
P2 = - 2 + 16
= 14
Since P3 > 0, Thus, Next point is (24, 13)
Now k = 3 Pk+1 = Pk + 2 Δ y - 2 Δ x
P4 = 14 – 4
= 10
Since P4 > 0, So, Next point is (25, 14)
Now k = 4 Pk+1 = Pk + 2 Δ y - 2 Δ x
P5 = 10 – 4
= 6
Since P5 > 0, □□Next point is (26, 15)
Now k = 5 Pk+1 = Pk + 2 Δ y - 2 Δ x
P6 = 6 – 4
= 2
Since P6 > 0, So,Next point is (27, 16)
Now k = 6 Pk+1 = Pk + 2 Δ y - 2 Δ x
P7 = 2 + (- 4)
= - 2
Since P7 < 0, So, Next point is (28, 16)
Now k = 7 Pk+1 = Pk + 2 Δ y
P8 = - 2 + 16
= 14

Since P8 > 0, So, Next point is (29, 17)
Now k = 8 Pk+1 = Pk + 2 Δ y - 2 Δ x
P9 = 14 – 4
= 10
Since P9 > 0, So, Next point is (30, 18)
i.e. To Summarize the Raster Points

K	Pk	xk+1	yk+1
0	6	21	11
1	2	22	12
2	-2	23	12
3	14	24	13
4	10	25	14
5	6	26	15
6	2	27	16
7	-2	28	16
8	14	29	17
9	10	30	18

2.3 Circle Drawing Algorithm

A circle is a set of points that are at a given distance r form the center position (x_c, y_c). This distance relationship is given as :

$(x – x_c)^2 + (y – y_c)^2 – r^2 = 0$

We cannot display a continuous arc on the raster display. Instead, we have to choose the nearest pixel position to complete the arc.

This equation is used to calculate the position of points along the circle path by moving in the x direction from (x_c - r) to (x_c + r) and we have put the pixel at (X, Y) location and now need to decide where to put the next pixel − at N (X+1, Y) or at S (X+1, Y-1).

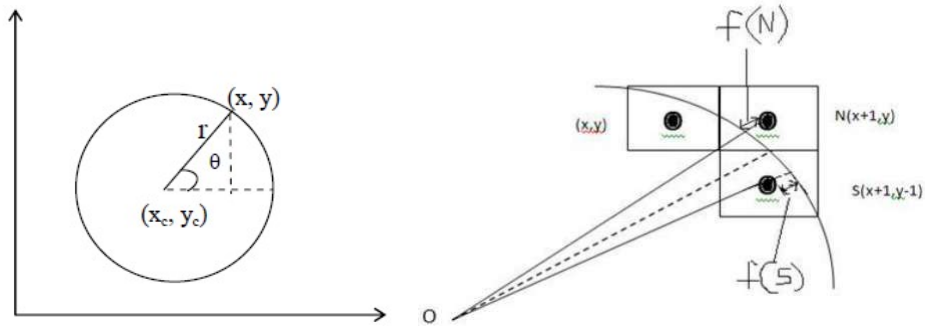


Fig :- Circle Generation Algorithm with polar coordinates

By calculating the polar coordinates r and θ where

$x = x_c + r \cos \theta$

$y = y_c + r \sin \theta$

Thus, an efficient approach based on incremental calculations of decision parameter is employed for circle generation. . There are two popular algorithms for generating a circle − **Bresenham's Algorithm** and **Midpoint Circle Algorithm**. These algorithms are based on the idea of determining the subsequent points required to draw the circle. For a given radius and center position (x, y,) we first setup our algorithm to calculate pixel position around the path of circle centered at coordinate origin (0, 0) *i.e.,* we translate

(*xc*, *yc*) → (0, 0) and after the generation we do inverse translation (0, 0) → (*xc*, *yc*)

hence each calculated position (*x*, *y*) of circumference is moved to its proper screen position by adding *xc* to *x* and *yc* to *y*.

2.3.1 Bresenham's Circle Generation

Here, the decision parameter **d** is used to rasterize the next point on the circle.

- If d <= 0, then N(X+1, Y) is to be chosen as next pixel.
- If d > 0, then S(X+1, Y-1) is to be chosen as the next pixel.

Step 1 − Get the coordinates of the center of the circle and radius, and store them in x, y, and R respectively. Set P=0 and Q=R.

Step 2 − Set decision parameter D = 3 – 2R.

Step 3 − Repeat through step-8 while X < Y.

Step 4 − Call Draw Circle (X, Y, P, Q).

Step 5 − Increment the value of P.

Step 6 − If D < 0 then D = D + 4x + 6.

Step 7 − Else Set Y = Y + 1, D = D + 4(X-Y) + 10.

Step 8 − Call Draw Circle (X, Y, P, Q).

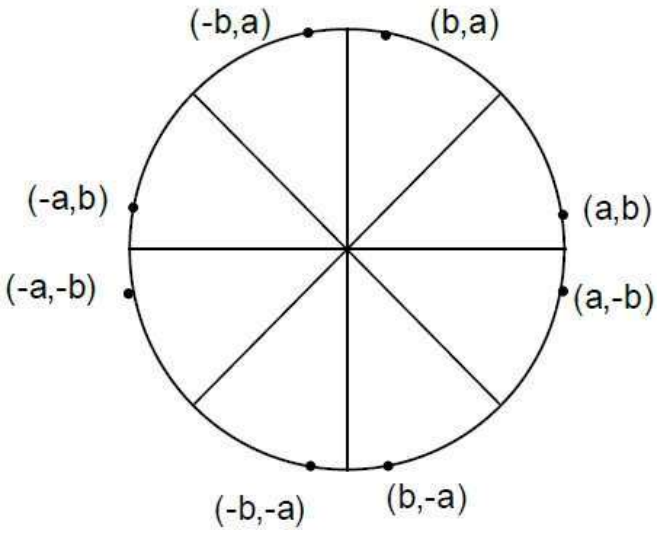


Figure : Quadrant end-points of drawing the circle

```
Draw Circle Method(X, Y, P, Q).
Call Putpixel (X + P, Y + Q).
Call Putpixel (X - P, Y + Q).
Call Putpixel (X + P, Y - Q).
Call Putpixel (X - P, Y - Q).
Call Putpixel (X + Q, Y + X).
Call Putpixel (X - Q, Y + X).
Call Putpixel (X + Q, Y - X).
Call Putpixel (X - Q, Y - X).
```

2.3.2 Mid-Point Circle Generation

Step 1 − Input radius **r** and circle center (x_c,y_c) and obtain the first point on the circumference of the circle centered on the origin as

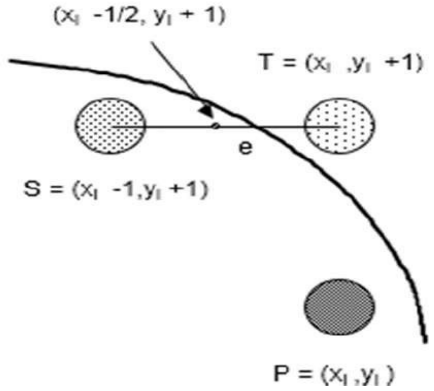
$$(x_0, y_0) = (0, r)$$

Step 2 − Calculate the initial value of decision parameter as

P0P0 = 5/4 – r (See the following description for simplification of this equation.)

$$f(x, y) = x^2 + y^2 - r^2 = 0$$

$$f(x_i - 1/2 + e, y_i + 1)$$
$$= (x_i - 1/2 + e)^2 + (y_i + 1)^2 - r^2$$
$$= (x_i - 1/2)^2 + (y_i + 1)^2 - r^2 + 2(x_i - 1/2)e + e^2$$
$$= f(x_i - 1/2, y_i + 1) + 2(x_i - 1/2)e + e^2 = 0$$



Let $d_i = f(x_i - 1/2, y_i + 1) = -2(x_i - 1/2)e - e^2$

Thus,

If $e < 0$ then $di > 0$ so choose point $S = (x_i - 1, y_i + 1)$.

$d_{i+1} = f(x_i - 1 - 1/2, y_i + 1 + 1) = ((x_i - 1/2) - 1)^2 + ((y_i + 1) + 1)^2 - r^2$

$= d_i - 2(x_i - 1) + 2(y_i + 1) + 1$

$= d_i + 2(y_{i+1} - x_{i+1}) + 1$

If $e >= 0$ then $di <= 0$ so choose point $T = (x_i, y_i + 1)$

$d_{i+1} = f(x_i - 1/2, y_i + 1 + 1)$

$= d_i + 2y_{i+1} + 1$

The initial value of di is

$d_0 = f(r - 1/2, 0 + 1) = (r - 1/2)^2 + 1^2 - r^2$

$= 5/4 - r$ {1-r can be used if r is an integer}

When point $S = (x_i - 1, y_i + 1)$ is chosen then

$d_{i+1} = d_i + -2x_{i+1} + 2y_{i+1} + 1$

When point $T = (x_i, y_i + 1)$ is chosen then

$d_{i+1} = d_i + 2y_{i+1} + 1$

Step 3 − At each X_K position starting at K=0, perform the following test −

If $P_K < 0$ then next point on circle (0,0) is (X_{K+1}, Y_K) and

$P_{K+1} = P_K + 2X_{K+1} + 1$

Else

$P_{K+1} = P_K + 2X_{K+1} + 1 - 2Y_{K+1}$

Where, $2X_{K+1} = 2X_{K+2}$ and $2Y_{K+1} = 2Y_{K-2}$.

Step 4 − Determine the symmetry points in other seven octants.

Step 5 − Move each calculate pixel position (X, Y) onto the circular path centered on (X_C, Y_C) and plot the coordinate values.

$X = X + X_C, \quad Y = Y + Y_C$

Step 6 − Repeat step-3 through 5 until X >= Y.

2.4 Polygon Filling Techniques

Polygon A polygon can be defined as an ordered list of vertices that is formed by line segments that are placed end to end, creating a continuous closed path. Polygons can be divided into three basic types: convex, concave, and complex.

Types of Polygons

Regular - all angles are equal and all sides are the same length. Regular polygons are both equiangular and equilateral.

Equiangular - all angles are equal.

Equilateral - all sides are the same length.

Convex - a straight line drawn through a convex polygon **crosses at most two sides**. Every interior angle is less than 180°. **Convex polygons** are the simplest type of polygon to fill.

Concave - you can draw at least one straight line through a concave polygon that **crosses more than two sides**. At least one interior angle is more than 180°.

Polygon Formulas

(N = # of sides and S = length from center to a corner)

Area of a regular polygon = (1/2) N sin(360°/N) S2

Sum of the interior angles of a polygon = (N - 2) x 180°

The **number of diagonals** in a polygon = 1/2 N *(N-3)

In order to fill a polygon, we do not want to have to determine the type of polygon that we are filling. The easiest way to avoid this situation is to use an algorithm that works for all three types of polygons. Since both convex and concave polygons are subsets of the complex type, using an algorithm that will work for complex polygon filling should be sufficient for all three types.

For filling polygons with particular colors, you need to determine the pixels falling on the border of the polygon and those which fall inside the polygon. There are two basic approaches to filling on raster systems. One way is Line Filling approach which is adopted later in area filling method by drawing straight lines between the edges of polygon called *scan-line polygon filling*. Second way is to start from an interior point and paint outward from this point till we reach the boundary called *boundary-fill*. A slight variation of this technique is used to fill an area specified by cluster (having no specific boundary). The technique is called *flood-fill* and having almost same strategy that is to start from an interior point and start painting outward from this point till the end of cluster. Now having an idea we will try to see each of these one by one, starting from scan-line polygon filling.

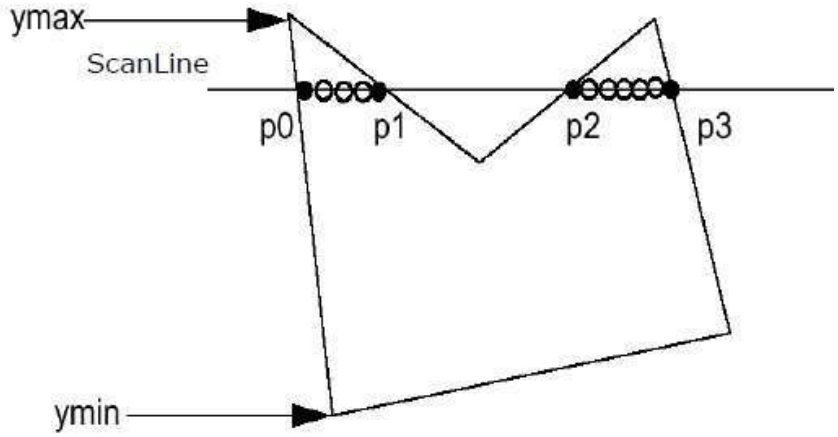
2.4.1 Scan Line Algorithm

What is a scan-line?

A scan-line is a line of constant y value, i.e., y=c, where c lies within our drawing region, e.g., the window on our computer screen.

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 − Find out the Ymin and Ymax from the given polygon.



Step 2 − ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

2.4.2 Boundary Fill Algorithm

Another important class of area-filling algorithms starts at a point known to be inside a figure and starts filling in the figure outward from the point. Using these algorithms a graphic artist may sketch the outline of a figure and then select a color or pattern with which to fill it. The actual filling process begins when a point inside the figure is selected. These routines are like the *paint-scan function* seen in common interactive paint packages.

The boundary-fill method requires the coordinates of a starting point, a fill color, and a boundary color as arguments.

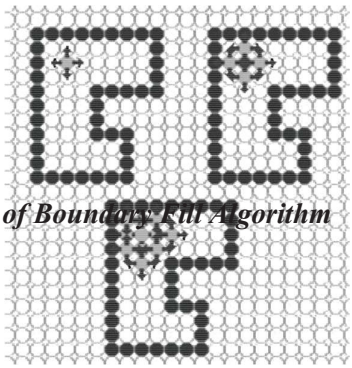
The Boundary fill algorithm performs the following steps:

Check the pixel for boundary color

Check the pixel for fill color

Set the pixel in fill color

Run the process for neighbors



Process of Boundary Fill Algorithm

BOUNDARY FILL PSEUDOCODE:-

```
boundaryFill (x, y, fillColor, boundaryColor)
    if ((x < 0) || (x >= width)) return
        if ((y < 0) || (y >= height)) return

        current = GetPixel(x, y)
        if ((current != boundaryColor) && (current != fillColor))
        setPixel(fillColor, x, y);
        boundaryFill (x+1, y, fillColor, boundaryColor)
        boundaryFill (x, y+1, fillColor, boundaryColor)
        boundaryFill (x-1, y, fillColor, boundaryColor)
        boundaryFill (x, y-1, fillColor, boundaryColor)
```

Note that this is a **recursive routine**. Each invocation of *boundaryFill ()* may call itself four more times.

The logic of this routine is very simple. If we are not either on a boundary or already filled we first fill our point, and then tell our neighbors to fill themselves.

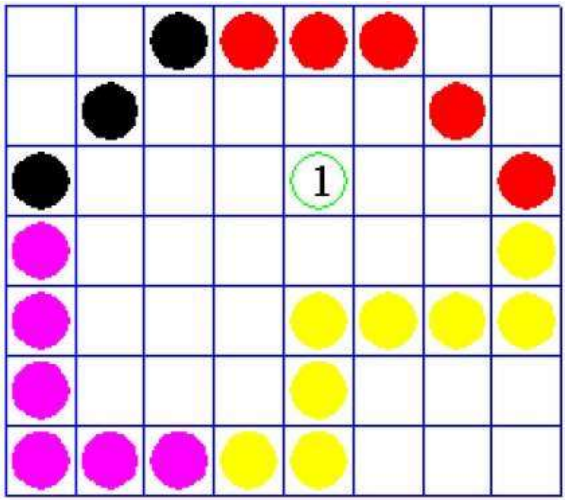
The boundary fill algorithm works as its name. This algorithm picks a point inside an object and starts to fill until it hits the boundary of the object. The color of the boundary and the color that we fill should be different for this algorithm to work.

2.4.3 Flood Fill Algorithm

Sometimes we come across an object where we want to fill the area and its boundary with different colors. We can paint such objects with a specified interior color instead of searching for particular boundary color as in boundary filling algorithm.

Instead of relying on the boundary of the object, it relies on the fill color. In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

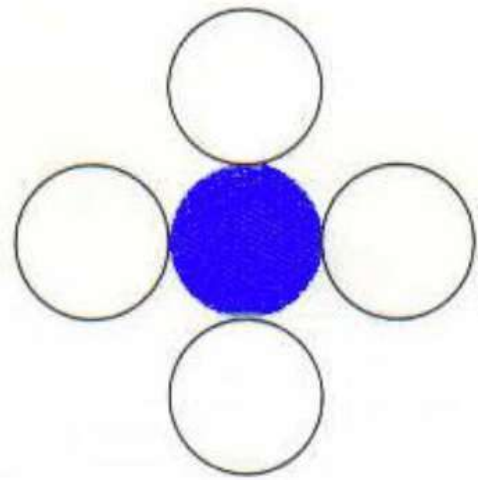
Once again, this algorithm relies on the Four-connect or Eight-connect method of filling in the pixels. But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.



In this algorithm, we assume that color of the boundary is same for the entire object. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-Connected Polygon

In this technique 4-connected pixels are used as shown in the figure. We are putting the pixels above, below, to the right, and to the left side of the current pixels and this process will continue until we find a boundary with different color.



4- Connected Boundary Fill Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color, then repeat the steps 4 and 5 till the boundary pixels reached.

```
If getpixel(x, y) = dcol then repeat step 4 and 5
```

Step 4 − Change the default color with the fill color at the seed point.

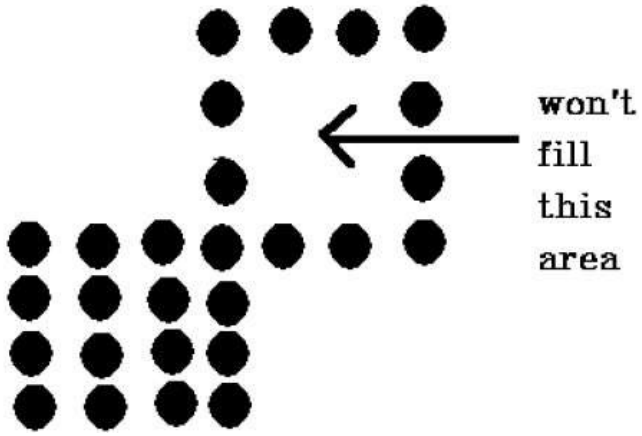
```
setPixel(seedx, seedy, fcol)
```

Step 5 − Recursively follow the procedure with four neighborhood points.

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
```

Step 6 − Exit

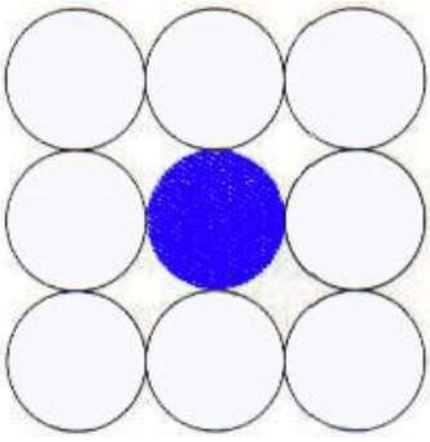
There is a problem with this technique. Consider the case as shown below where we tried to fill the entire region. Here, the image is filled only partially. In such cases, 4-connected pixels technique cannot be used.



8-Connected Polygon

In this technique 8-connected pixels are used as shown in the figure. We are putting pixels above, below, right and left side of the current pixels as we were doing in 4-connected technique.

In addition to this, we are also putting pixels in diagonals so that entire area of the current pixel is covered. This process will continue until we find a boundary with different color.



8- Connected Boundary Fill Algorithm

Step 1 − Initialize the value of seed point (seedx, seedy), fcolor and dcol.

Step 2 − Define the boundary values of the polygon.

Step 3 − Check if the current seed point is of default color then repeat the steps 4 and 5 till the boundary pixels reached

```
If getpixel(x,y) = dcol then repeat step 4 and 5
```

Step 4 − Change the default color with the fill color at the seed point.

```
setPixel(seedx, seedy, fcol)
```

Step 5 − Recursively follow the procedure with four neighbourhood points

```
FloodFill (seedx – 1, seedy, fcol, dcol)
FloodFill (seedx + 1, seedy, fcol, dcol)
FloodFill (seedx, seedy - 1, fcol, dcol)
FloodFill (seedx, seedy + 1, fcol, dcol)
FloodFill (seedx – 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy + 1, fcol, dcol)
FloodFill (seedx + 1, seedy - 1, fcol, dcol)
FloodFill (seedx – 1, seedy - 1, fcol, dcol)
```

Step 6 − Exit

The 4-connected pixel technique failed to fill the area as marked in the following figure which won't happen with the 8-connected technique.

2.5 Clipping Methods

Clipping may be described as the procedure that identifies the portions of a picture lie inside the region, and therefore, should be drawn or, outside the specified region, and hence, not to be drawn. The algorithms that perform the job of clipping are called clipping algorithms there are various types, such as:

• Point Clipping

• Line Clipping

• Polygon Clipping

• Text Clipping

• Curve Clipping

Further, there are a wide variety of algorithms that are designed to perform certain types of clipping operations, some of them which will be discussed in unit.

Line Clipping Algorithms:

• Cohen Sutherland Line Clippings

• Cyrus-Beck Line Clipping Algorithm

Polygon or Area Clipping Algorithm

• Sutherland-Hodgman Algorithm

There are various other algorithms such as, Liang – Barsky Line clipping, Weiler-Atherton Polygon Clipping, that are quite efficient in performing the task of clipping images. But, we will restrict our discussion to the clipping algorithms mentioned earlier.

Before going into the details of point clipping, let us look at some basic terminologies used in the field of clipping, such as, window and viewport.

Window may be described as the world coordinate area selected for display.

Viewport may be described as the area on a display device on which the window is mapped.

So, it is the window that specifies what is to be shown or displayed whereas viewport specifies where it is to be shown or displayed. Specifying these two coordinates, i.e., window and viewport coordinates and then the transformation from window to viewport coordinates is very essential from the point of view of clipping.

Note:

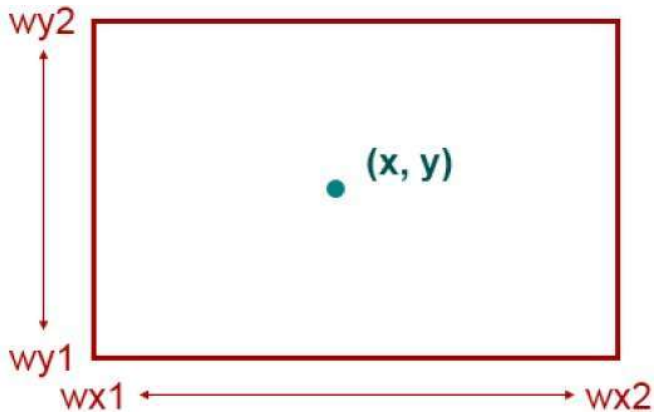
• Assumption: That the window and viewport are rectangular. Then only, by specifying the maximum and the minimum coordinates *i.e.,* (Xwmax, Ywmax) and (Xwmin, Ywmin) we can describe the size of the overall window or viewport.

• Window and viewport are not restricted to only rectangular shapes they could be of any other shape (Convex or Concave or both).

Point clipping tells us whether the given point (X, Y) is within the given window or not; and decides whether we will use the minimum and maximum coordinates of the window.

The X-coordinate of the given point is inside the window, if X lies in between $Wx1 \leq X \leq Wx2$.

The Y coordinate of the given point is inside the window, if Y lies in between $Wy1 \leq Y \leq Wy2$.



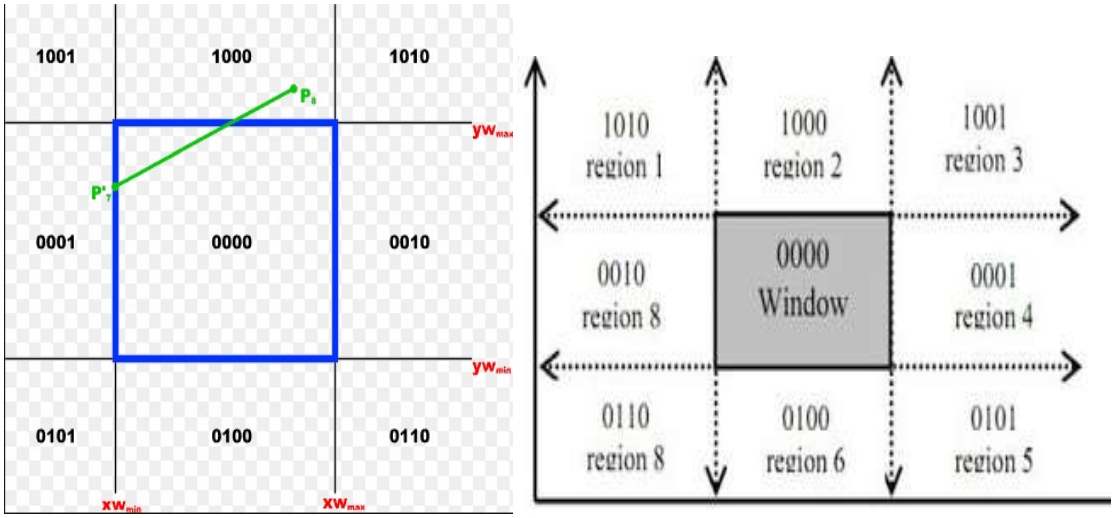
A variety of line clipping algorithms are available in the world of computer graphics, two of which are :

1) Cohen Sutherland algorithm,

2) Cyrus-Beck of algorithm

2.5.1 Cohen Sutherland Line Clipping

The clipping problem is simplified by dividing the area surrounding the window region into four segments Up, Down, Left, Right (U,D,L,R) and assignment of number 1 and 0 to respective segments helps in positioning the region surrounding the window. How this positioning of regions is performed can be well understood by considering the Figure below.



All coding of regions U,D,L,R is done with respect to window region. As window is neither Up nor Down, neither Left nor Right, so, the respective bits UDLR are 0000 of Central region. The positioning code UDLR is 1010, i.e., the region 1 lying on the position which is upper left side of the window. Thus, region 1 has UDLR code 1010 (Up so U=1, not Down so D=0, Left so L=1, not Right so R=0).

The meaning of the UDLR code to specify the location of region with respect to window is:

1st bit ⇒ Up(U) ; 2nd bit ⇒ Down(D) ;3rd bit ⇒ Left(L) ; 4th bit ⇒ Right(R),

Now, to perform Line clipping for various line segment which may reside inside the window region fully or partially, or may not even lie in the widow region; we use the tool of logical ANDing between the UDLR codes of the points lying on the line.

Logical ANDing (^) operation => 1 ^ 1 = 1; 1 ^ 0 = 0;

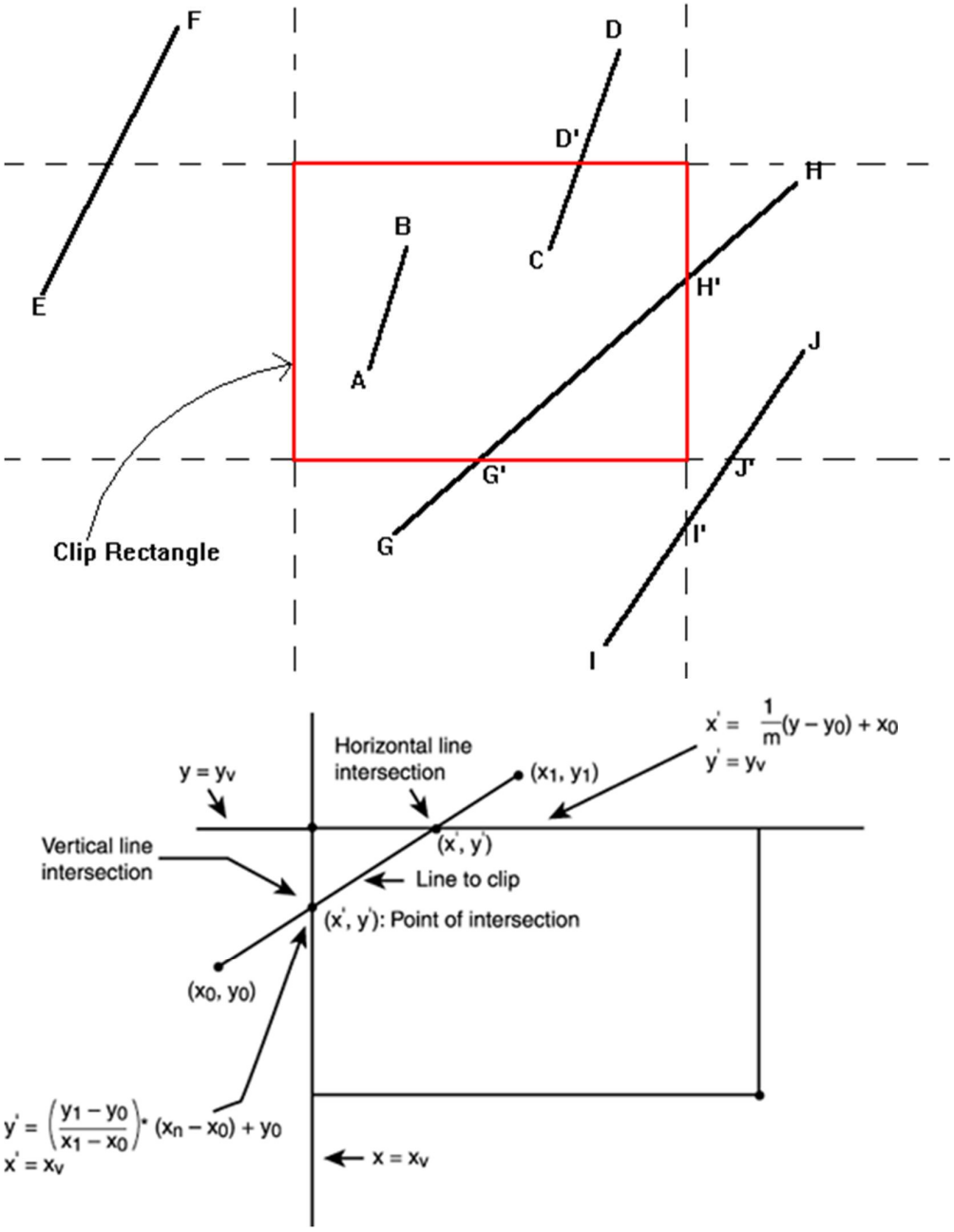
between respective bits implies 0 ^ 1 = 0; 0 ^ 0 = 0

Note:

• UDLR code of window is 0000 always and w.r.t. this will create bit codes of other regions.

• A line segment is visible if both the UDLR codes of the end points of the line segment equal to 0000 i.e. UDLR code of window region. If the resulting code is not 0000 then, that line segment or section of line segment may or may not be visible.

Now, let us study how this clipping algorithm works. For the sake of simplicity we will tackle all the cases with the help of example lines l1 to l5 shown in *Figure 4*.



There are 3 possibilities for the line −

- Line can be completely inside the window (This line should be accepted).

- Line can be completely outside of the window (This line will be completely removed from the region).
- Line can be partially inside the window (We will find intersection point and draw only that portion of line that is inside region).

Algorithm

Step 1 − Assign a region code for each endpoints.

Step 2 − If both endpoints have a region code **0000** then accept this line.

Step 3 − Else, perform the logical **AND** operation for both region codes.

Step 3.1 − If the result is not **0000,** then reject the line.

Step 3.2 − Else you need clipping.

Step 3.2.1 − Choose an endpoint of the line that is outside the window.

Step 3.2.2 − Find the intersection point at the window boundary (base on region code).

Step 3.2.3 − Replace endpoint with the intersection point and update the region code.

Step 3.2.4 − Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.

Step 4 − Repeat step 1 for other lines.

Example : For the rectangular window boundaries given as x_L=2, y_B=2, x_R=8, y_T=8, check the visibility of the following segments using the Cohen-Sutherland algorithm and, if necessary, clip them against the appropriate window boundaries.

Line AB: A(3,10) B(6,12); Line CD: C(4,1), D(10,6)

Solution :-

Step 1. Set up the end codes of the two lines

A(3,10) → (1000); B(6,12) → (1000); → Line AB is invisible

C(4,1) → (0100); D(10,6) → (0010);→ Line CD is indeterminate

Step 2. Clipping of line CD

(a) Endpoint C has a code of (0100). Since bit 2 is not zero, intersection must be found with the boundary $y=y_0=2$. The parametric equation of line CD is

X= 4 +(10-4)t = 4 +6t

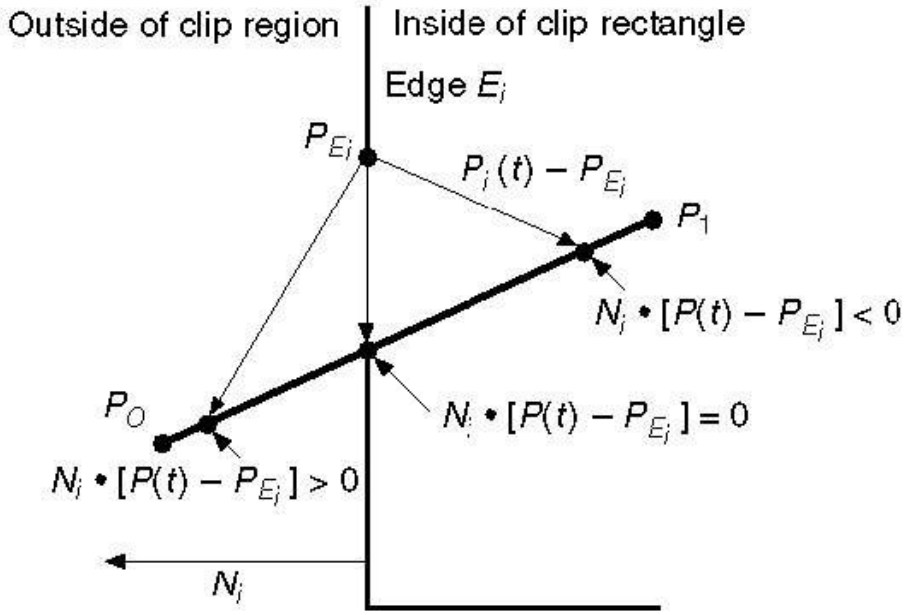
Y = 1+(6-1)t= 1+5t *Solving → t=0.2 so intersection point is* ***I1=(5.2,2)***

(b) Endpoint D has a code of (0010). For bit 3 not equal to zero, the intersection with the boundary x=xR=8 must be found. Substituting x=8 to previous equation yields t=4/6=0.667 and y=4.33. So the intersection point is ***I2(8,4.33)***

Since both I1 and I2 lie on the window boundary, their end codes are (0000) and (0000), respectively. The line segment is, therefore, visible between the two intersection points.

2.5.2 Cyrus-Beck Line Clipping Algorithm

The Cyrus-Beck algorithm is of O(N) complexity, and it is primarily intended for a clipping a line in the parametric form against a convex polygon in 2 dimensions.It was designed to be more efficient than the Sutherland Cohen algorithm which uses repetitive clipping. It was introduced back in 1978 by Cyrus and Beck and it employs parametric line representation and simple dot products.



Parametric equation of line is −

$P_0P_1:P(t) = P_0 + t(P_1 - P_0)$

Let N_i be the outward normal edge E_i. Now pick any arbitrary point P_{Ei} on edge E_i then the dot product $N_i.[P(t) – P_{Ei}]$ determines whether the point P(t) is “inside the clip edge” or “outside” the clip edge or “on” the clip edge.

The point P(t) is inside if $N_i.[P(t) – P_{Ei}] < 0$

The point P(t) is outside if $N_i.[P(t) – P_{Ei}] > 0$

The point P(t) is on the edge if $N_i.[P(t) – P_{Ei}] = 0$ (Intersection point)

$N_i.[P(t) – P_{Ei}] = 0$

$N_i.[P_0 + t(P_1 - P_0) – P_{Ei}] = 0$ (Replacing P(t) with $P_0 + t(P_1 - P_0)$)

$N_i.[P_0 – P_{Ei}] + N_i.t[P_1 - P_0] = 0$

$N_i.[P_0 – P_{Ei}] + N_i \cdot tD = 0$ (substituting D for $[P_1 - P_0]$)

$N_i.[P_0 – P_{Ei}] = - N_i \cdot tD$

The equation for t becomes,

$$t=Ni.[Po−PEi]−Ni.Dt=Ni.[Po−PEi]−Ni.D$$

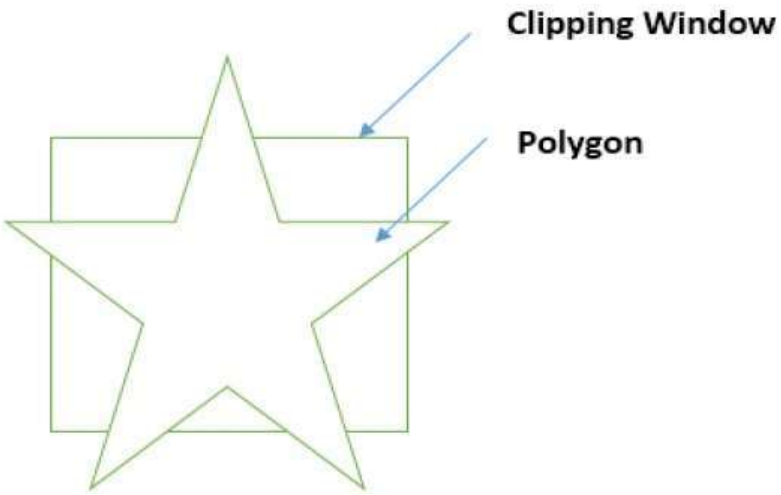
It is valid for the following conditions −

- $N_i \neq 0$ (error cannot happen)
- $D \neq 0$ ($P_1 \neq P_0$)
- $N_i \cdot D \neq 0$ (P_0P_1 not parallel to E_i)

2.5.3 Polygon Clipping (Sutherland Hodgman Algorithm)

A polygon can also be clipped by specifying the clipping window. Sutherland Hodgeman polygon clipping algorithm is used for polygon clipping. In this algorithm, all the vertices of the polygon are clipped against each edge of the clipping window.

First the polygon is clipped against the left edge of the polygon window to get new vertices of the polygon. These new vertices are used to clip the polygon against right edge, top edge, bottom edge, of the clipping window as shown in the following figure.



While processing an edge of a polygon with clipping window, an intersection point is found if edge is not completely inside clipping window and the a partial edge from the intersection point to the outside edge is clipped. The following figures show left, right, top and bottom edge clippings −

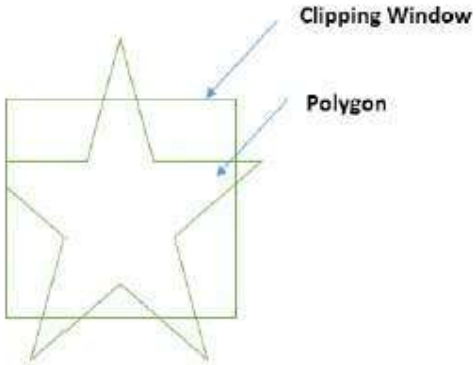


Figure: Clipping Left Edge

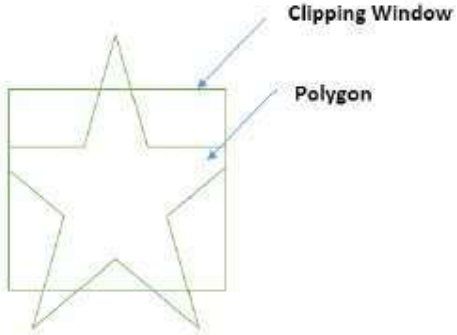


Figure: Clipping Right Edge

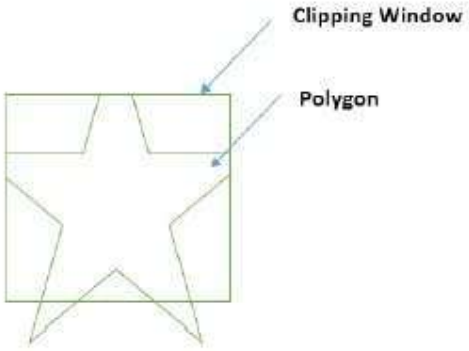


Figure: Clipping Top Edge

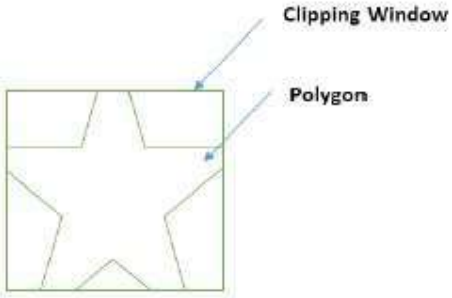


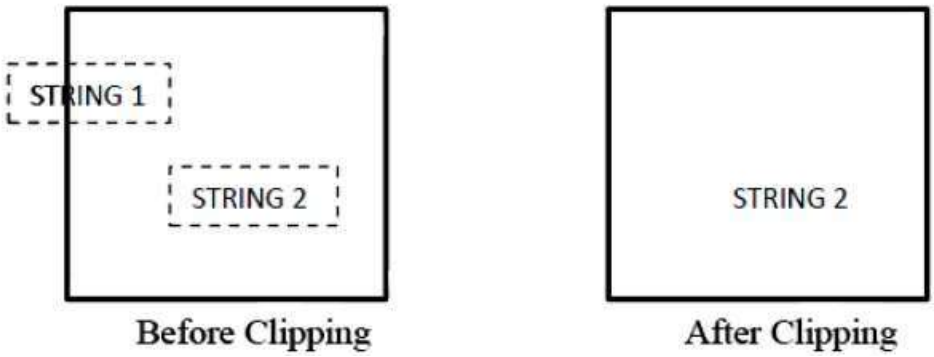
Figure: Clipping Bottom Edge

2.5.4 Text Clipping

Various techniques are used to provide text clipping in a computer graphics. It depends on the methods used to generate characters and the requirements of a particular application. There are three methods for text clipping which are listed below −

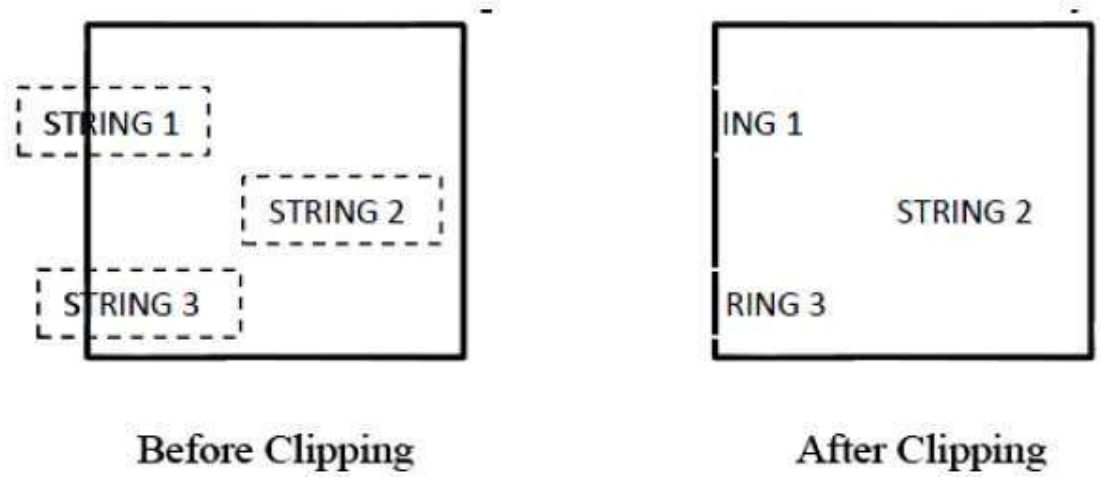
- All or none string clipping
- All or none character clipping
- Text clipping

The following figure shows all or none string clipping −



In all or none string clipping method, either we keep the entire string or we reject entire string based on the clipping window. As shown in the above figure, STRING2 is entirely inside the clipping window so we keep it and STRING1 being only partially inside the window, we reject.

The following figure shows all or none character clipping −



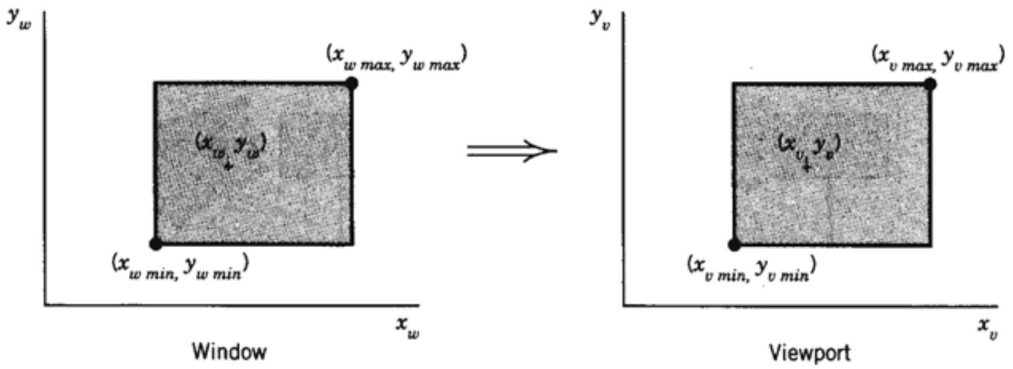
This clipping method is based on characters rather than entire string. In this method if the string is entirely inside the clipping window, then we keep it. If it is partially outside the window, then

- You reject only the portion of the string being outside
- If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.

2.6 Windowport-to-Viewport mapping

In a typical application, we have a rectangle made of pixels, with its natural pixel coordinates, where an image will be displayed. This rectangle will be called the **viewport**. We also have a set of geometric objects that are defined in a possibly different coordinate system, generally one that uses real-number coordinates rather than integers. These objects make up the "scene" or "world" that we want to view, and the coordinates that we use to define the scene are called **world coordinates**.

For 2D graphics, the world lies in a plane. It's not possible to show a picture of the entire infinite plane. We need to pick some rectangular area in the plane to display in the image. Let's call that rectangular area the **window**, or view window. A coordinate transform is used to map the window to the viewport.



A window-to-viewport mapping can be expressed by the following relationships, based on elements shown

$$\frac{y_v - y_{v\min}}{y_{v\max} - y_{v\min}} = \frac{y_w - y_{w\min}}{y_{w\max} - y_{w\min}}$$

$$x_v = (x_w - x_{w\min})\left(\frac{x_{v\max} - x_{v\min}}{x_{w\max} - x_{w\min}}\right) + x_{v\min}$$

$$y_v = (y_w - y_{w\min})\left(\frac{y_{v\max} - y_{v\min}}{y_{w\max} - y_{w\min}}\right) + y_{v\min}$$

Any window can be specified by any four co-ordinates i.e. $W_x$min, $W_x$max, $W_y$min and $W_y$max and similarly a viewport can be represented by four normalized co-ordinates i.e. $V_x$min, $V_x$max, $V_y$min and $V_y$max viewing transformation of any point let W(W_x, W_y), can be implemented using following equation. If normalized co-ordinates are V(V_x, V_y)

then $$\frac{W_x - W_{xmin}}{W_{xmax} - W_{xmin}} = \frac{V_x - V_{xmin}}{V_{xmax} - V_{xmin}}$$

$$and \quad \frac{W_y - W_{ymin}}{W_{ymax} - W_{ymin}} = \frac{V_y - V_{ymin}}{V_{ymax} - V_{ymin}}$$

$$or, \quad V_x = \frac{V_{xmax} - V_{xmin}}{W_{xmax} - W_{xmin}}(W_x - W_{xmin}) + V_{xmin}$$

$$V_y = \frac{V_{ymax} - V_{ymin}}{W_{ymax} - W_{ymin}}(W_y - W_{ymin}) + V_{ymin}$$

The viewing transformation can be represented in terms of basic transformation as above figure. If the composite transformation is N then

$$\begin{bmatrix} V_x \\ V_y \\ 1 \end{bmatrix} = N \cdot \begin{bmatrix} W_x \\ W_y \\ 1 \end{bmatrix}$$

Where $$N = T_v \cdot S_{sx}, sy \cdot T_{v^1}$$

$$(Since \ V = i\,V_{xmin} + jV_{ymin})$$

$$V' = i\,W_{xmin} + jW_{ymin}$$

$$T_v = \begin{bmatrix} 1 & 0 & V_{xmin} \\ 1 & 0 & V_{ymin} \\ 0 & 0 & 1 \end{bmatrix}, \quad T_{-v'} \begin{bmatrix} 1 & 0 & W_{xmin} \\ 1 & 0 & W_{ymin} \\ 0 & 0 & 1 \end{bmatrix}$$

and

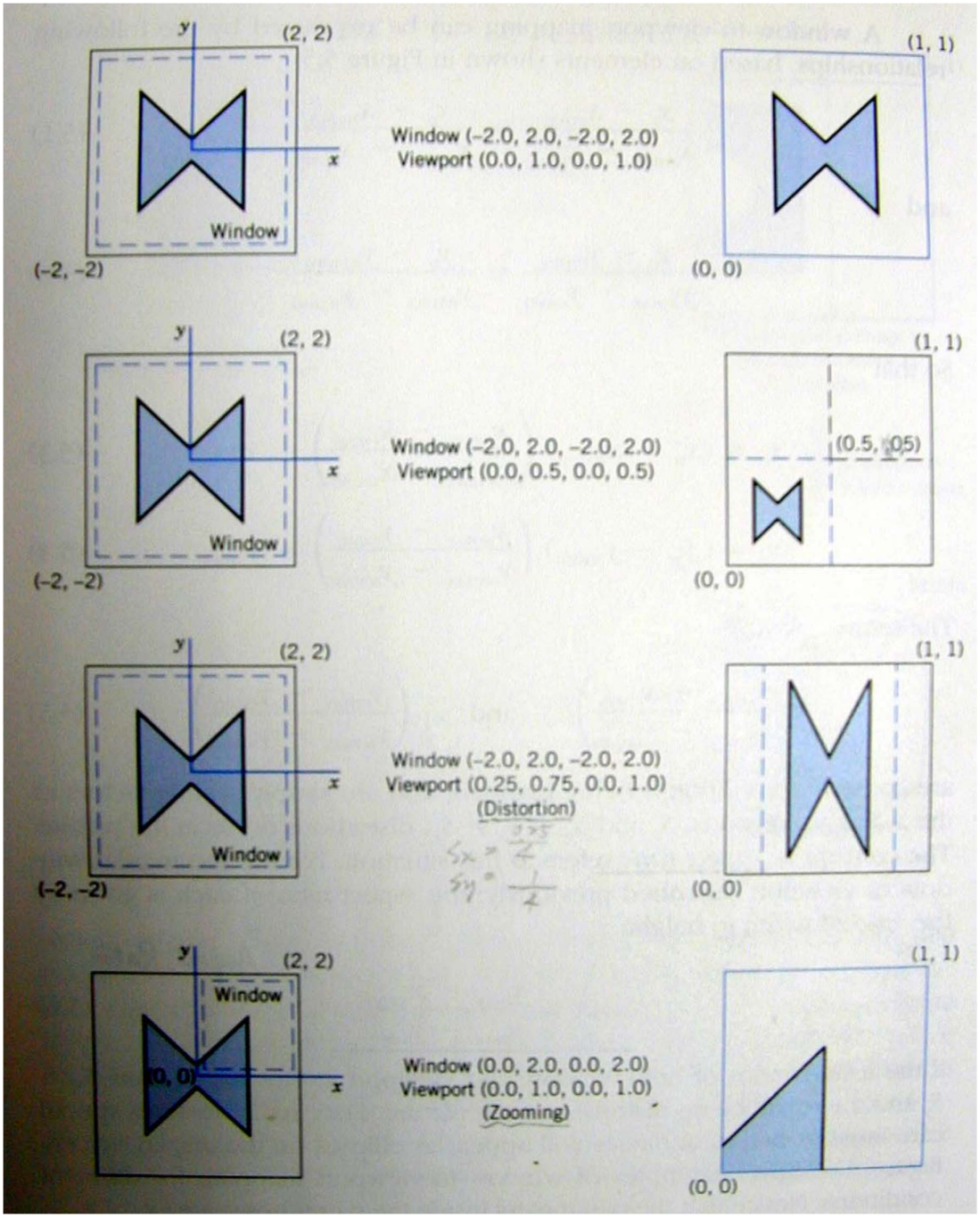
$$\begin{bmatrix} S_x = \frac{V_{xmax} - V_{xmin}}{W_{xmax} - W_{xmin}} & 0 & 0 \\ 0 & S_y = \frac{V_{ymax} - V_{ymin}}{W_{ymax} - W_{ymin}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

For rectangular window or viewport, the ***aspect ratio*** is given by the ratio of width to height

$$AR = \frac{x_{max} - x_{min}}{y_{max} - y_{min}}$$

Example :- Find the transformation matrix that will map points contained in a window whose lower left corner is at (2, 2) and upper right corner is at (6, 5) onto a normalized viewport that has a lower left corner at (1/2, 1/2) and upper right corner at (1, 1).

Solution :-

= 

The Shear Transformation in each case :-

Sx =1/4 → 0.5/4 → 0.5/4 → 1/2

Sy =1/4 → 0.5/4 →1/ 4 → ½

So that the distortion picture (sx=1/8, sy=1/4) in the third row is can be rewritten in matrix form,

$$[x_v, y_v, 1] = [x_w, y_q, 1]\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ (-s_x x_{w\min} + x_{v\min}) & (-s_y y_{w\min} + y_{v\min}) & 1 \end{bmatrix}$$

the transformation matrix becomes:

$$M_{map} = \begin{bmatrix} \frac{1}{8} & 0 & 0 \\ 0 & \frac{1}{6} & 0 \\ \frac{1}{4} & \frac{1}{6} & 1 \end{bmatrix}$$

2.7 SUMMARY

In this chapter, we understood the techniques of rendering primitives like straight line and circles. The filling algorithms are quite important as they give privilege to quickly fill colours into the graphics created by you. We have also covered the viewing areas and conditions for clipping by various algorithms. We understood that the world is viewed through a world coordinate window, which is mapped onto a device coordinate viewport. The unit is quite important from the point of view of achieving realism through computer graphics. This unit contains algorithms, which are easy to implement in any programming language.

2.8 QUESTIONS FOR EXERCISE

1) Compare Bresenham line generation with DDA line generation.

2) Illustrate the Bresenham line generation algorithm by digitising the line with end points (15, 5) and (25,13).

3) Given a circle radius $r = 5$ determine positions along the circle octants in 1st Quadrant from $x = 0$ to $x = y$?

4) Distinguish between Scan line polygon fill and Seed fill or Flood fill algorithm?

5) A clipping window is given by P0(10,10), P1(20,10), P2(10,20), P3(20,20). Using Cyrus Beck algorithm clip the line A(0,0) and B(15,25)

6) Compare Cohen Sutherland Line clipping with Cyrus Beck Line clipping algorithm?

7) For a window A(100, 10), B(160, 10, C(160, 40), D(100, 40). Using Sutherland-Cohen clipping algorithm find the visible portion of the line segments EF, GH and P1P2. E(50,0), F(70,80), G(120, 20), H(140, 80), P1(120, 5), P2(180, 30).

2.9 SUGGESTED READINGS

- Computer Graphics: Principles and Practice, 4th Edition
- Computer Graphics, C Version, 2nd Edition Fundamentals of Computer Graphics, 3rd Edition
- Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6th ed.
- 3D Computer Graphics: A Mathematical Introduction with OpenG

UNIT -3

GEOMETRICAL TRANSFORMATIONS

3.0 OBJECTIVE

The objects are referenced by their coordinates. Changes in orientation, size and shape are accomplished with geometric transformations that allow us to calculate the new coordinates. After going through this unit, you should be able to:

• Describe the basic transformations for 2-D translation, rotation, scaling and shearing;

• Discuss the role of composite transformations

• Describe composite transformations for Rotation about a point and reflection about a line;

• Define and explain the use of homogeneous coordinate systems for the transformations

• Describe the 3-D transformations of translation, rotation, scaling and shearing;

3.1 TWO DIMENSIONAL TRANSFORMATIONS

Geometric transformations have numerous applications in geometric modeling, e.g., manipulation of size, shape, and location of an object. Transformations are also used to generate surfaces and solids by sweeping curves and surfaces, respectively. The term 'sweeping' refers to parametric transformations, which are utilized to generate surfaces and solids. When we sweep a curve, it is transformed through several positions along or around an axis, generating a surface.

Transformation is the backbone of computer graphics, enabling us to manipulate the shape, size, and location of the object. It can be used to effect the following changes in a geometric object:

• Change the location • Change the Shape • Change the size • Rotate • Copy

• Generate a surface from a line • Generate a solid from a surface • Animate the object

The appearance of the generated surface depends on the number of instances of the transformation. Basically there are two categories of transformation:-

i. Geometric Transformation

ii. Co-ordinate Transformation

i. **Geometric Transformation**: - Every object is assumed to be a set of points. Every object point P is denoted by the ordered pairs known as coordinates (x, y) and so the object is the sum of total of all the coordinates points. If any object is transform to a new position then the coordinates of new position can be obtained by the application of Geometric Transformation.

ii. **Co-ordinate Transformation**:- When object itself relative to a stationary co-ordinate system or background, referred as geometric transformations and applied to each point of the object. And while the co-ordinate system is moved relative to the object and object is held stationary then this process is termed as co-ordinate transformation.

Let us look at the procedure for carrying out basic transformations, which are based on matrix operation. A transformation can be expressed as

$$[P^*] = [P]\,[T]$$

where, [P*] is the new coordinates matrix

[P] is the original coordinates matrix, or points matrix

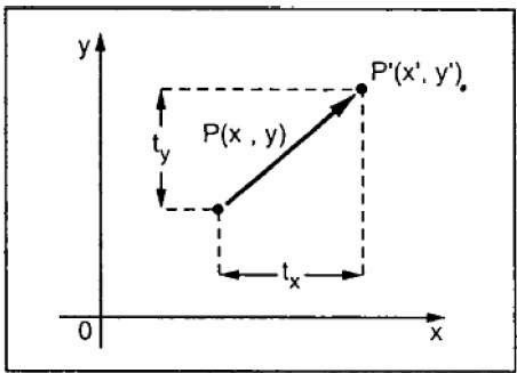
[T] is the transformation matrix

With the z-terms set to zero, the P matrix can be written as, [P] = $\begin{matrix} x1 & y1 & 0 \\ x2 & y2 & 0 \\ xn & yn & 0 \end{matrix}$

Values of the elements in the matrix will change according to the type of transformation being used, as we will see shortly. The transformation matrix changes the size, position, and orientation of an object, by mathematically adding, or multiplying its coordinate values.

3.1.1 Translation

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate (X, Y) to get the new coordinate (X', Y').



From the above figure, you can write that −

$$X' = X + tx$$

$$Y' = Y + ty$$

In the matrix form

$$T_V = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

The pair (tx, ty) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

$$P=[X][Y]P=[X][Y]\ P' = [X'][Y'] \text{ and } T = [tx][ty]$$

We can write it as −

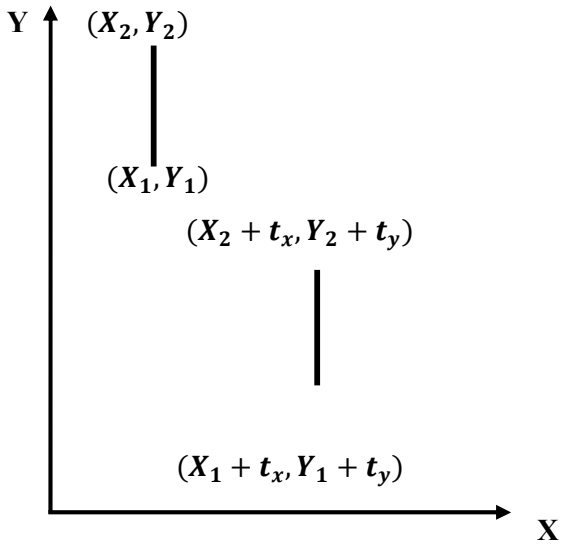
$$\mathbf{P' = P + T}$$

If (x,y) is the original point and (x1,y1) is the transformed point, then the formula for a translation is

x1 = x + e

y1 = y + f

where e is the number of units by which the point is moved horizontally and f is the amount by which it is moved vertically.



Fig

Translation of a line in X and Y-direction

Example1: Translate a square ABCD with the coordinates A(0,0),B(5,0),C(5,5),D(0,5) by 2 units in x-direction and 3 units in y-direction.

Hint : We can represent the given square, in matrix form, using homogeneous coordinates of vertices as :-

$$\begin{matrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{matrix} = \begin{matrix} 0 & 0 & 1 \\ 5 & 0 & 1 \\ 5 & 5 & 1 \end{matrix}$$

The translation factors are, tx=2, ty=3

The transformation matrix for translation :

$$Tx = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tx & ty & 1 \end{matrix} = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{matrix}$$

New object point coordinates are:

[A'B'C'D'] = [ABCD].Tx

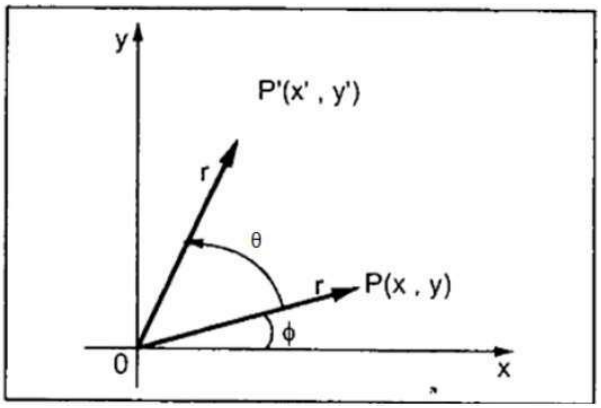
$$\begin{matrix} A' \\ B' \\ C' \\ D' \end{matrix} \begin{pmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 \\ x'_4 & y'_4 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 5 & 0 & 1 \\ 5 & 5 & 1 \\ 0 & 5 & 1 \end{pmatrix} \cdot \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 3 & 1 \\ 7 & 3 & 1 \\ 7 & 8 & 1 \\ 2 & 8 & 1 \end{pmatrix}$$

3.1.2 Rotation

In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle φ from the horizontal X coordinate with distance r from the origin.

Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P' (X', Y').



Using standard trigonometric the original coordinate of point P(X, Y) can be represented as −

$X=r\cos\phi$.....(1)

$Y=r\sin\phi$.....(2)

Same way we can represent the point P' (X', Y') as −

$x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta$.....(3)

$y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta$.....(4)

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

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

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

Representing the above equation in matrix form for **Anti-clockwise** direction,

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

$$[x' \quad y'] = [x \quad y]\begin{bmatrix} cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \qquad \text{OR} \quad P' = P \cdot R$$

Where R is the rotation matrix

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, that is for **Clockwise** rotation we have to put $\theta = -\theta$, thus the rotation matrix R_θ

$$R_{-\theta} = \begin{pmatrix} \cos(-\theta) & \sin(-\theta) & 0 \\ -\sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Example 2: Perform a 45^0 rotation of a triangle A(0,0),B(1,1),C(5,2) about the origin.

Solution: We can represent the given triangle, in matrix form, using homogeneous coordinates of the vertices:

$$\begin{matrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{matrix}$$

The matrix of rotation is: $R_\theta = R_{45}^0 = \begin{matrix} \cos 45 & \sin 45 & 0 \\ -\sin 45 & \cos 45 & 0 \\ 0 & 0 & 1 \end{matrix} = \begin{matrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 1 \end{matrix}$

So the new coordinates A'B'C' of the rotated triangle ABC can be found as:

$$[A'B'C'] = [ABC].\ R_{45^\circ} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 & 0 \\ -\sqrt{2}/2 & \sqrt{2}/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & \sqrt{2} & 1 \\ 3\sqrt{2}/2 & 7\sqrt{2}/2 & 1 \end{bmatrix}$$

3.1.3 Scaling

In scaling transformation, the original coordinates of an object are multiplied by the given scale factor. There are two types of scaling transformations: uniform and non-uniform. In the uniform scaling, the coordinate values change uniformly along the x, y, and z coordinates, where as, in non-uniform scaling, the change is not necessarily the same in all the coordinate directions.

The mathematical expression for pure scaling is

$$x' = S_x \,.\, x$$

$$y' = S_y \,.\, y$$

Matrix equation of a non-uniform scaling has the form:

$$\Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

where, s_x, s_y are the scale factors for the x and y coordinates of the object.

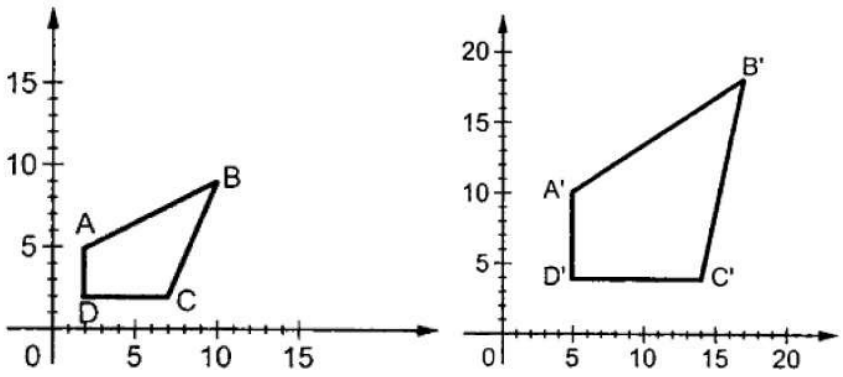
That is, if a point (x,y) is scaled by a factor of a in the x direction and by a factor of d in the y direction, then the resulting point $(x1,y1)$ is given by

x1 = a * x

y1 = d * y

If you apply this transform to a shape that is centered at the origin, it will stretch the shape by a factor of *a* horizontally and *d* vertically.

Where S is the scaling matrix. The scaling process is shown in the following figure.



If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object.

If we provide values greater than 1, then we can increase the size of the object.

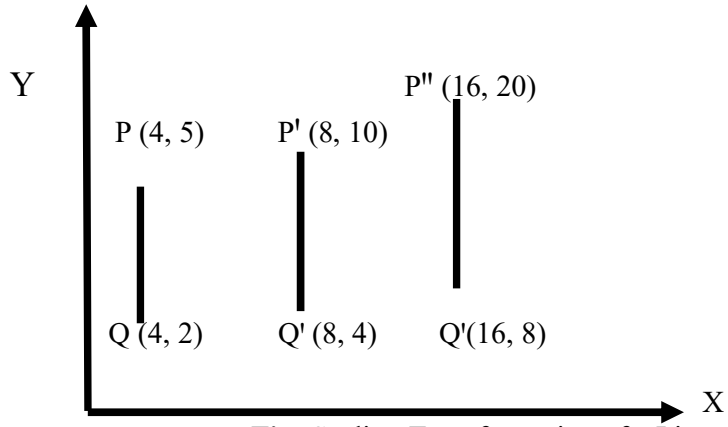


Fig: Scaling Transformation of a Line

Line PQ be scaled to P'Q' when S_x=S_y=0.5 and to P"Q" when S_x=S_y=2
Obviously

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 8 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

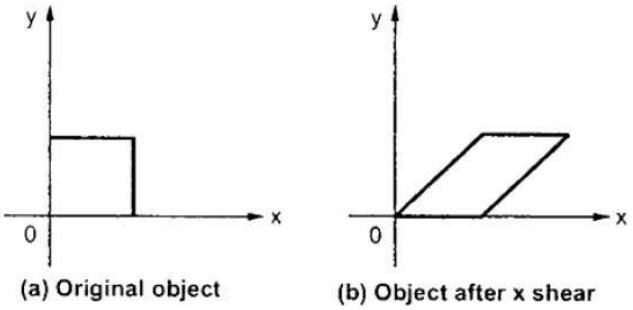
Thus $Q' = (4, 2)$
Similarly $P' = (4, 5)$
when $S_x = S_y = 0.5$

3.1.4 Shearing

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



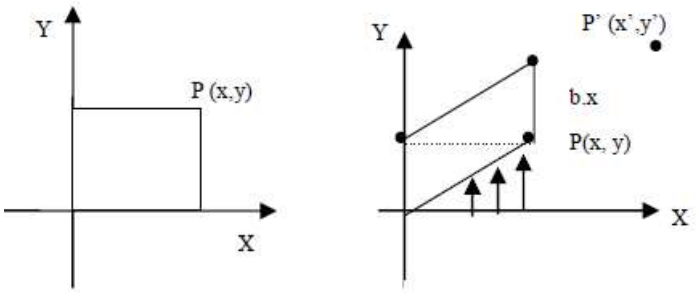
The transformation matrix for X-Shear about the origin by ***a*** distance can be represented as –

$$\begin{matrix} 1 & 0 & 0 \\ a & 1 & 1 \\ 0 & 0 & 1 \end{matrix}$$

$X' = X + Sh_x \cdot Y$ $\qquad\qquad$ $Y' = Y$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



The Y-Shear can be represented in matrix from as –

$$\begin{matrix} 1 & b & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{matrix}$$

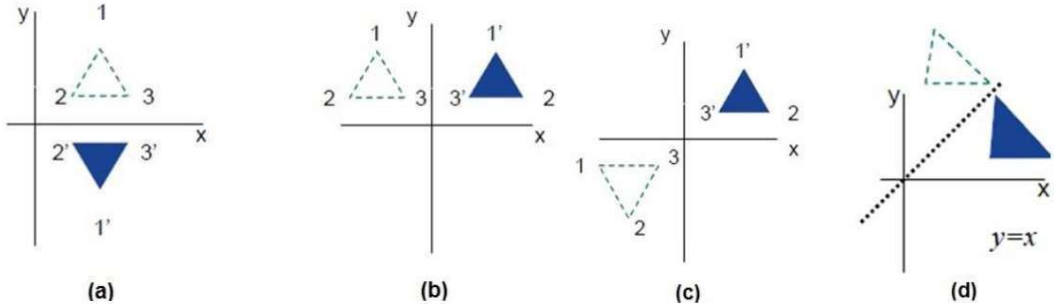
$Y' = Y + Sh_y \cdot X$ $\qquad\qquad$ $X' = X$

Although shears can in fact be built up out of rotations and scalings if necessary, it is not really obvious how to do so. A shear will "tilt" objects. A horizontal shear will tilt things towards the left (for negative shear) or right (for positive shear). A vertical shear tilts them up or down.

3.1.5 Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.



Therefore the reflection between the point P(x, y) and its image $P'(x, y')$

about x-axis is $x' = x, y' = -y$. Obviously the transformation matrix for the reflection about x-axis is given by

$$[T_m] = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

And the transformation is represented as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

i.e. $[x'] = [T_m]_{y=0} [x]$

A reflection about y-axis flips x coordinates while y coordinates remains the same.

Reflection about the Straight line Y= -X

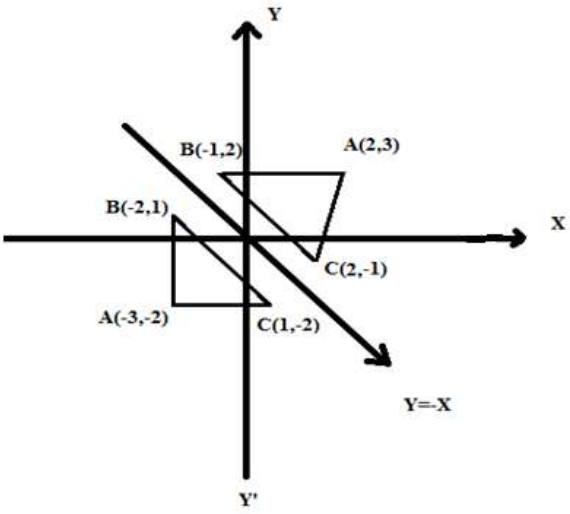


Fig-3.21

If $x' = -y$ $y' = -x$

This can be represented by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Where $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ is the transformation matrix $[T_M] = -X$

Thus we can infer that unlike in the case of reflection about diagonal axis y=x, in reflections about the other diagonal y=-x, the co-ordinate values are interchanged with their signs reversed.The changes of the vertices of triangle ABC to $A'B'C'$ are

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & -1 & 2 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -3 & -2 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

3.2 Composite Transformation

We have seen the basic matrix transformations for translation, rotation, reflection, scaling and shearing with respect to the origin of the coordinate system. By multiplying these basic matrix transformations, we can build complex transformations, such as rotation about an arbitrary point, mirror reflection about a line etc. This process is called concatenation of matrices and the resulting matrix is often referred to as the composite transformation matrix. Inverse transformations play an important role when you are dealing with composite transformation. They come to the rescue

of basic transformations by making them applicable during the construction of composite transformation. You can observed that the Inverse transformations for translation, rotation, reflection, scaling and shearing have the following relations, and v, θ, a, b, sx, sy, sz are all parameter involved in the transformations. If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix −

$$[T][X] = [X]\ [T1]\ [T2]\ [T3]\ [T4]\ \ldots.\ [Tn]$$

Where [Ti] is any combination of

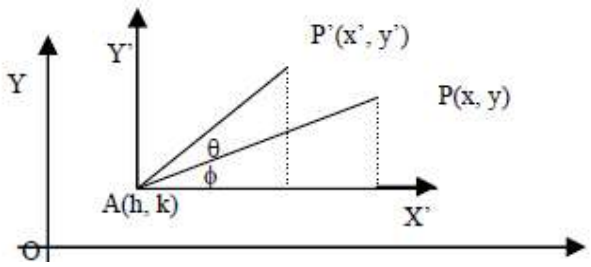
- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps −

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.
- Finally, translate the center of rotation back where it belonged.

Example : Given a 2-D point P(x,y), which we want to rotate, with respect to an arbitrary point A(h,k).Let P'(x'y') be the result of anticlockwise rotation of point P by angle θ about A



Since, the rotation matrix Rθ is defined only with respect to the origin, we need a set of basic transformations, which constitutes the composite transformation to compute the rotation about a given arbitrary point A, denoted by Rθ,A. We can determine the transformation Rθ,A in three steps:

1) Translate the point A(h,k) to the origin O, so that the center of rotation A is at the origin.

2) Perform the required rotation of θ degrees about the origin, and

3) Translate the origin back to the original position A(h,k).

$$R_{\theta,A} = T_{-v} \cdot R_{\theta} \cdot T_{v}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -h & -k & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ h & k & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ (1-\cos\theta).h+k.\sin\theta & (1-\cos\theta).k-h.\sin\theta & 1 \end{pmatrix}$$

Example: Perform a 45^0 rotation of a triangle A (0,0), B (1,1), C (5,2) about an arbitrary point P(–1, –1).

Solution: Given triangle ABC, as show in Figure (a), can be represented in homogeneous coordinates of vertices as:

$$[A\ B\ C] = \begin{matrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{matrix}$$

A rotation matrix R_Q, about a given arbitrary point A (h, k) is:

$$\begin{matrix} \cos\theta & \sin\theta & 1 \\ -\sin\theta & \cos\theta & 1 \\ (1-\cos\theta).j+k.\sin\theta & (1-\cos\theta).k-j.\sin\theta & 1 \end{matrix}$$

Thus, $R_{45} = \begin{matrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -1 & \sqrt{2}-1 & 1 \end{matrix}$

So the new coordinates [A′B′C′] of the rotated triangle [ABC] can be found as:

$$[A'B'C'] = [ABC] \cdot R_{45} = \begin{matrix} 0 & 0 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 1 \end{matrix} \cdot \begin{matrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ -1 & \sqrt{2}-1 & 1 \end{matrix} = \begin{matrix} -1 & \sqrt{2}-1 & 1 \\ -1 & 2\sqrt{2}-1 & 1 \\ 3/2.\sqrt{2}-1 & 9/2.\sqrt{2}-1 & 1 \end{matrix}$$

3.3 Homogenous Coordinate Systems

Let P(x,y) be any point in 2-D Euclidean (Cartesian) system. In Homogeneous Coordinate system, we add a third coordinate to a point. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; with the condition that

(x1,y1,H1)=(x2,y2,H2) ↔ x1/H1 = x2/H2 ; y1/H1 = y2/H2.

If we multiply $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ with a non zero scalar 'h' Then the matrix it forms is $\begin{bmatrix} xh \\ yh \\ h \end{bmatrix}$

In 2D Plane. $\begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x_h \\ y_h \\ h \end{bmatrix} = h \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

The extra coordinate h is known as weight, which is homogeneously applied to the Cartesian components.

(Here, if we take H=0, then we have point at infinity, i.e., generation of horizons).

Thus all geometric transformation equations can be represented uniformly as matrix multiplication. Coordinates are represented with three element column, vectors and transformation operations are written in form of 3 by 3 matrices. For translation transformation (x,y) --> (x+tx,y+ty) in Euclidian system, where tx and ty are the translation factor in x and y direction, respectively. Unfortunately, this way of describing translation does not use a matrix, so it cannot be combined with other transformations by simple matrix multiplication.

Thus, for translation, we now have,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \text{ OR } (x',y',1) = (x,y,1) \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ tx & ty & 1 \end{matrix}$$

or, Symbolically $[X'] = [T_T][X]$

Equations of Relation and Scaling With respect to coordinate origin may be modified as

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

and

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$ respectively

Similarly the modified General expression for reflection may be

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Such a combination would be desirable; for example, we have seen that rotation about an arbitrary point can be done by a translation, a rotation, and another translation. We would like to be able to combine these three transformations into a single transformation for the sake of efficiency and elegance. One way of doing this is to use homogeneous coordinates. In homogeneous coordinates we use 3x3 matrices instead of 2x2, introducing an additional dummy coordinate H. Instead of (x,y), each point is represented by a triple (x,y,H) such that H≠0; In two dimensions the value of H is usually kept at 1 for simplicity.

The **advantage** of introducing the matrix form of translation is that it simplifies the operations on complex objects, i.e., we can now build complex transformations by multiplying the basic matrix transformations.

In other words, we can say, that a sequence of transformation matrices can be concatenated into a single matrix. This is an effective procedure as it reduces the computation because instead of applying initial coordinate position of an object to each transformation matrix, we can obtain the final transformed position of an object by applying composite matrix to the initial coordinate position of an object. Matrix representation is standard method of implementing transformations in computer graphics.

Thus, from the point of view of matrix multiplication, with the matrix of translation, the other basic transformations such as scaling, rotation, reflection, etc., can also be expressed as 3x3 homogeneous coordinate matrices. This can be accomplished by augmenting the 2x2 matrices

with a third row (0,0,x) and a third column. That is to perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process −

- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W.

In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point P(X, Y) can be converted to homogenous coordinates by P' (X_h, Y_h, h).

3.4 Three Dimensional Transformations

Transformations in 3 dimensional space are analogous to those in two dimensional space in many ways. Transformation matrix is the basic tool for 3-D transformation as was for 2-D. A matrix with n x m dimensions is multiplied with the coordinate of objects. Usually 3 x 3 or 4 x 4 matrices are used for transformation

3.4.1 Transformations for 3-D Translation

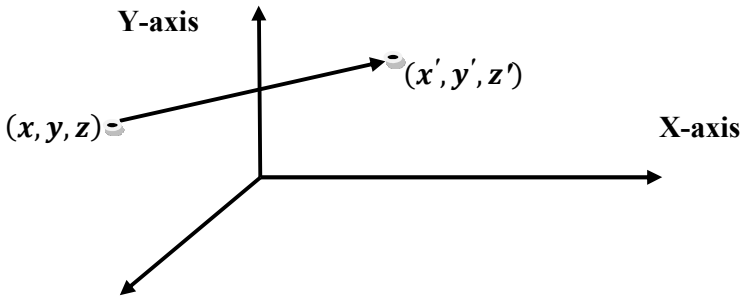
In a three dimensional homogeneous co-ordinate representation a point is translated from position $\boldsymbol{P(x, y, z)}$ to position $\boldsymbol{P'(x', y', z')}$ with the matrix operation.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Or $P' = T.P$

Parameters t_x, t_y and t_z specifying translation distances for the co-ordinate directions x, y and z are assigned any real values. The above matrix representation is equivalent to the three equations.

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$



Z-axis

Fig : Translating an object with translation vector $\boldsymbol{T = (t_x, t_y, t_z)}$

An object is translated in three dimensions by transforming each of the defining points of the object. For an object represented as a set of polygon surfaces, we translate each vertex of each surface and redraw the polygon facets in the new position.

We obtain the inverse translation matrix in the above matrix by negating the translation distances t_x, t_y and t_z. This produces a translation in the opposite direction and the product of a translation matrix and its inverse produces the identity matrix.

3.4.2 Transformations for 3-D Scaling

The process of compressing and expanding any object is called **Scaling**. Thus scaling alters the size of an object if scale factors. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result.

$S_x = S_y = S_z = S > 1$ The scaling is called magnification.

And if $S_x = S_y = S_z = S < 1$ the scaling is termed as reduction.

The points after scaling with respect to origin can be calculated by the following relation

$$P'(x', y', z') = S.P(x, y, z)$$

$$x' = S_x.x$$
$$y' = S_y.y$$
$$z' = S_z.z$$

The equation can be written in the matrix form as:-

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & a \\ 0 & S_y & 0 & b \\ 0 & 0 & S_z & c \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Where $S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Scaling with respect to a selected fixed position (x_f, y_f, z_f) can be represented with the following transformation sequence:

(1) Translation of the fixed point to the origin.

(2) Scaling of the object related to the coordinate origin.

(3) Translation of the fixed point back to its original position.

$$T(x_f,\ y_f,\ z_f)\ .S(S_x\ ,S_y\ ,S_z)\ .T(-x_f,-y_f,-z_f)$$

$$=\begin{bmatrix} S_x & 0 & 0 & (1-S_x\ )x_f \\ 0 & S_y & 0 & (1-S_y\ )y_f \\ 0 & 0 & S_z & (1-S_z\ )z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The following figure shows the effect of 3D scaling.

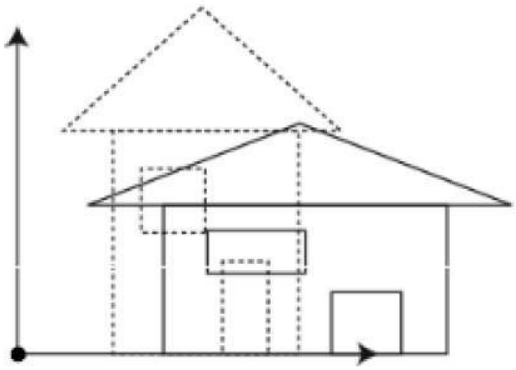


Fig :- Three Dimensional Scaling

3.4.3 Transformations for 3-D Rotation

Rotation in three dimensional geometry is more complex than that of two dimensional, because in this case we consider the angle of rotation and axis of rotation. Therefore, there may be three cases, when we choose one of the positive $x-axis$,$y-axis$,$z-axis$, as an axis of rotation.

(a) Rotation about x-axis (i.e. axis of rotation is x):

X

Y

$\boldsymbol{r}$

$\boldsymbol{\alpha}$ **θ** $\boldsymbol{p'(x', y'0)}$

$\boldsymbol{P(x, y, 0)}$

Z

From the figure : $x' = x$

$$y' = r\cos(\alpha + \theta) = r\cos\alpha.\cos\theta - r\sin\alpha.$$

$$\text{since } r\cos\alpha = y$$

and $r\sin\alpha = z$

or, $y' = y\cos\theta - z\sin\theta$

$$z' = r\sin(\alpha + \theta) = r\sin\alpha \cdot \cos\theta + r\cos\alpha \cdot \sin\theta$$

$$= z\cos\theta + y\sin\theta$$

$$x' = x$$

or $R_{\theta,i} = y' = y\cos\theta - z\sin\theta$

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

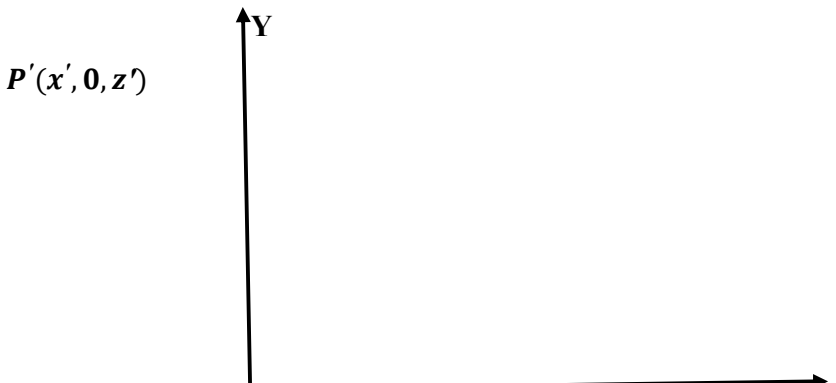
The points after rotation can be calculated by the equation

$$P'(x', y', z') = R_{\theta,i} \,.\, P(x, y, z)$$

When x', y' *and* z' are defined by the above equation. The above equation can be written into matrix form as follows.

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

b) Rotation about y-axis (i.e. y-axis is the axis of rotation):-



Rotation about Y-axis

$\boldsymbol{P(x, 0, z)}$

θ

X

$\boldsymbol{\alpha}$

Fig-

$$x' = xcos\theta + zsin\theta$$

Similarly $$y' = y$$

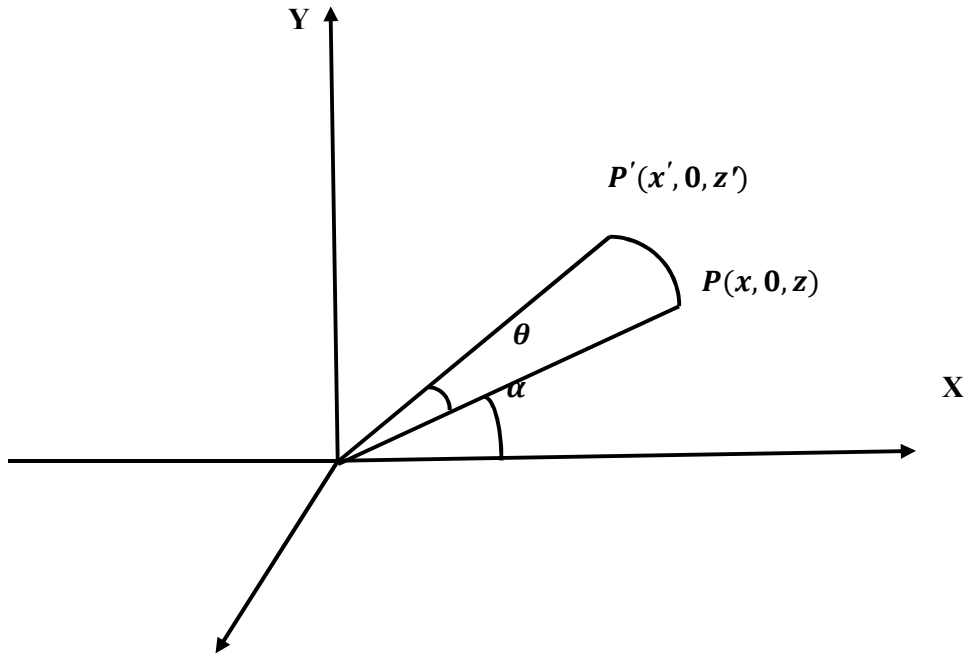
$$z' = -xsin\theta + zcos\theta$$

$$R_{\theta,i} = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix}$$

And the equation in the matrix form,

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

c) Rotation about Z-axis (i.e. Z-axis is the axis of rotation):-



Rotation about Z-axis

Z

$$x' = xcos\theta - ysin\theta$$

Similarly $$z' = z$$

$$y' = zsin\theta + ycos\theta$$

$$R_{\theta,k} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

And points after rotation are given by the following equation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Here direction of a positive angle of rotation has been taken in accordance with the right handed rule with the axis of a rotation.

3.4.4 Transformations for 3-D Shearing

Shear along any pair of axes is proportional to the third axes for instance, to shear along ***Z*** in **3D, x** and **y** values are altered by an amount proportional to the value of ***Z,*** leaving unchanged. Let Sh_{zx} , Sh_{zy} Is the shear due to ***Z*** along ***x and y*** directions respectively and are real values. Then the matrix representation is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Sh_{zx} & Sh_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear for x, y axis is similar to that of ***Z.*** The general form of shear is given by

$$\begin{bmatrix} 1 & Sh_{xy} & Sh_{xz} & 0 \\ 0 & 1 & Sh_{yz} & 0 \\ Sh_{yx} & Sh_{zy} & 1 & 0 \\ Sh_{zx} & 0 & 0 & 1 \end{bmatrix}$$

In 2- D shear transformation we discussed shear transformations relative to the x or, y-axis to produce distortions in the shapes of planer objects. In the 3D we can also generate shears relative to z-axis and the result is change of volume and the 3-D shape of any object.

Shear

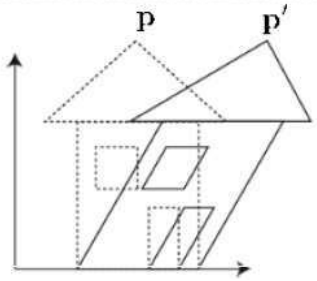


Fig :- Resultant of 3D Shearing

Example for shear along Z- axis

$$x' = x + az$$
$$y' = y + bz$$
$$z' = z$$

The corresponding transformation matrix

$$[T_{SZ}] = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

The effect of this transformation matrix is to alter **x** and **y** coordinate values by an amount that is proportional to ***Z*** value, while leaving the ***Z***-coordinate unchanged.

3D shearing transformation can be carried out about the other two principal axes as well. An X-axis 3D shear can be expressed as,

$$x' = x$$
$$y' = y + ax$$
$$z' = z + bx$$

The corresponding transformation matrix is,

$$[T_{Sx}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly Y-axis 3D shear can be expressed as,

$$x' = x + y$$
$$y' = y$$
$$z' = z + by$$

The corresponding transformation matrix is,

$$[T_{SY}] = \begin{bmatrix} 1 & a & 0 & 0 \\ a & 1 & 0 & 0 \\ b & b & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.5 Co-ordinate Transformations :-

A co-ordinate transformation is a mathematical operation which transforms the co-ordinates of a point in one system to the co-ordinate of the same point in another co-ordinate system. Also there exist on inverse transformation to get back to the first co-ordinate system by suitable mathematical operations. Co-ordinate transformations are frequently used in geodesy, surveying, photogrammetric and related branches. In general, the effect of a transformation of 2D or 3D object varies from a simple change of location and orientation, without any change in shape or size.

Affine Transformation

A co-ordinate system of the form

$$x' = A_1 x + B_1 y + C_1$$
$$y' = A_2 x + B_2 y + C_2$$

Where A_i, B_i, C_i are parameters fixed for a given transformation is called *affine transformation.* Obviously each of the transformed co-ordinates x' and y' is a linear function of the original co-ordinates x and y. Affine transformations have general properties that parallel lines and finite points map to finite points. An affine transformation that involves Translation, Rotation and Reflection preserve the length of an angle between two lines.

Most general transformation model is the affine transformation. It changes the position, size and shape of a network. The escape factor of such transformation depends on the orientation but not only position within the net. Here the lengths of all lines in a certain direction are multiplied by the same scalar. 3d affine transformations have been widely used as computer vision and particularly in the area of model based object recognition. Different numbers of parameters may be involved in this transformation.

i. **8-Parameters Affine transformation** (two translation, three rotations, two scale factors and skew distortion within image space) to describe a model that transform 3D object space to 2D space.

ii. **9-Parameter Affine Transformation** (three translations, three rotations, three scales) can be used in reconstructing the relief and evaluating the geometric features of the original documentation of the cultural heritage by 3D modeling.

iii. **12-Parameter Affine transformation** (3D translation, 3D rotation, different scale factor along each axis and 3D skew) used to define relationship between two

3D image volumes. For instance, in medical image computing, the transformation model is part of different software programmes that compute fully automatically the spatial transformation that maps points in one 3D image volume into their geometrically corresponding points in another related 3D image.

3.6 SUMMARY

In this chapter, we have concluded that 2D and 3D objects are represented by points and lines that join them. That is, Transformations can be applied only to the points defining the lines. The Unit answers the following facts :-

a) How do we represent a geometric object in the plane?

b) How do we transform a geometric object in the plane ?

c) How can we scale an object without moving its origin ?

d) How can we rotate an object without moving its origin ?

e) . Rigid transformation:– Translation + Rotation (distance preserving).

- Similarity transformation:– Translation + Rotation + uniform Scale (angle preserving).
- Affine transformation:– Translation + Rotation + Scale + Shear (parallelism preserving).

f) In Composite Transformation :- A sequence of transformations can be collapsed into a single matrix.

g) Homogeneous Coordinates is a mapping from Rn to Rn+1 Dimension.

h) Affine Transformations

3.7 QUESTIONS FOR EXERCISE

1) A square consists of vertices A(0,0),B(0,1),C(1,1),D(1,0). After the translation C is found to be at the new location (6,7). Determine the new location of other vertices.

2) A square ABCD is given with vertices A(0,0),B(1,0),C(1,1), and D(0,1). Illustrate the effect of a) x-shear b) y-shear c) xy-shear on the given square, when a=2 and b=3

3) Find the new coordinates of a triangle A(0,0),B(1,1),C(5,2) after it has been (a) magnified to twice its size and (b) reduced to half its size.

4) A point P(3,3) makes a rotating of 450 about the origin and then translating in the direction of vector v=5I+6J. Find the new location of P.

5) Show that the order in which transformations are performed is important by applying the transformation of the triangle ABC by:

(i) Rotating by 45o about the origin and then translating in the direction of the vector (1,0), and

(ii) Translating first in the direction of the vector (1,0), and then rotating by 45o about the origin, where A = (1, 0) B = (0 ,1) and C = (1, 1).

6) Give a single 3x3 homogeneous coordinate transformation matrix, which will have the same effect as each of the following transformation sequences.

a) Scale the image to be twice as large and then translate it 1 unit to the left.

b) Scale the x direction to be one-half as large and then rotate counterclockwise by 900 about the origin.

c) Rotate counterclockwise about the origin by 900 and then scale the x direction to be one-half as large.

d) Translate down ½ unit, right ½ unit, and then rotate counterclockwise by 450.

3.8 SUGGESTED READINGS

Donald Hearn, M. Pauline Baker, “Computer Graphics – C Version”, Pearson Education, 2007

Study Of Geometric Transformations In Computer Graphics- 2017 By Dr. Vijay Kumar Singh

Ch-2 Lecture Notes © by R. B. Agarwal Computer Aided Design in Mechanical Engineering

William M. Newman, Robert F. Sproull, “Principles of Interactive Computer Graphics”, Tata-McGraw Hill, 2000

J.D. Foley, A.Dam, S.K. Feiner, J.F. Hughes, “Computer Graphics – principles and practice”, Addison-Wesley, 1997

Harrington,S. "Computer Graphics - a programming approach." McGraw-Hill

UNIT -4

GRAPHICS STANDARDS

4.0 Objective

4.1 Introduction

4.0 Objective

The novelty of this Unit in the course is to appreciate the Standards adopted in Computer Graphics. It aims to enlighten about :-

- GKS a graphics system which allows programs to support a wide variety of graphics devices. It is defined independently of programming languages.
- To provide the application programmer with a good understanding of the principles behind GKS
- To serve as an informal manual for GKS.

4.1 Introduction

Standards in computer graphics were estabilished in late 70s to early 80s. Whereas de facto standards in programming languages were common very early on (FORTRAN and ALGOL 60) and international standards soon followed, there had been a long period of graphics history

where, at best, regional de facto standards have existed and no international standards had evolved. The following international organizations involved to develop the graphics standards:

- ACM (Association for Computer Machinery)
- ANSI (American National Standards Institute)
- ISO (International Standards Organization)
- GIN (German Standards Institute)

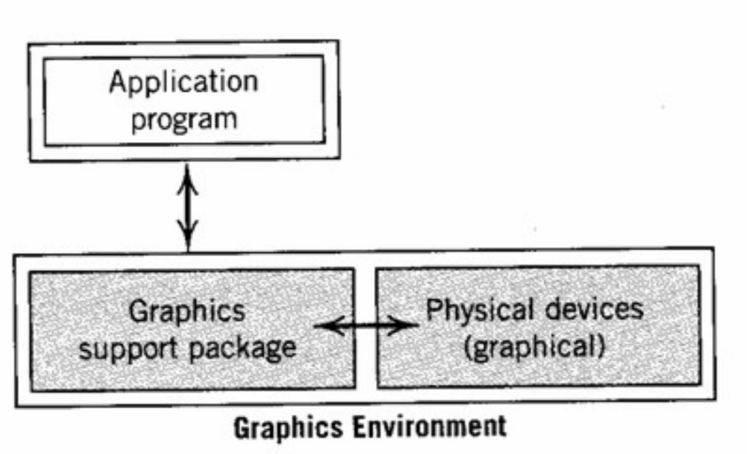


Figure :- An Interactive Graphics System

The graphics system is divided into two parts: the kernel system, which is hardware independent and the device driver, which is hardware dependent. The kernel system, acts as a buffer independent and portability of the program. At interface 'X', the application program calls the standard functions and sub routine provided by the kernel system through what is called language bindings. These functions and subroutine, call the device driver functions and subroutines at interface 'Y' to complete the task required by the application program

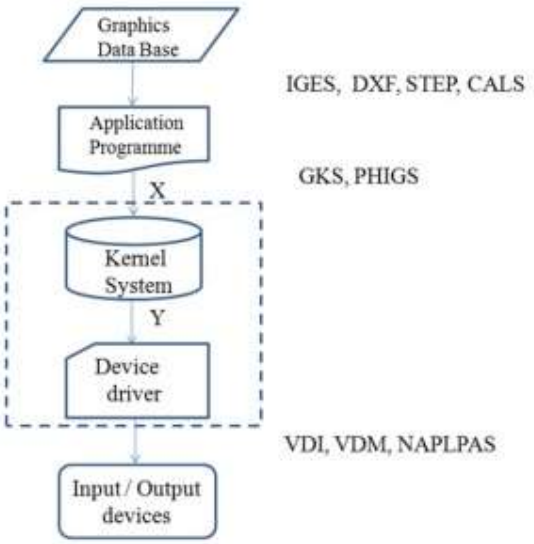


Fig.4.2. Graphics Standards in Graphics Programming

4.2 Types of Graphic Standards

As a result of these international organization efforts, various standard functions at various levels of the graphics system developed. These are:

1. **IGES** (Initial Graphics Exchange Specification) enables an exchange of model data basis among CAD system.
2. **DXF** (Drawing / Data Exchange Format) file format was meant to provide an exact representation of the data in the standard CAD file format.
3. **STEP** (Standard for the Exchange of Product model data) can be used to exchange data between CAD, Computer Aided Manufacturing (CAM) , Computer Aided Engineering (CAE) , product data management/enterprise data modeling (PDES) and other CAx systems.
4. **CALS** (Computer Aided Acquisition and Logistic Support) is an US Department of Defense initiative with the aim of applying computer technology in Logistic support.
5. **GKS** (Graphics Kernel System) provides a set of drawing features for two-dimensional vector graphics suitable for charting and similar duties.

6. **PHIGS** (Programmer'sHierarchical Interactive Graphic System) The PHIGS standard defines a set of functions and data structures to be used by a programmer to manipulate and display 3-D graphical objects.

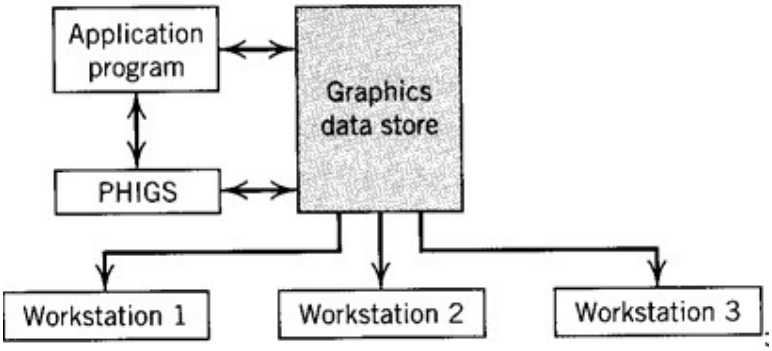


Figure :- PHIGS stores data at a centralized location

7. **VDI** (Virtual Device Interface) lies between GKS or PHIGS and the device driver code. VDI is now called CGI (Computer Graphics Interface).

8. **VDM (**Virtual Device Metafile) can be stored or transmitted from graphics device to another. VDM is now called CGM (Computer Graphics Metafile).

9. **NAPLPS** (North American Presentation- Level Protocol Syntax) describes text and graphics in the form of sequences of bytes in ASCII code.

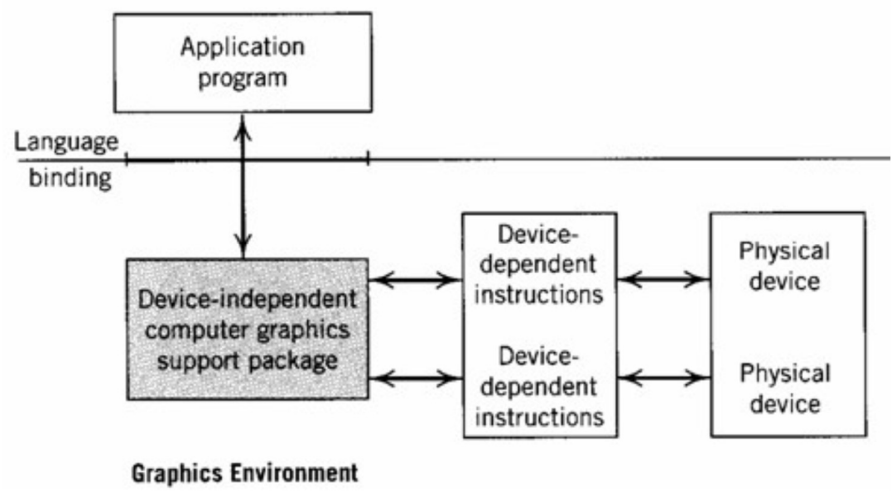


Figure 5.3 :- Model for Standardization of Graphics Environment

4.3 The Graphical Kernel System

The Graphical Kernel System (GKS) is the first international standard for computer graphics.

It was established in 1977. GKS offers a group of drawing aspects for 2D vector graphics appropriate for mapping and related duties. It serves as base for programming computer graphics applications. GKS covers the most significant parts of the area of generative computer graphics. It also lends itself for use with applications out of the areas of picture analysis and picture processing. GKS offers functions for picture generation, picture presentation, segmentation, transformations and input.

The calls are defined to be moveable across various programming languages, graphics hardware, so that applications noted to use GKS will be willingly portable to different devices and platforms.

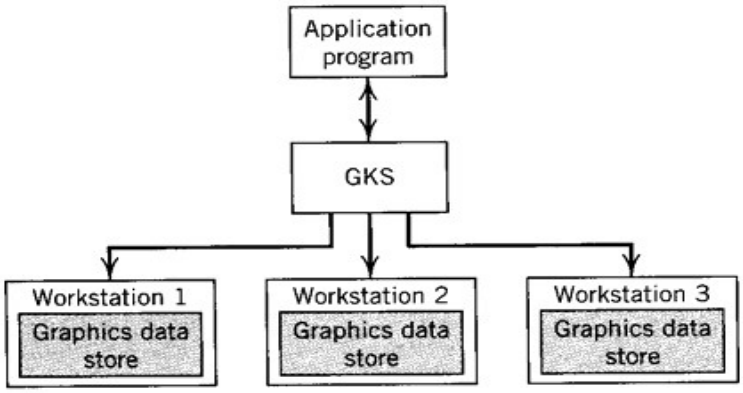


Figure :- GKS stores Graphic data at Workstation Level

The main concepts of a graphics system are closely related to the tasks of such a system. Among these tasks are:

- generation and representation of pictures;
- routing parts of the pictures created in different user coordinate systems to different workstations and transforming them into the respective device coordinate systems;
- controlling the workstations attached to the system;
- handling input from workstations;
- allowing the structuring of pictures into parts that can be manipulated (displayed, transformed, copied, deleted) separately;
- long time storage of pictures.

An important aspect of a graphics system is the dimensionality of the graphical objects it processes. The current GKS standard defines a purely two-dimensional (2D) system. However, efforts are under way to define a consistent 3D extension. The major GKS concepts are outlined in the following sections:

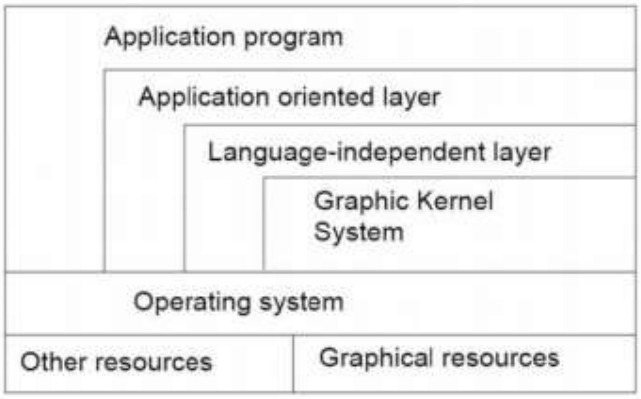


Figure :- Layers of GKS

4.3.1 GKS Standards

The following documents are representing GKS standards:

- The language bindings are called in ISO 8651 standard.
- ANSI X3.124 (1985) is part of ANSI standard.
- ISO/IEC 7942 noted in ISO standard, first part of 1985 and two to four parts of 1997-99.
- ISO 8805 and ISO 8806.

The main uses of the GKS standard are**:**

- To give for portability of application graphics programs.
- To assist in the learning of graphics systems by application programmers.
- To offer strategy for manufacturers in relating practical graphics capabilities.

The GKS consists of three basic parts:

i) A casual exhibition of the substances of the standard which contains such things as how text is placed, how polygonal zones are to be filled, and so onward.

ii) An official of the **descriptive** material in (i), by way of conceptual the ideas into separate functional explanations. These functional descriptions have such data as descriptions of input and output parameters, specific descriptions of the result of every function should have references into the **descriptive** material in (i), and a description of fault situation. The functional descriptions in this division are language autonomous.

iii) Language bindings are an execution of the abstract functions explained in (ii). in a explicit computer language such as C.

GKS arrange its functionality into twelve functional stages, based on the complexity of the graphical input and output. There are four stages of output (m, 0, 1, 2) and three stages of input (A, B, C). NCAR GKS has a complete execution of the GKS C bindings at level 0 A.

4.3.2 GKS Input

With input, the new dimension of interactivity is added to GKS, The actions of pointing, selecting, sketching, placing or erasing in a direct manner and the instantaneous system response to these actions are truly adapted to the human way of dealing with his environment,

Besides input that is specific for graphical applications (coordinate data or the identification of a part of the picture). GKS also handles alphanumeric input, choice devices like function keys, and value-delivering devices like potentiometer dials. GKS handles input in a device-independent way by defining logical- input devices. Each logical input device can be operated, in one of three different operating modes (REQUEST, SAMPLE and EVENT). Depending on the mode, input values can be entered by the operator and passed to the application program in different ways: one value at a time, requested by the application program and supplied by an operator action (REQUEST); sampling an input device irrespective of an operator action (SAMPLE); and input values collected in a queue by operator actions (EVENT).

4.3.3 GKS Segmentation

In many application areas, there is a need to display the same or similar graphical information many times possibly at different positions on the device. This leads to the need for some storage mechanism whereby pictures or subpictures can be saved for later use during a program's execution. GKS has the concept of a segment for this purpose. The task of manipulating parts of

the pictures leads to the concept of segmentation, A picture is composed of parts called segments that can be displayed, transformed, copied, or deleted independently of each other.

Graphical output can be stored in segments during the execution of a program and later reused. Segments have a number of segment attributes associated with them which allow the user to modify the appearance of the whole segment in a significant way when it is reused.Segments can be identified by an operator and their identification passed to the application program, GKS contains a very powerful segment facility, primarily by providing a device-independent segment storage, together with functions for copying segments to workstations or into other segments.

4.3.4 Graphics Output Primitives

One of the basic tasks of a graphics system is to generate pictures. The concept corresponding to this task is graphical output. The objects from which a picture is built up are output primitives, given by their geometrical aspects and by the way how they appear on the display surface of a workstation. The way to present objects is controlled by a set of attributes that belong to a primitive (e.g., colour, linewidth). Certain attributes may vary from one workstation to the other. E.g., a line may appear on one workstation black and dashed, on the other one red and solid. These aspects of a primitive are called workstation-dependent attributes.

In GKS, functions are present for the creation of primitives and for the setting of attributes (including workstation-dependent attributes).

GKS is based on a number of elements that may be drawn in an object know as graphical primitives.

GKS has output primitives that allow the convenient addressing of line graphics devices as well as special output primitives for addressing raster device capabilities. However, raster primitives will be displayed on line graphics devices as well; and line primitives will be displayed on raster devices. Line drawing primitives are: **POLYLINE** and **POLYMARKER**, the text primitive is **TEXT**, raster primitives are **PIXEL ARRAY** and **FILL AREA**, a special escape-primitive function is provided for addressing device capabilities, the GENERALIZED DRAWING PRIMITIVE **(GDP).**The fundamental set of primitives has the word names POLYLINE, POLYMARKER, FILLAREA, TEXT and CELLARRAY, even though a few implementations widen this basic set.

i) **POLYLINES**

The GKS function for drawing line segments is called 'POLYLINE'The. 'POLYLINE' command takes an array of X-Y coordinates and creates line segments joining them. The elements that organize the look of a 'POLYLINE' are :-

- *Line type* : solid, dashed or dotted.
- *Line width scale factor* : thickness of the line.
- *Polyline color index* : color of the line.

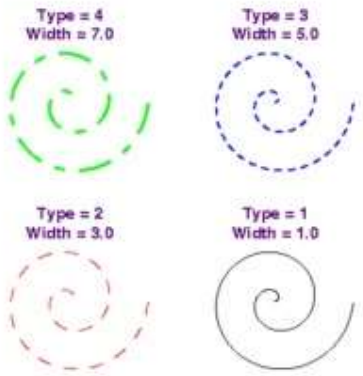


Fig.5.3. GKS POLYLINES

ii) **POLYMARKERS**

The GKS POLYMARKER function permits to draw symbols of marker centered at coordinate points. The features that control the look of 'POLYMARKERS'are :

· Marker characters : dot, plus, asterisk, circle or cross.

· Marker size scale factor : size of marker

· Polymarker color index : color of the marker.

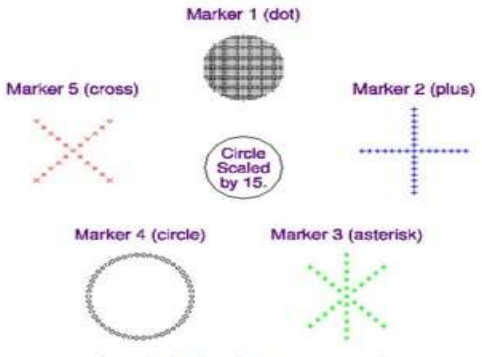


Fig.5.4. GKS POLYMARKERS

Fig.:-. GKS POLYMARKERS

iii**) FILLAREA**

The GKS 'FILLAREA' function permits to denote a polygonal shape of a zone to be filled with various interior shapes. The features that control the look of fill areas are :-

· *FILL AREA interior style* : solid colors, hatch patterns.

· *FILL AREA style index* : horizontal lines; vertical lines; left slant lines;right slant lines; horizontal and vertical lines; or left slant and right slant lines.

· *FILLAREA color index* : color of the fill patterns / solid areas.

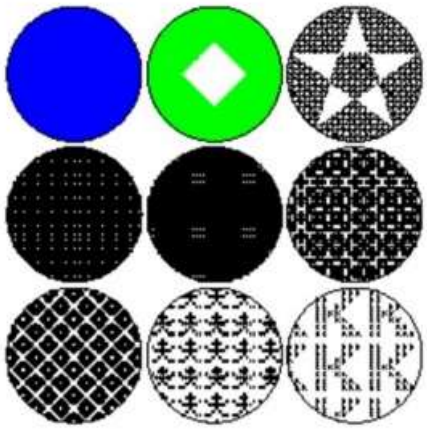


Fig.5.5. GKS FILLAREA

Fig.. GKS FILLAREA

iv) TEXT

The GKS TEXT function permits to sketch a text string at a specified coordinate place. The features that control the look of text are:

· Text font and precision : text font should be used for the characters

· Character expansion factor : height-to-width ratio of each character.

· Character spacing : additional white space should be inserted between characters

· Text color index : color the text string

· Character height : size of the characters

· Character up vector : angle the text

· Text path : direction the text should be written (right, left, up, or down).

· Text alignment : vertical and horizontal centering options for the text string.

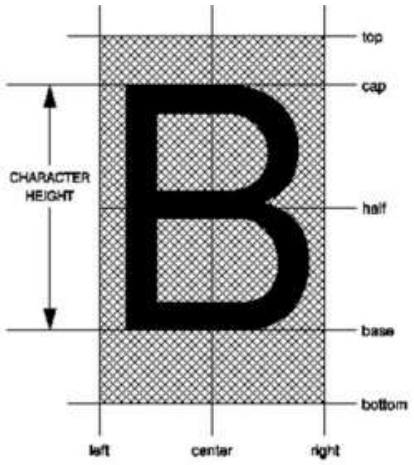


Fig.5.6. GKS TEXT

Fig. GKS TEXT

v) CELL ARRAY

The GKS CELL ARRAY function shows raster like pictures in a device autonomous manner. The CELL ARRAY function takes the two corner points of a rectangle that indicate, a number of partitions (M) in the X direction and a number of partitions (N) in the Y direction. It then partitions the rectangle into M x N sub rectangles noted as cells.

4.3.5 Coordinate Stems and Transformations

The application program can use one or several user coordinate systems that are related to the application for the creation of graphical elements. Output devices that are used for presenting the visual image of the elements, however, normally require the use of a device specific coordinate system. The routing and the transformation of output primitives along this output pipeline is performed by GKS. By using appropriate functions, the output transformations can be controlled by the application program.

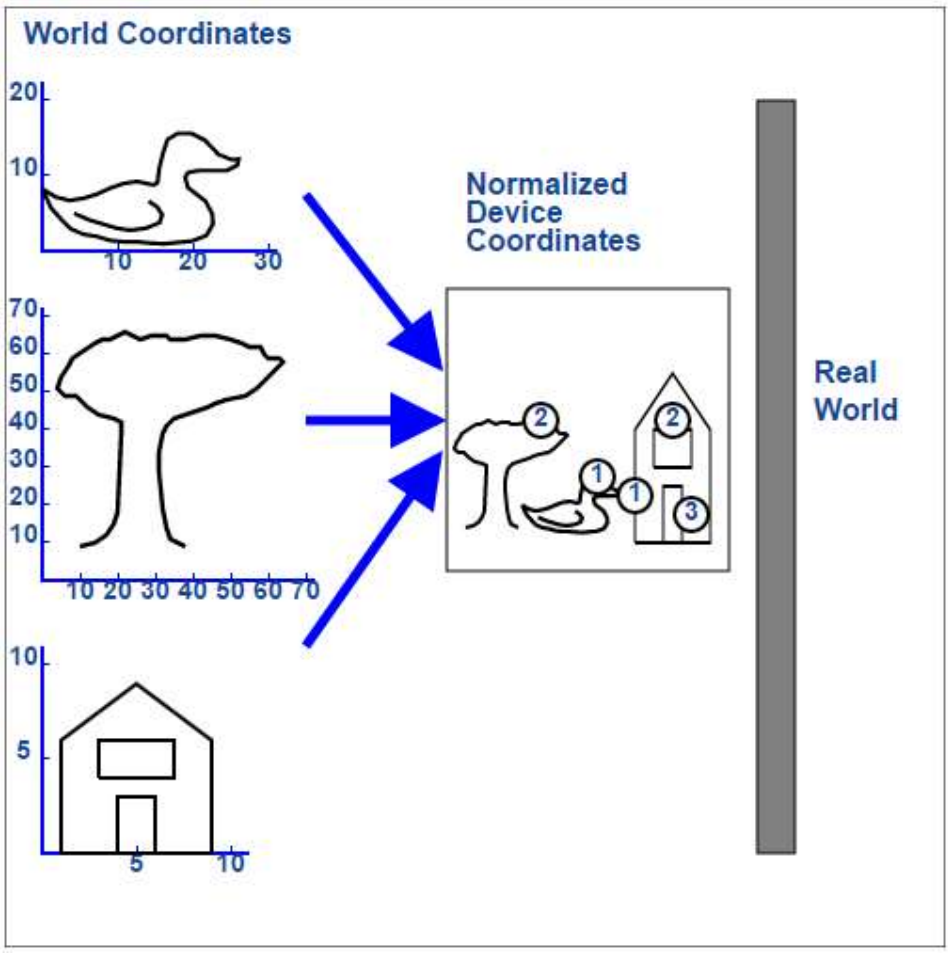


Figure :- GKS Object rasterisation by Polylines

4.4 The GKS Workstation

A workstation is a very useful and important concept: GKS uses workstations for input and output. A workstation is a display plus a number of input devices attached to a single line or channel. Workstations have only a single display surface but may have any number of input devices. A workstation is, e.g., a plotter or a display with a keyboard or a tablet connected to it. The workstation concept is one of the original contributions of GKS to the methodology of graphics system design. The graphical workstations of GKS are an abstraction of physical devices. An abstract graphical workstation can have one display surface and a number of input devices. Output can be sent selectively or in parallel to one or several workstations. Also, input can be obtained from different workstations.

There are three types of workstations:

INPUT only OUTPUT only OUTIN both input and output

A mechanism for long term storage of graphical information is the metafile. In GKS this important type of output device is given a special name - GKSM. The metafile is looked upon as just being another workstation - either INPUT or OUTPUT type.

Different formats for the metafile could be defined as different number workstations by the systems manager. This could cater for metafiles that are not produced by GKS but are common enough to warrant facilities for interpretation / creation of that format.

A GKSM stores type information, data record length information and then the data itself.

The output devices and several input devices are assembled into groups called graphical workstations. They usually are operated by a single operator.

4.5 The GKS Metafile

The metafile concept results from the need to store pictures for archiving purposes or for transfer to a different location or different system. The advent of low cost interactive graphics drivers in the 1960s led to a phenomenal change in the field of Computer Graphics. The integration of graphics in interactive programming was a challenge which led to the development of a variety of graphic systems. Thus, the synthesis of Graphical Systems can be analogous to the outset of Computer Graphics as a discipline.

GKS addresses a metafile called GKS metafile (GKSM) that allows for long-term storage and retrieval of pictures. The metafile interface of GKS adds considerably to the flexibility of the system. GKS provides metafiles for the storage of graphical information. Their principal uses are:

1. transporting graphical information between computer systems
2. transporting graphical information from one site to another (by magnetic tape for eg.)
3. device spooling, e.g. for a plotter

The CGM(Computer Graphics Metafile) provides a means of graphics data interchange for computer representation of 2D graphical information independent from any particular application, system, platform, or device. As a metafile, i.e., a file containing information that describes or specifies another file, the CGM format has numerous elements to provide functions and to represent entities, so that a wide range of graphical information and geometric primitives can be accommodated. Rather than establish an explicit graphics file format, CGM contains the instructions and data for reconstructing graphical components to render an image using an object-oriented approach.

Although CGM is not widely supported for web pages and has been supplanted by other formats in the graphic arts, it is still prevalent in engineering, aviation, and other technical applications.

A GKS metafile output workstation has the following characteristics:

1. Output functions are stored if the workstation is active.

2. Attribute functions are stored.
3. Segments are stored if the workstation is active.
4. Geometric data is stored in a form equivalent to NDC.
5. Non-GKS data may be written using the special function WRITE ITEM TO GKSM.

A metafile is regarded by GKS as a sequence of items, each of which has three components:

1. Item type
2. Item data record length
3. Item data record

The initial CGM implementation was effectively a streamed representation of a sequence of Graphical Kernel System primitive operations. It has been adopted to some extent in the areas of technical illustration and professional design, but has largely been superseded by formats such as SVG and DXF.

As part of the standard, the GKS document contains a definition of the interface to and from the GKSM. The contents and the format of the GKSM are described in an appendix that is not part of the standard. This separation was done in order to allow for a development of standardized graphics metafile independently of specific systems or devices.

Error handling

GKS contains an error handling facility. All errors expected during system operation are listed. A standard error handling procedure is provided. However, the user can replace it by his own error handling.

4.6 The GKS Interface

GKS is based on the concept of abstract graphical workstations. These workstations provide the logical interface through which an application program controls physical devices.

All GKS workstations are conceptualized as having a single display surface of fixed resolution allowing only rectangular display spaces. These workstations support drawing lines, plotting text, filling polygons, and so on. GKS workstations fall into several types depending on whether they support graphical output or graphical input, or store graphical instructions.

This section discusses what types of physical workstations are available and how to direct output to specific workstations and how to store graphics instructions, either in a metafile for permanent storage, or in a segment for temporary storage and subsequent copying to other workstations.

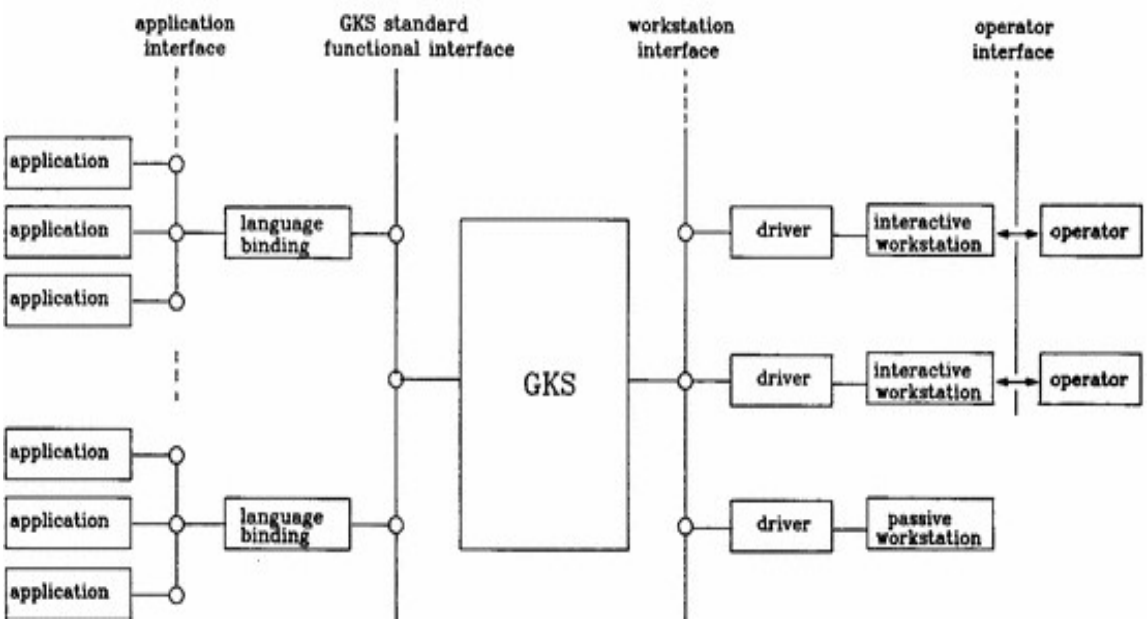
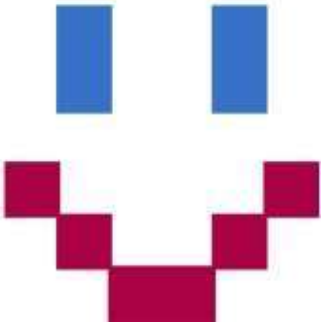


Fig :- The Graphical Kernel System Interfaces

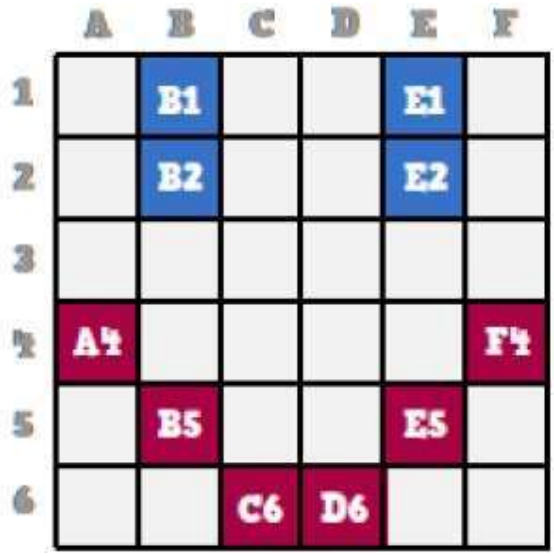
Device drivers of GKS are also called workstation drivers. The figure above shows the Graphical Kernel System as the nucleus between the application interface and the workstation interface. If the workstation interface is an interactive workstation with an operator; he is communicating with the system via the operator interface.

4.7 Bitmap Graphics

A bitmap is a collection of pixels that describes an image. It is a type of computer graphics that the computer uses to store and display pictures. In this type of graphics, images are stored bit by bit and hence it is named Bit-map graphics. For better understanding let us consider the following example where we draw a smiley face using bit-map graphics.



Now we will see how this smiley face is stored bit by bit in computer graphics.



By observing the original smiley face closely, we can see that there are two blue lines which are represented as B1, B2 and E1, E2 in the above figure.

In the same way, the smiley is represented using the combination bits of A4, B5, C6, D6, E5, and F4 respectively.

The main disadvantages of bitmap graphics are −

- We cannot resize the bitmap image. If you try to resize, the pixels get blurred.
- Colored bitmaps can be very large.

4.8 SUMMARY

This unit emphasizes a common interface to interactive computer graphics for application programs. Application programs can thus move freely between different graphics devices and different host computers. The chapter describes :-

a. What do you need to know about Graphic Standardisation
b. What are the key elements of a GKS workstation.
c. Details of GKS Input,Segmentation,Output Primitives and Metafiles

4.9 QUESTIONS FOR EXERCISE

1) Why do we have graphic standards?
2) Name the fundamental set of graphic output primitives.
3) Discuss the significance of writing Metafiles in GKS.
4) Exemplify the GKS Workstation categories of INPUT, OUTPUT, OUTIN

4.10 SUGGESTED READING

- http://www.chilton-computing.org.uk/acd/literature/books/gks/p007.htm
- "Status Report of the Graphic Standards Planning Committee of ACM/SIGGRAPH." Computer Graphics 13(3)
- "Special Issue: Graphics Standards." ACM Computing Surveys 10(4)
- F. R. A. Hopgood, D. A. Duce, J. R. Gallop, D. C. Sutcliffe Academic Press, 1986
- Encarnacao, J. et at.. 'The workstation concept of GKS and the resulting conceptual differences to the GSPC core system.' Computer Graphics 14(3), 226-230
- 'The Detailed Semantics of Graphics Input Devices.' Computer Graphics 16(3), 33-38 Rosenthal, D.S.H. et al
- "Computer Graphics Programming." Springer Verlag, Heidelberg, Enderle, G. et al
- "Information processing systems - Computer graphics - Graphical Kernel System (GKS) functional description." ISO, Geneva.

UNIT -5 3-D GRAPHICS,CURVES AND SURFACES

UNIT STRUCTURE

5.0 OBJECTIVE

With the edge of mathematics in computer graphics we can achieve realism. In this chapter we :-

- Study about the polygon projections,
- Categorize various types of Perspective and Parallel projections
- Develop the general transformation matrix for Parallel projection; Orthographic and Oblique parallel projections and for multi view (front, right, top, rear, left and bottom view) projections

- Describe and derive the projection matrix for single-point, two-point and three-point perspective transformations and identify the vanishing points.
- Understand the meaning of Visible-surface detection
- Distinguish between image-space and object-space approach for visible-surface
- Develop the Depth-buffer method; Scan-line method; Area-Subdivision method for visible-surface determination.

The unit adopts computer oriented approach to understand the implementation of mathematical concepts. We are going to discuss one more important topic in this unit, which is Bezier Curves and their properties.

5.1 INTRODUCTION TO 3D GRAPHICS

In the 2D system, we use only two coordinates X and Y but in 3D, an extra coordinate Z is added. 3D graphics techniques and their application are fundamental to the entertainment, games, and computer-aided design industries. It is a continuing area of research in scientific visualization. Furthermore, 3D graphics components are now a part of almost every personal computer and, although traditionally intended for graphics-intensive software such as games, they are increasingly being used by other applications.

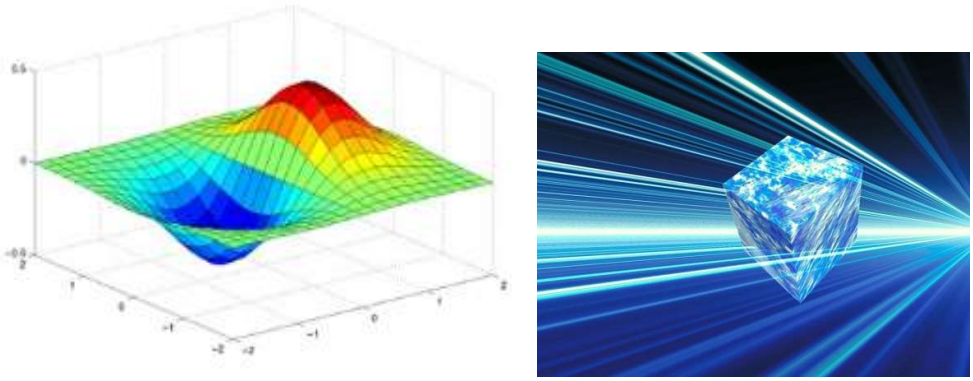


Fig : - Applications in 3D Modelling Technology

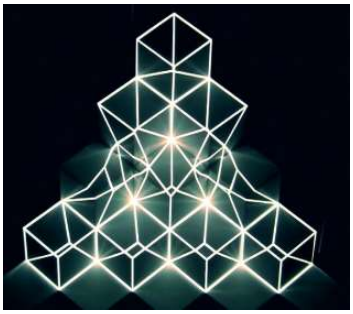
5.2 PROJECTIONS

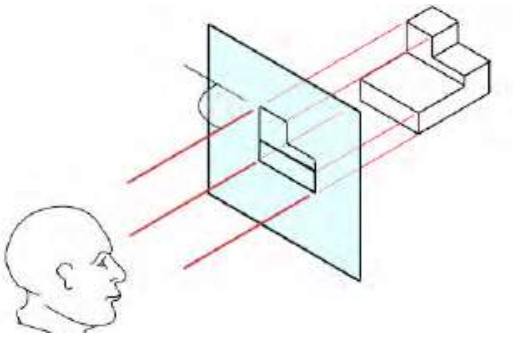
A transformation which maps 3-D objects onto 2-D screen, we are going to call it **Projections.** We have two types of Projections namely:-

A) Parallel projection

B) Perspective projection.

This categorisation is based on the fact whether rays coming from the object converge at the centre of projection or not. If, the rays coming from the object converge at the centre of projection, then this projection is known as ***Perspective projection***, otherwise it is ***Parallel projection***.





5.2.1 Parallel Projection

Parallel projection discards z-coordinate and parallel lines from each vertex on the object are extended until they intersect the view plane. In the case of parallel projection the rays from an object converge at infinity, unlike perspective projection where the rays from an object converge at a finite distance (called **COP**). In parallel projection, we specify a direction of projection instead of *center of projection*.

In parallel projection, the distance from the center of projection to project plane is infinite. In this type of projection, we connect the projected vertices by line segments which correspond to connections on the original object.

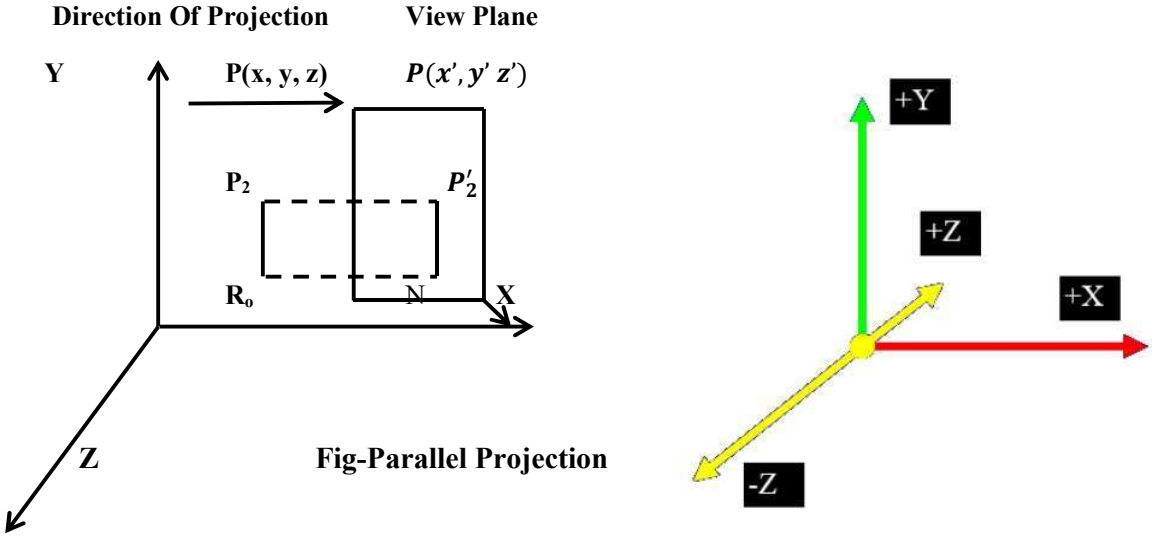
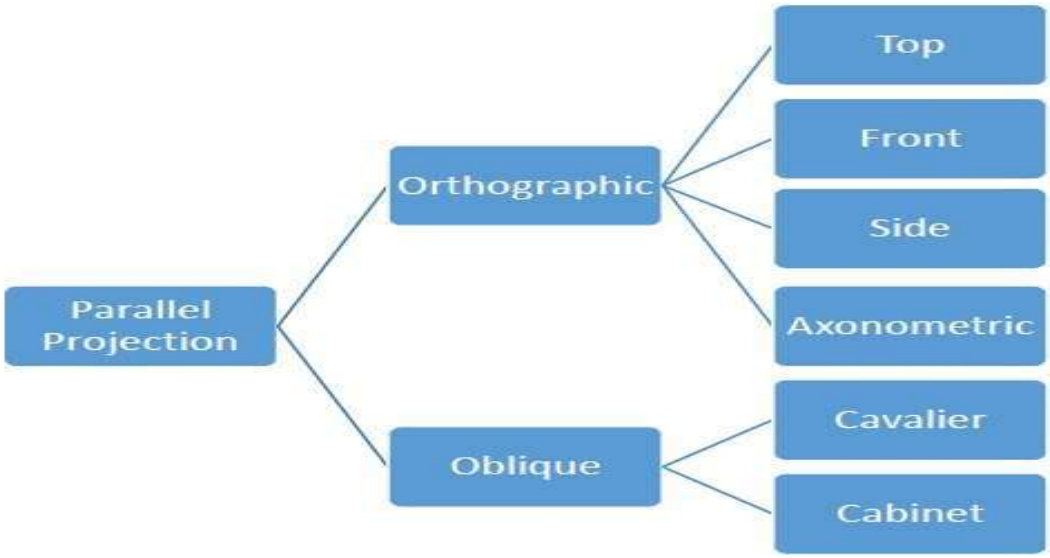


Fig-Parallel Projection

Parallel projection can be categorized according to the angle that the direction of projection makes with the projection plane If the direction of projection of rays is perpendicular to the projection plane then this parallel projection is known as ***Orthographic projection*** and if the direction of projection of rays is not perpendicular to the projection plane then this parallel projection is known as ***Oblique projection***.

Parallel projections are less realistic, but they are good for exact measurements. In this type of projections, parallel lines remain parallel and angles are not preserved.

Various types of parallel projections are shown in the following hierarchy :-

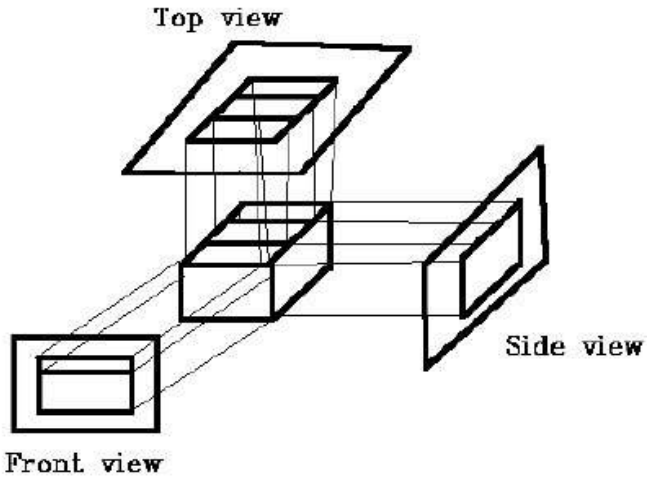


5.2.1.1 Orthographic Projection

In orthographic projection the direction of projection is normal to the projection of the plane. The orthographic (perpendicular) projection shows only the front face of the given object, which includes only two dimensions: length and width.

There are three types of orthographic projections −

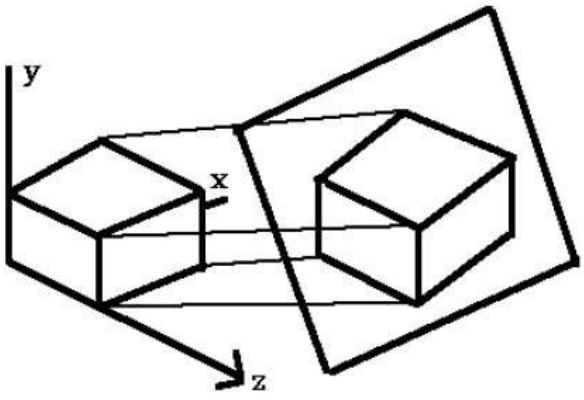
- Front Projection
- Top Projection
- Side Projection



Isometric Projections

Orthographic projections that show more than one side of an object are called **axonometric orthographic projections**. . Isometric projection is the most frequently used type of axonometric projection, which is a method used to show an object in all three dimensions (length, width, and height) in a single view. Axonometric projection is a form of orthographic projection in which the projectors are always perpendicular to the plane of projection.

In **isometric projection** ,the projection plane intersects each coordinate axis in the model coordinate system at an equal distance. In this projection parallelism of lines are preserved but angles are not preserved. The following figure shows isometric projection –



5.2.1.2 Oblique Projection

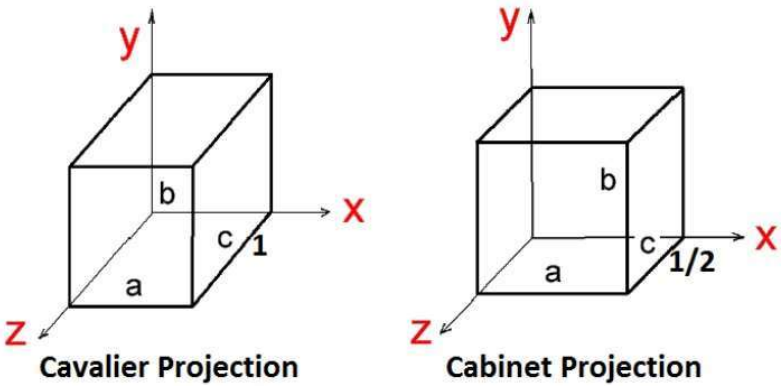
In orthographic projection, the direction of projection is not normal to the projection of plane. In oblique projection, we can view the object better than orthographic projection. The oblique projection, on the other hand, shows the front surface and the top surface, which includes three dimensions: length, width, and height.

Thus, oblique projection is one way to show all three dimensions of an object in a single view.

There are two types of oblique projections − **Cavalier** and **Cabinet**. The Cavalier projection makes 45° angle with the projection plane. The projection of a line perpendicular to the view plane has the same length as the line itself in Cavalier projection. In a cavalier projection, the foreshortening factors for all three principal directions are equal.

The Cabinet projection makes 63.4° angle with the projection plane. In Cabinet projection, lines perpendicular to the viewing surface is projected at ½ their actual length.

Both the projections are shown in the following figure −

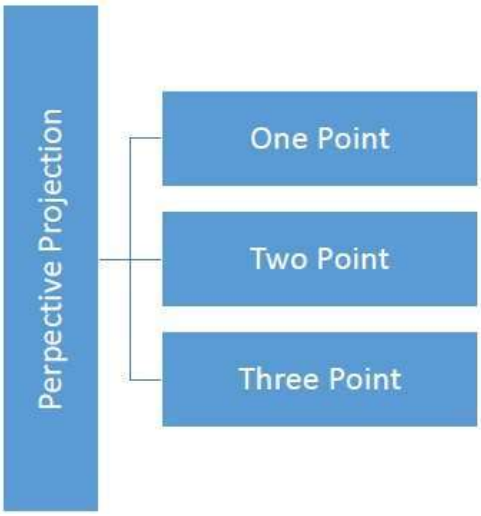


5.2.2 PERSPECTIVE PROJECTION

In perspective projection, the distance from the center of projection to project plane is finite and the size of the object varies inversely with distance which looks more realistic.

The distance and angles are not preserved and parallel lines do not remain parallel. Instead, they all converge at a single point called **center of projection** or **projection reference point**. There are 3 types of perspective projections which are shown in the following chart.

- **One point** perspective projection is simple to draw.
- **Two point** perspective projection gives better impression of depth.
- **Three point** perspective projection is most difficult to draw.



These are non linear transforms to obtain a perspective projection of a 3D objects we transform points along projection lines which are not parallel to each other and converge to meet at a finite point known as the center of projection.

If the centre of projection is at (x_c, y_c, z_c) and the point on the objects is (x_0, y_0, z_0), then the projection rays will be the line containing these point and will be given by the following, if projected co-ordinate is (x_p, y_p, z_p), then

$$X_p = x_c + (x_0 - x_c)\, t$$
$$Y_p = y_c + (y_0 - y_c)\, t$$
$$Z_p = z_c + (z_0 - z_c)\, t$$

Points (x_z, y_z) is points where line of projection intersects the *xy* plane i.e $z=0$

$$t = \frac{z_c}{z_0 - z_c}$$

Substitution into the first two equations gives,

$$x_p = x_c - z_c(x_0 - x_c/(z_0 - z_c)$$

Or, $$x_p = (x_c\, z_0 - x_0 z_c)/z_0 - z_c$$

$$y_p = y_c - z_c(y_0 - y_c/z_0 - z_c)$$

$$y_p = \frac{y_c z_c - y_0 z_0}{z_0 - z_c}$$

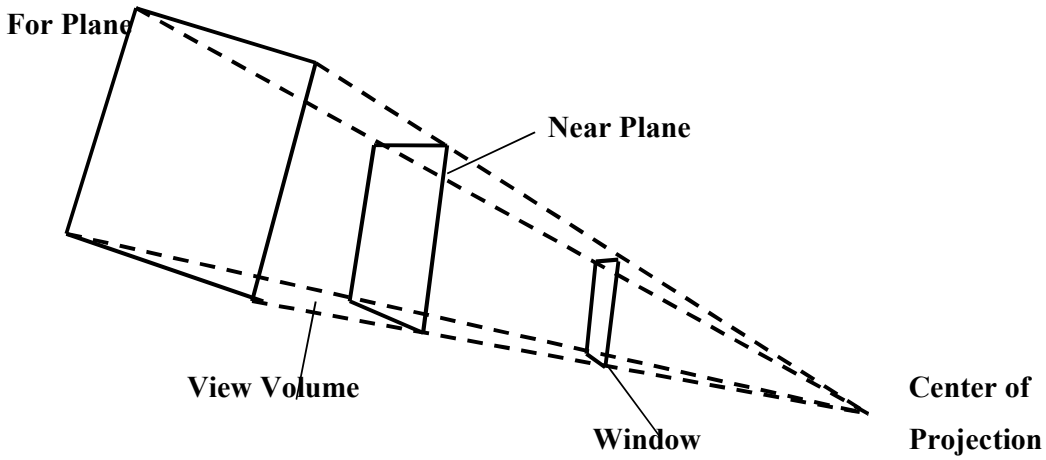


Fig-4.13

These equations can be written in the form of homogenous matrix as follows:-

$$P = \begin{bmatrix} -z_c & 0 & x_c & 0 \\ 0 & -z_c & y_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -z_c \end{bmatrix}$$

Similarly, we maintain the z-information and the resulting matrix is

$$P = \begin{bmatrix} -z_c & 0 & x_c & 0 \\ 0 & -z_c & y_c & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -z_c \end{bmatrix}$$

This perspective projection produces realistic views but doesn't preserve Relative proportions of objects. Projections of distant objects are smaller than the projections of objects of the same size that are closed to the projection plane. This characteristic feature (anomaly) of perspective projection is known as perspective foreshortening. Another characteristic feature (anomaly) of perspective projection is the illusion that after projection certain sets of parallel lines appear to meet at some point on the projection plane. These points are called vanishing points.

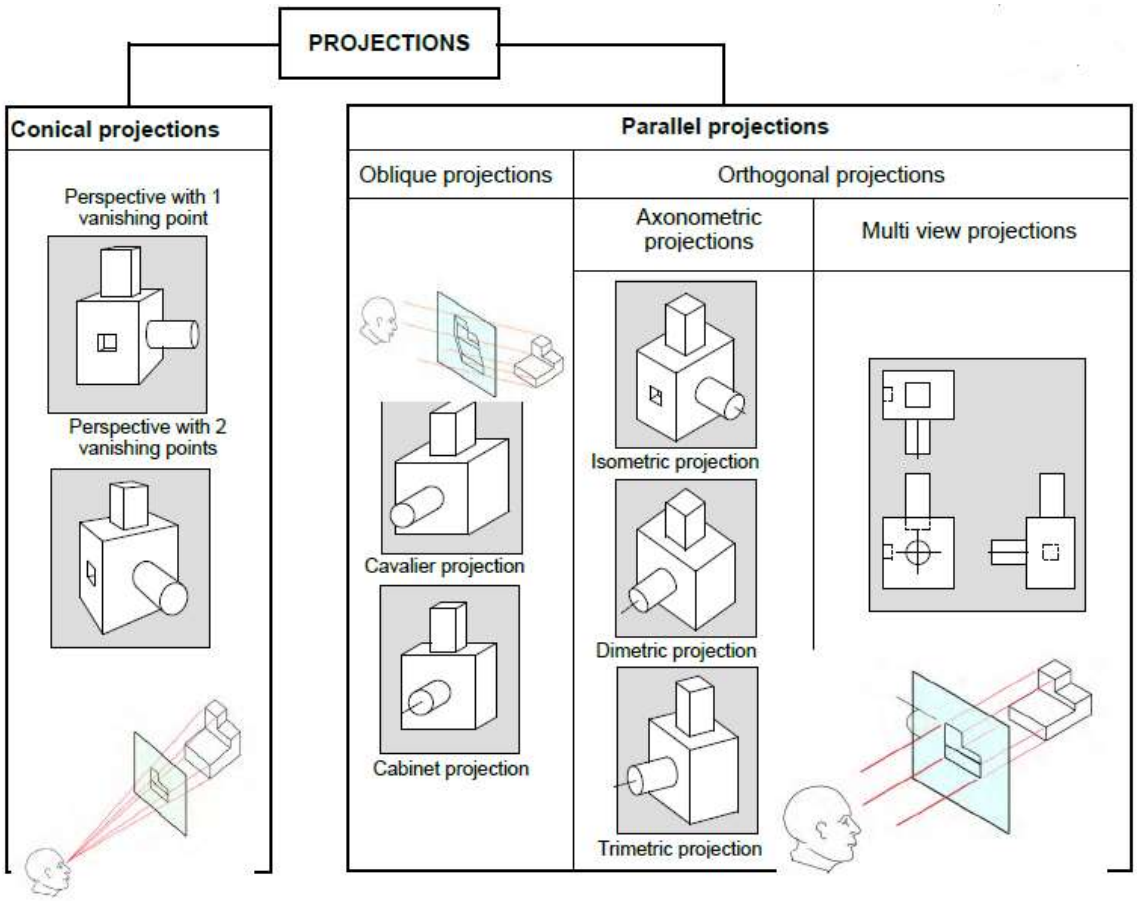


Fig :- Taxonomy of Projections

5.3 VISIBILITY AND HIDDEN SURFACE REMOVAL

In 3D computer graphics, **hidden surface determination** is the process used to determine which surfaces and parts of surfaces are not visible from a certain viewpoint. It is also known as **hidden surface removal** (HSR), **occlusion culling** (OC) or **visible surface determination** (VSD). When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which is behind from objects closer to eye. We must remove these hidden surfaces to get a realistic screen image. The identification and removal of these surfaces is called **Hidden-surface problem**.

The objects that lie behind opaque objects such as walls are prevented from being rendered. Despite advances in hardware capability there is still a need for advanced rendering algorithms. The responsibility of a rendering engine is to allow for large world spaces and as the world's size approaches infinity the engine should not slow down but remain at constant speed. Optimising this process relies on being able to ensure the deployment of as few resources as possible towards

the rendering of surfaces that will not. When we want to display a 3D object on a 2D screen, we need to identify those parts of a screen that are visible from a chosen viewing position.

A hidden surface determination algorithm is a solution to the visibility problem, which was one of the first major problems in the field of 3D computer graphics. The process of hidden surface determination is sometimes called hiding, and such an algorithm is sometimes called a hider. The analogue for line rendering is hidden line removal. Hidden surface determination is necessary to render an image correctly, so that one cannot look through walls in virtual reality.

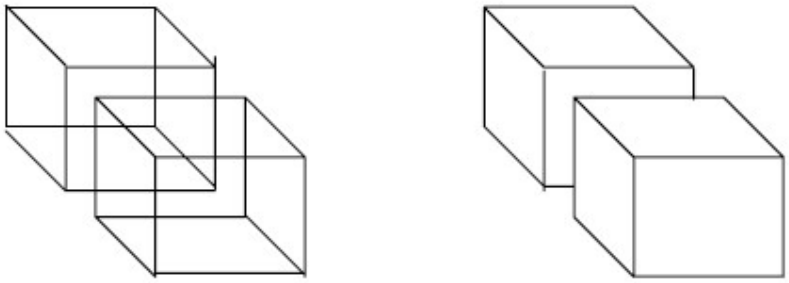


Fig :- A complex model Vs. Realistic view after removal of hidden lines

5.4 HIDDEN SURFACE REMOVAL ALGORITHMS

There are many techniques for hidden surface determination. They are fundamentally an exercise in sorting, and usually vary in the order in which the sort is performed and how the problem is subdivided. Sorting large quantities of graphics primitives is usually done by divide and conquer.

There are two approaches for removing hidden surface problems − **Object-Space method** and **Image-space method**. The Object-space method is implemented in physical coordinate system and image-space method is implemented in screen coordinate system.

In both cases, we can think of each object as comprising one or more polygons (or more complex surfaces).

The first approach (image-space) determines which of n objects in the scene is visible at each pixel in the image. The pseudocode for this approach looks like as:

for(each pixel in the image)

{ determine the object closest to the viewer that is passed by the projector through the pixel; draw the pixel in the appropriate color; }

This approach requires examining all the objects in the scene to determine which is closest to the viewer along the projector passing through the pixel. That is, in an image-space algorithm, the visibility is decided point by point at each pixel position on the projection plane. If the number of objects is ‘n’ and the pixels is ‘p’ then effort is proportional to n.p.

The second approach (object-space) compares all objects directly with each other within the scene definition and eliminates those objects or portion of objects that are not visible. In terms of pseudocode, we have:

for (each object in the world)

{ determine those parts of the object whose view is unobstructed (not blocked) by other parts of it or any other object; draw those parts in the appropriate color; }

This approach compares each of the n objects to itself and to the other objects, and discarding invisible portions. Thus, the computational effort is proportional to n^2. Image-space approaches require two buffers: one for storing the pixel intensities and another for updating the depth of the visible surfaces from the view plane.

5.4.1 Depth Buffer (Z-Buffer) Method

This method is developed by Cutmull. It is an image-space approach. The basic idea is to test the Z-depth of each surface to determine the closest (visible) surface.

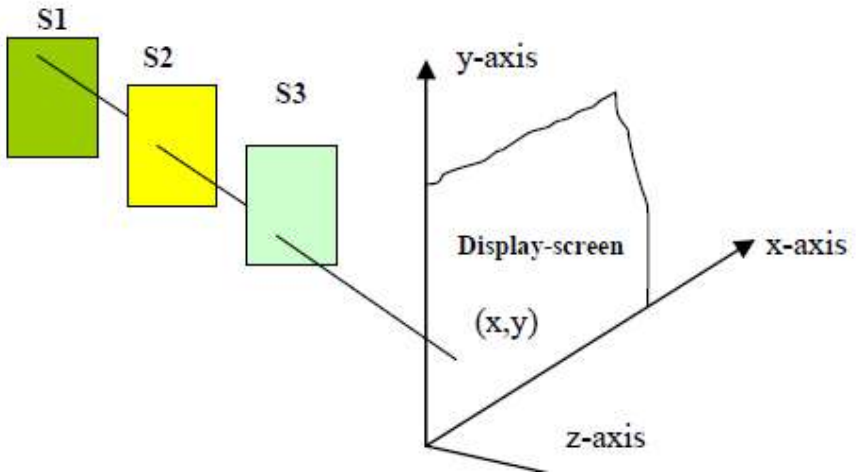
In this method each surface is processed separately one pixel position at a time across the surface. The depth values for a pixel are compared and the closest (smallest z) surface determines the color to be displayed in the frame buffer.

It is applied very efficiently on surfaces of polygon. Surfaces can be processed in any order. To override the closer polygons from the far ones, two buffers named **frame buffer** and **depth buffer,** are used.

Depth buffer is used to store depth values for (x, y) position, as surfaces are processed ($0 \leq$ depth ≤ 1).

The **frame buffer** is used to store the intensity value of color value at each position (x, y).

The z-coordinates are usually normalized to the range [0, 1]. The 0 value for z-coordinate indicates back clipping pane and 1 value for z-coordinates indicates front clipping pane.



Z-Buffer Algorithm

Step-1 − Set the buffer values –

Depthbuffer (x, y) = 0

Framebuffer (x, y) = background color

Step-2 − Process each polygon (One at a time)

For each projected (x, y) pixel position of a polygon, calculate depth z.

If Z > depthbuffer (x, y)

Compute surface color,

set depthbuffer (x, y) = z,

framebuffer (x, y) = surfacecolor (x, y)

Advantages of Z- Buffer Algorithm

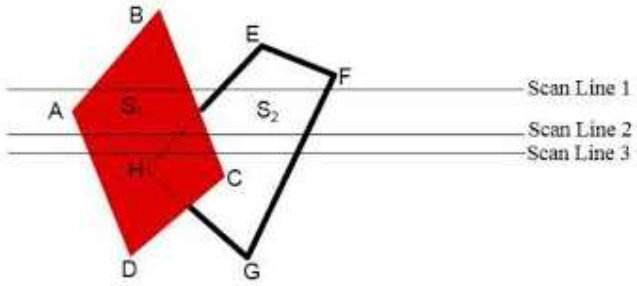
- It is easy to implement.
- It reduces the speed problem if implemented in hardware.
- It processes one object at a time.

Disdvantages of Z- Buffer Algorithm

- It requires large memory.
- It is time consuming process.

5.4.2 Scan-Line Method

It is an image-space method to identify visible surface. This method has a depth information for only single scan-line. In order to require one scan-line of depth values, we must group and process all polygons intersecting a given scan-line at the same time before processing the next scan-line. Two important tables, **edge table** and **polygon table,** are maintained for this.



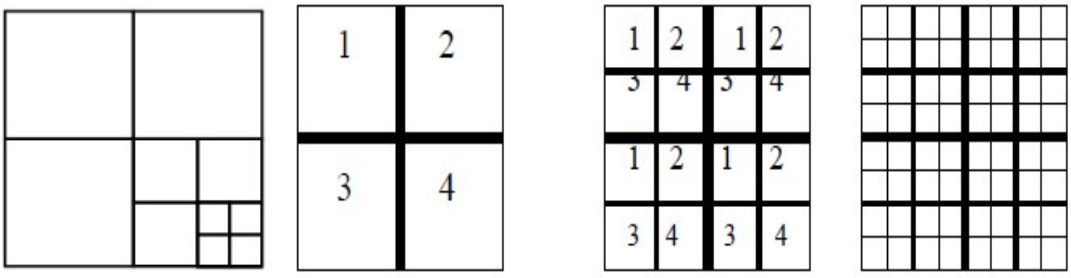
The Edge Table − It contains coordinate endpoints of each line in the scene, the inverse slope of each line, and pointers into the polygon table to connect edges to surfaces.

The Polygon Table − It contains the plane coefficients, surface material properties, other surface data, and may be pointers to the edge table.

To facilitate the search for surfaces crossing a given scan-line, an active list of edges is formed. The active list stores only those edges that cross the scan-line in order of increasing x. Also a flag is set for each surface to indicate whether a position along a scan-line is either inside or outside the surface. Pixel positions across each scan-line are processed from left to right. At the left intersection with a surface, the surface flag is turned on and at the right, the flag is turned off. You only need to perform depth calculations when multiple surfaces have their flags turned on at a certain scan-line position.

5.4.3 Area Subdivison Method

Area-subdivision method is essentially an image-space method but uses object-space calculations for reordering of surfaces according to depth. The method makes use of area coherence in a scene by collecting those areas that form part of a single surface. In this method, we successively subdivide the total viewing area into small rectangles until each small area is the projection of part of a single visible surface or no surface at all.

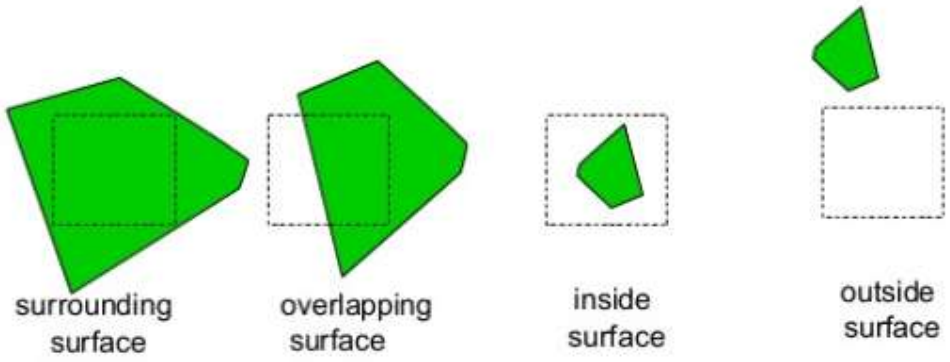


The area-subdivision method takes advantage by locating those view areas that represent part of a single surface. Divide the total viewing area into smaller and smaller rectangles until each small area is the projection of part of a single visible surface or no surface at all.

Continue this process until the subdivisions are easily analyzed as belonging to a single surface or until they are reduced to the size of a single pixel. An easy way to do this is to successively divide the area into four equal parts at each step.

Test to determine the visibility of a single surface are made by comparing surfaces with respect to a given screen area A. There are four possible relationships that a surface can have with a specified area boundary.

- **Surrounding surface** − One that completely encloses the area.
- **Overlapping surface** − One that is partly inside and partly outside the area.
- **Inside surface** − One that is completely inside the area.
- **Outside surface** − One that is completely outside the area.



The tests for determining surface visibility within an area can be stated in terms of these four classifications. No further subdivisions of a specified area are needed if one of the following conditions is true −

- All surfaces are outside surfaces with respect to the area.
- Only one inside, overlapping or surrounding surface is in the area.
- A surrounding surface obscures all other surfaces within the area boundaries.

5.4.4 Back-Face Detection

A fast and simple object-space method for identifying the back faces of a polyhedron is based on the "inside-outside" tests. A point (x, y, z) is "inside" a polygon surface with plane parameters A, B, C, and D if When an inside point is along the line of sight to the surface, the polygon must be a back face (we are inside that face and cannot see the front of it from our viewing position).

We can simplify this test by considering the normal vector **N** to a polygon surface, which has Cartesian components (A, B, C).

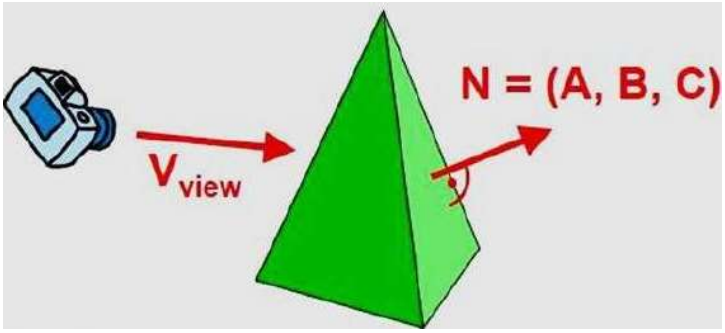
In general, if V is a vector in the viewing direction from the eye (or "camera") position, then this polygon is a back face if

V.N > 0

Furthermore, if object descriptions are converted to projection coordinates and your viewing direction is parallel to the viewing z-axis, then −

$V = (0, 0, V_z)$ and $V.N = V_Z C$

So that we only need to consider the sign of C the component of the normal vector **N**.



In a right-handed viewing system with viewing direction along the negative ZVZVaxis, the polygon is a back face if C < 0. Also, we cannot see any face whose normal has z component C = 0, since your viewing direction is towards that polygon. Thus, in general, we can label any polygon as a back face if its normal vector has a z component value −

C <= 0

Similar methods can be used in packages that employ a left-handed viewing system. In these packages, plane parameters A, B, C and D can be calculated from polygon vertex coordinates specified in a clockwise direction (unlike the counterclockwise direction used in a right-handed system).

Also, back faces have normal vectors that point away from the viewing position and are identified by C >= 0 when the viewing direction is along the positive ZvZvaxis. By examining parameter C for the different planes defining an object, we can immediately identify all the back faces.

5.4.5 A-Buffer Method

The A-buffer method is an extension of the depth-buffer method. The A-buffer method is a visibility detection method developed at Lucas film Studios for the rendering system Renders Everything You Ever Saw (REYES).

The A-buffer expands on the depth buffer method to allow transparencies. The key data structure in the A-buffer is the accumulation buffer.

Each position in the A-buffer has two fields −

- **Depth field** − It stores a positive or negative real number
- **Intensity field** − It stores surface-intensity information or a pointer value

If depth >= 0, the number stored at that position is the depth of a single surface overlapping the corresponding pixel area. The intensity field then stores the RGB components of the surface color at that point and the percent of pixel coverage.

If depth < 0, it indicates multiple-surface contributions to the pixel intensity. The intensity field then stores a pointer to a linked list of surface data. The surface buffer in the A-buffer includes −

- RGB intensity components
- Opacity Parameter
- Depth
- Percent of area coverage
- Surface identifier

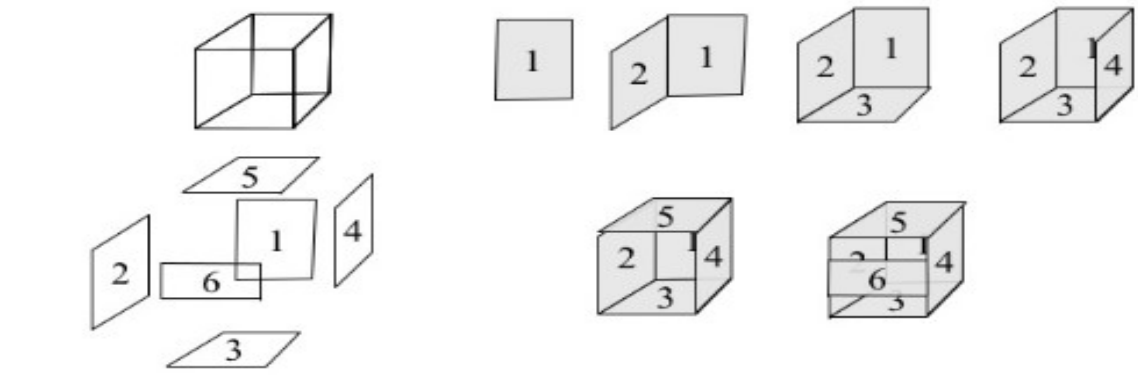
The algorithm proceeds just like the depth buffer algorithm. The depth and opacity values are used to determine the final color of a pixel.

5.4.6 Depth Sorting Method

Depth sorting method uses both image space and object-space operations. The depth-sorting method performs two basic functions −

- First, the surfaces are sorted in order of decreasing depth.
- Second, the surfaces are scan-converted in order, starting with the surface of greatest depth.

The scan conversion of the polygon surfaces is performed in image space. This method for solving the hidden-surface problem is often referred to as the **painter's algorithm**. The following figure shows the effect of depth sorting −



The algorithm begins by sorting by depth. For example, the initial "depth" estimate of a polygon may be taken to be the closest z value of any vertex of the polygon.

Let us take the polygon P at the end of the list. Consider all polygons Q whose z-extents overlap P's. Before drawing P, we make the following tests. If any of the following tests is positive, then we can assume P can be drawn before Q.

- Do the x-extents not overlap?
- Do the y-extents not overlap?
- Is P entirely on the opposite side of Q's plane from the viewpoint?
- Is Q entirely on the same side of P's plane as the viewpoint?
- Do the projections of the polygons not overlap?

If all the tests fail, then we split either P or Q using the plane of the other. The new cut polygons are inserting into the depth order and the process continues. Theoretically, this partitioning could generate $O(n^2)$ individual polygons, but in practice, the number of polygons is much smaller.

5.4 BEZIER CURVES

Curves are trajectories of moving points. We will specify them as functions assigning a location of that moving point (in 2D or 3D) to a parameter t and are useful in geometric modeling.

Some of the advantages of representing the curves, defined by a set of points by mathematical expression is – *precision, compact storage and ease of calculating the intermediate points, the slope and radius of curvature of the curve.*

Any curve can be described by an array of points. One class of mathematical function is particularly suitable for this purpose - the polynomial function:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

Curves can be represented in two forms: Parametric and Implicit.

The *parametric representation*: **x = x(t) ; y = y(t) ; z= z(t)**

The *implicit representations*: **f(x,y)=0 ; s(x,y,z) = 0**

A spline curve is defined, modified, and manipulated with operations on the control points. Spline means 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. 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.

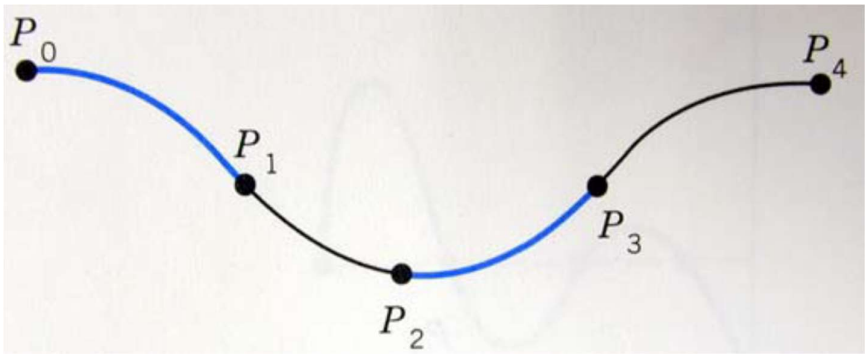


Fig :- Piecewise parametric cubic interpolation

The cubic spline is represented by a piecewise cubic polynomial withsecond order derivative continuity at the common joints between segments.

Parametric continuity:

C0 continuity: no gaps or jumps in a curve

C1 continuity: slope/tangent or first derivative continuity

C2 continuity: curvature or second derivative continuity

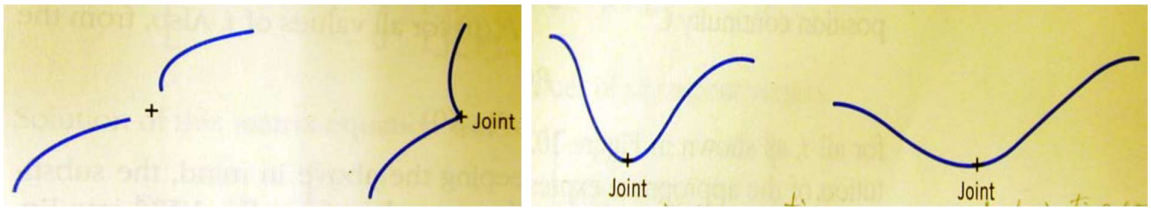


Fig :- Examples of parametric continuity: (a) Discontinuous; (b) C_0; (c) C_1; (d) C_2.

Bezier curves are used in computer graphics to produce curves which appear reasonably smooth at all scales. The spline approximation method was developed by French engineer **Pierre Bezier** for automobile body design. Bezier spline was designed in such a manner that they are very useful and convenient for curve and surface design, and are easy to implement .**The Bezier curve require only two end points and other points that control the endpoint tangent vector.**

Bezier curve is defined by a sequence of N + 1 control points, P_0, P_1,. . . , P_n. We defined the Bezier curve using the algorithm (invented by **DeCasteljeau**), based on recursive splitting of the intervals joining the consecutive control points.

The Bezier curve can be represented mathematically as –

$$\sum_{k=0}^{n} P_i B_i^n(t)$$

Where $\mathbf{P_i}$ is the set of points and $\mathbf{B^n_i(t)}$ represents the Bernstein polynomials which are given by

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

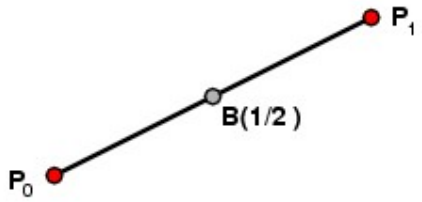
–

Where **n** is the polynomial degree, **i** is the index, and **t** is the variable.

The simplest Bézier curve is the straight line from the point P0P0 to P1P1. A quadratic Bezier curve is determined by three control points. A cubic Bezier curve is determined by four control points.

5.4.1 TYPES OF BEZIER CURVES

The **Simple Bézier curve** is the straight line from one point ***P₀*** to another ***P₁***, with the parametric equation



B(t) = P₀ + t(P₁ - P₀) = (1-t) P₀ + t P₁

from which it follows immediately that

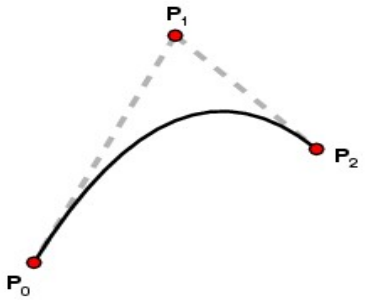
B(0) = P₀

B(1) = P₁ .

For ***t*** in between ***0*** and ***1*** the point ***B(t)*** is ***t*** of the way from one to the other. This is the same as the weighted average of the two points, with ***P₀***given weight ***1-t*** and ***P₁*** given weight ***t***.

When ***t=1/2***, for example, ***B(t)*** is the point ***(1/2)(P₀ + P₁)*** halfway between ***P₀*** and ***P₁***.

A **Quadratic Bézier curve** is determined by three control points P_0, P_1, and P_2. It has the parametric form



$$B(t) = (1-t)^2 P_0 + 2t(1-t) P_1 + t^2 P_2$$

When $t=0$ all but terms but the first vanish, and when $t=1$ all but the last vanish. Therefore

$$B(0) = P_0$$

$$B(1) = P_2$$

A Bezier curve is determined by a defining polygon with the tangent vectors at the ends of the curve having the same direction as the first and last polygon spans respectively and contained within the **convex hull** of the polygon. **The blending function** of the defining polygon for a Bezier curve has **Bernstein basis**, which is global in nature. Because of this, in certain applications, the curve lacks local control.

5.4.2 Properties of Bezier Curves

Bezier curves have the following properties –

Interpolation; Tangency; Convex hull property; Variation diminishing property

They generally follow the shape of the control polygon, which consists of the segments joining the control points.

- They always pass through the first and last control points.
- They are contained in the convex hull of their defining control points.
- The degree of the polynomial defining the curve segment is one less that the number of defining polygon point. Therefore, for 4 control points, the degree of the polynomial is 3, i.e. cubic polynomial.
- A Bezier curve generally follows the shape of the defining polygon.
- The direction of the tangent vector at the end points is same as that of the vector determined by first and last segments.
- The convex hull property for a Bezier curve ensures that the polynomial smoothly follows the control points.
- No straight line intersects a Bezier curve more times than it intersects its control polygon.
- They are invariant under an affine transformation.

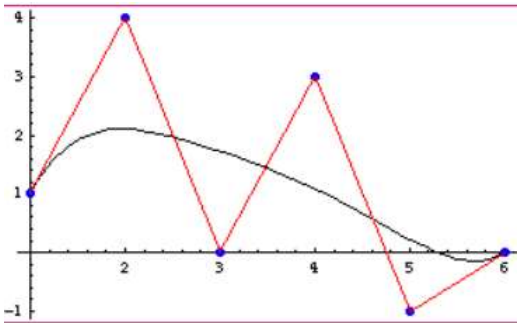
- Bezier curves exhibit global control means moving a control point alters the shape of the whole curve.
- A given Bezier curve can be subdivided at a point t=t0 into two Bezier segments which join together at the point corresponding to the parameter value t=t0.

5.4.3 Numerical Analysis of Bezier Curve

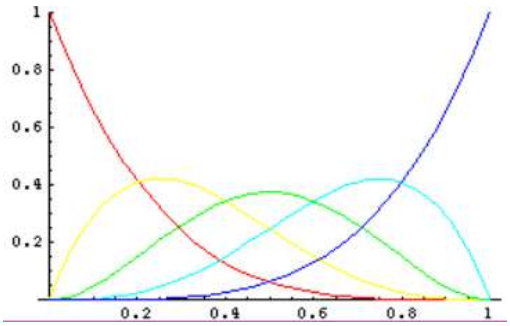
Let $P=\{P_0, P_1, ..., P_n\}$ be a set of points $P_i \in IR^d$, d=2,3. The Bézier curve associated with the set P is defined by:

$$\sum_{i=0}^{n} \mathbf{P}_i B_i^n(t)$$ Where $B^n_i(t)$ represent the Bernstein polynomials, which are given by:

$$B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i \qquad i = 0, \ldots, n$$ **n** being the polynomial degree. **i** is the index. **t** the variable



Bézier curve with n=5 (six control or **Bézier points**)



Bernstein polynomials $\mathbf{B}_i^4(t)$

$$B_{3k} = \frac{3}{t_{k+1}^2}(P_{k+1} - P_k) - \frac{2P_k'}{t_{k+1}} - \frac{2P_{k+1}'}{t_{k+1}}$$

$$B_{4k} = \frac{3}{t_{k+1}^3}(P_k - P_{k+1}) + \frac{P_k'}{t_{k+1}^2} + \frac{P_{k+1}'}{t_{k+1}^2}$$

In the matrix form, the equations for B for any spline segment k are:

$$[B] = \begin{bmatrix} B_{1K} \\ B_{2K} \\ B_{3K} \\ B_{4K} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{3}{t_{k+1}^2} & -\frac{2}{t_{k+1}} & \frac{3}{t_{k+1}^2} & -\frac{1}{t_{k+1}} \\ \frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} & -\frac{2}{t_{k+1}^3} & \frac{1}{t_{k+1}^2} \end{bmatrix} \begin{bmatrix} P_k \\ P_k' \\ P_{k+1} \\ P_{k+1}' \end{bmatrix}$$

Where $\mathrm{Pk}(t) = \sum_{i=1}^{4} \mathrm{Bik(t)i} - 1$

$$\mathrm{P_K(t)} = \begin{bmatrix} 1 & t & t2 & t3 \end{bmatrix} \begin{bmatrix} B1K \\ B2K \\ B3K \\ B4K \end{bmatrix}$$

LOCAL vs. GLOBAL CONTROL

Bézier curves exhibit global control: It involves moving a control point alters the shape of the whole curve.

B-splines allow local control: In B-splines , only a part of the curve is modified when changing a control point.

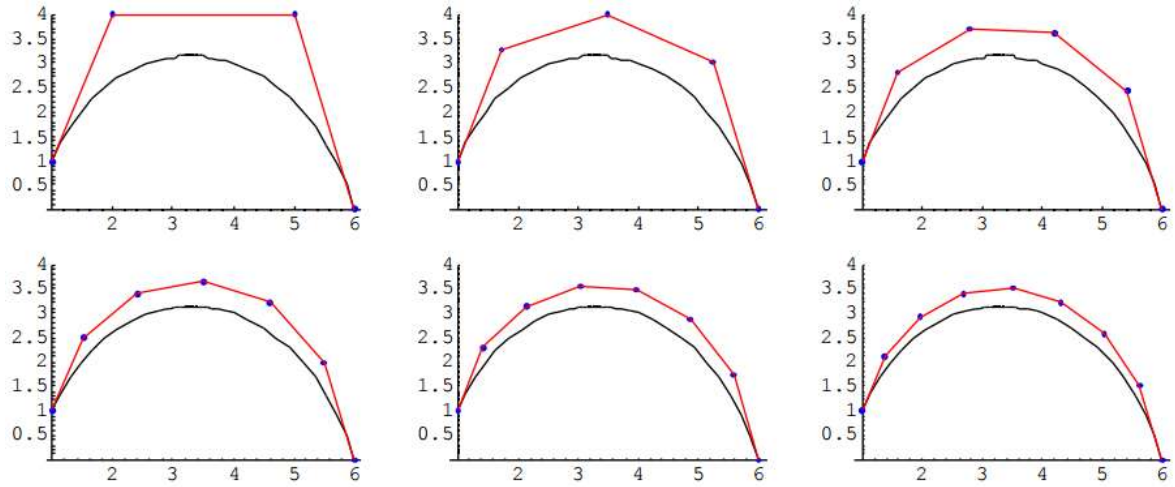


Fig : Degree raising of Bezier Curve of degree n=3 to degree n=8

5.4.4 B-Spline Curves

The Bezier-curve produced by the Bernstein basis function has limited flexibility.

- First, the number of specified polygon vertices fixes the order of the resulting polynomial which defines the curve.
- The second limiting characteristic is that the value of the blending function is nonzero for all parameter values over the entire curve.

The B-spline basis contains the Bernstein basis as the special case. The B-spline basis is non-global.

Properties of B-spline Curve

B-spline curves have the following properties −

- The sum of the B-spline basis functions for any parameter value is 1.
- Each basis function is positive or zero for all parameter values.
- Each basis function has precisely one maximum value, except for k=1.
- The maximum order of the curve is equal to the number of vertices of defining polygon.
- The degree of B-spline polynomial is independent on the number of vertices of defining polygon.
- B-spline allows the local control over the curve surface because each vertex affects the shape of a curve only over a range of parameter values where its associated basis function is nonzero.
- The curve exhibits the variation diminishing property.
- The curve generally follows the shape of defining polygon.
- Any affine transformation can be applied to the curve by applying it to the vertices of defining polygon.
- The curve line within the convex hull of its defining polygon.

5.5 SUMMARY

This unit emphasizes that for displaying a realistic view of the given 3D-object, hidden surfaces and hidden lines must be identified for elimination.

• The process of identifying and removal of these hidden surfaces is called the visible-line or visible-surface determination, or hidden-line or hidden-surface elimination.

• To construct a realistic view of the given 3D object, it is necessary to determine which lines or surfaces of the objects are visible. For this, we need to conduct visibility tests.

• Visibility tests are conducted to determine the surface that is visible from a given viewpoint.. These two approaches are called image-space approach and object-space approach, respectively.

5.6 QUESTIONS FOR EXERCISE

1) What are the different techniques of hidden surface removal ?

2) What are the conditions to be satisfied, in Area-subdivision method, so that a surface not to be divided further?

3) What is the role of Depth-buffer and Refresh-buffer in Z-buffer implementation?

4) Based on the Bezier curve definition, derive the equation of the 3 point Bezier curve defined by the following control points. (-1,0), (0,2), and (1,0).

4.10 SUGGESTED READING

Example 1: How does the z-buffer algorithm determine which surfaces are hidden? What is the maximum number of objects that can be handled by the zbuffer algorithm?

What happens when two polygons have the same z value and the z-buffer algorithm is used?

Distinguish between z-buffer method and scan-line method. What are the visibility test made in these methods?

What are the relative merits of object-space methods and image-space methods?

UNIT-6 ILLUMINATION, ANIMATION AND VISUALIZATION

UNIT STRUCTURE

23.0
24.0 describe the basic properties of animation;
25.0 • classify the animation and its types;
26.0 • discuss how to impart acceleration in animation, and
27.0 • give examples of different animation tools and applications.

6.1 INTRODUCTION TO ILLUMINATION

From Physics we can derive models, called "illumination models", of how light reflects from surfaces and produces what we perceive as color. In general, light leaves some light source, e.g. a lamp or the sun, is reflected from many surfaces and then finally reflected to our eyes, or through an image plane of a camera. Indirect light is all the inter-reflected light in a scene. We will first look at some basic properties of light and color, the physics of light-surface interactions, local illumination models, and global illumination models.

27.1.1 LOCAL VS. GLOBAL ILLUMINATION

The contribution from the light that goes directly from the light source and is reflected from the surface is called a "local illumination model". So, for a local illumination model, the shading of any surface is independent from the shading of all other surfaces. Local illumination is only the light provided directly from a light source (such as a spot light).
Direct light is emitted from a light source and travels in a straight path to the illuminated point (either on a surface or in a volume).
With direct illumination only each light source's contribution is used to calculate the overall light contribution to any given illuminated point.
Examples:

- A spot light illuminates an actor on stage
- Sunlight shines directly on sunbathers

A "global illumination model" adds to the local model the light that is reflected from other surfaces to the current surface. Global illumination is an approximation of real-world indirect light transmission.
With global illumination, the contribution of bounced light from other surfaces in the scene is used to calculate the overall light contribution and the color values at points on objects that are not directly illuminated (that is, at points that do not receive light directly from a light source, such as a spot light).

Global illumination occurs when light is reflected off of or transmitted through an opaque (reflection only), transparent or semi-transparent surface (see Diffuse, Specular, and Glossy refraction of light) from a surface to bounce off or be absorbed by another surface.

A global illumination model is more comprehensive, more physically correct, and produces more realistic images. It is also more computationally expensive.

Examples:

- A crack at the bottom of a door can cause light to spill into a room.
- White walls reflect light from the light source to another surface in a room.
- A body of water can transmit light from its surface to the floor.

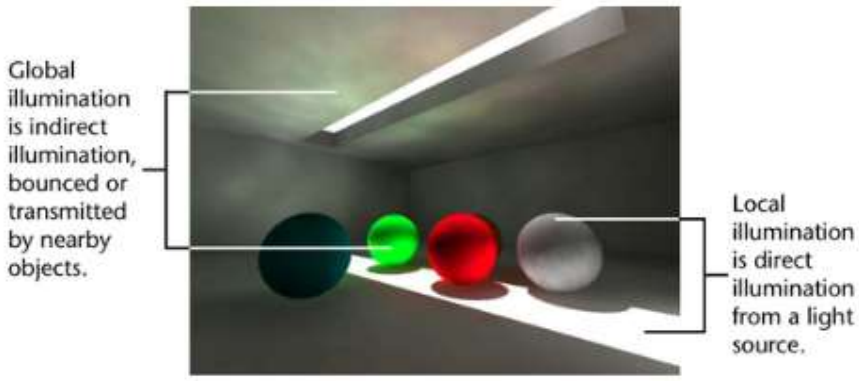


Fig :

Scan-line rendering methods use only local illumination models, although they may use tricks to simulate global illumination. Many current graphics images and commercial systems are in this category, but many systems are becoming global illumination based.

The two major types of graphics systems that use global illumination models are radiosity and ray tracing. These produce more realistic images but are more computationally intensive than scan-line rendering systems.

6.2 DIRECT SOURCES OF LIGHT

Every object in a scene is potentially a source of light. Light may be either be emitted or reflected from objects. Generally, in computer graphics we make a distinction between light emitters and light reflectors. The emitters are called ***light sources***, and the reflectors are usually the objects being ***rendered***. Light sources are

characterized by their intensities while reflectors are characterized by their material properties of :-

- Emittance Spectrum (colour)
- Geometry (position and direction)
- Directional Attenuation

Most computer graphic rendering systems only attempt to model the direct illumination from the emitters to the reflectors of the scene. On the other hand most systems ignore the geometry of light emitters, and consider only the geometry of reflectors. The rationalization behind these simplifications is that most of the light from a scene results from a single bounce of a emitted ray off of a reflective surface. In most computer generated pictures you will not see light directly emitted from the light source, nor the indirect illumination from a light reflecting off on surface and illuminating another.

Direct Lighting

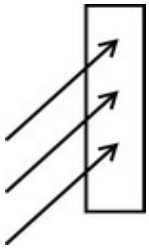
Indirect Lighting

Combined Direct and Indirect Lighting to render full Scene

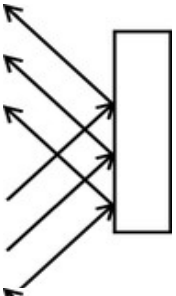
6.2.1 ABSORPTION, REFLECTION, REFRACTION OF LIGHT

The color of the objects we see in the natural world is a result of the way objects interact with light. When a light wave strikes an object, it can be absorbed, reflected, or refracted by the object. All objects have a degree of reflection and absorption.

In the natural world, light can also be transmitted by an object. That is, light can pass through an object with no effect (an x-ray, for example).

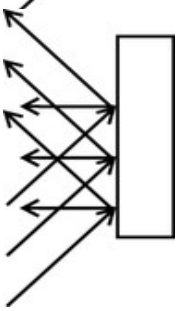


In **Absorption**; light stops at the object and does not reflect or refract. Objects appear dark or opaque. Example: **wood.**



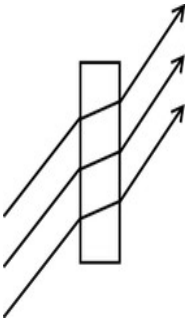
Reflection on a smooth surface : Light bounces off the surface of a material at an angle equal to the angle of the incoming light wave.

Example**: mirrors or glass.**



Scatter (Reflection on a rough surface) : Light waves bounce off at many of angles because the surface is uneven.

Example:**The earth** (that's why the sky is blue).



Refraction : Light goes through the object and bends at an angle.

Example: **Diamond** (greater angle) or **Water** (lesser angle)

6.2.2 DIRECTIONAL,SPOT, POINT AND AMBIENT LIGHT

Directional Light :

We use a directional light to simulate a very distant point light source (for example, the sun as viewed from the surface of the Earth).

A directional light shines evenly in one direction only. Its light rays are parallel to each other, as if emitted perpendicular from an infinitely large plane.

Ambient Light :

Ambient light is the illumination of an object caused by reflected light from other surfaces. To calculate this exactly would be very complicated. A simple model assumes ambient light is uniform in the environment.

Even though an object in a scene is not directly lit it will still be visible. This is because light is reflected indirectly from nearby objects. A simple hack that is commonly used to model this indirect illumination is to use of an ambient light source.

An ambient light shines in two ways—some of the light shines evenly in all directions from the location of the light (similar to a point light), and some of the light shines evenly in all directions from all directions (as if emitted from the inner surface of an infinitely large sphere).

Use an ambient light to simulate a combination of direct light (for example, a lamp) and indirect light (lamp light reflected off the walls of a room).Its characteristics are :-

- ***The amount of ambient light incident on each object is a constant for all surfaces in the scene.***
- ***An ambient light can have a color.***
- ***The amount of ambient light that is reflected by an object is independent of the object's position or orientation.***
- ***Surface properties are used to determine how much ambient light is reflected.***

Spot Light

A spot light shines a beam of light evenly within a narrow range of directions that are defined by a cone. The rotation of the spot light determines where the beam is aimed. The width of the cone determines how narrow or broad the beam of light is. You can adjust the softness of the light to create or eliminate the harsh circle of projected light. You can also project image maps from spot lights. We use a spot light to create a beam of light that gradually becomes wider (for example, a flashlight or car headlight).

Point Light

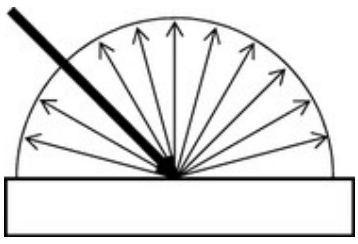
A point light shines evenly in all directions from an infinitely small point in space. Use a point light to simulate an incandescent light bulb or a star.

6.2.3 TYPES OF REFLECTION

Reflection is divided into three types: diffuse, specular, and glossy.

We consider the direction of the light source when computing both the diffuse and specular components of illumination. With a directional light source this direction is a constant.

Diffuse reflection

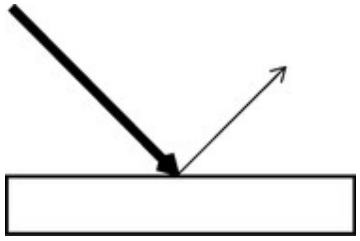


Diffuse surfaces reflect (scatter) light in many angles.

Most objects around us do not emit light of their own. Rather they absorb daylight, or light emitted from an artificial source, and reflect part of it. Here, light that reached the surfaces would be scattered equally in all directions. This implies that the amount of light as observed by the viewer is independent of the viewer's location.

Diffuse reflection accounts for more of the color than any other type of distribution because most objects are opaque and reflect light diffusely.

Specular reflection : Shiny surface : Very smooth surface

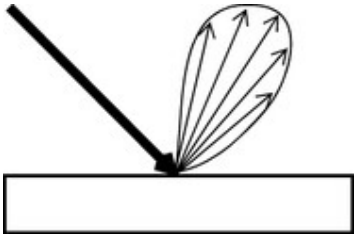


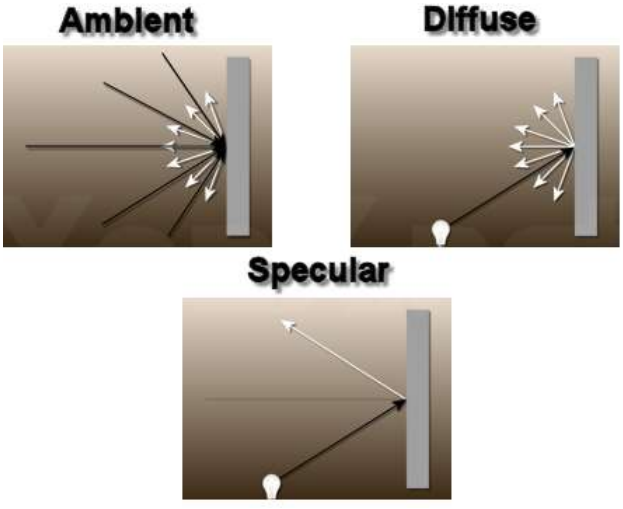
A second surface type is called a specular reflector. When we look at a shiny surface, such as polished metal or a glossy car finish, we see a highlight, or bright spot. Whe re this bright spot appears on the surface is a function of where the surface is seen from. This type of reflectance is view dependent. Specular surfaces reflect light at the same as the angle at which the light strikes the surface.

At the microscopic level a specular reflecting surface is very smooth, and usually these microscopic surface elements are oriented in the same direction as the surface itself. Specular reflection is merely the mirror reflection of the light source in a surface. Thus it should come as no surprise that it is viewer dependent, since if you stood in front of a mirror and placed your finger over the refelection of a light, you would expect that you could reposition your head to look around your finger and see the light again. An ideal mirror is a purely specular reflector.

Glossy Reflection :-

Glossy surfaces are actually specular surfaces with micro surfaces at angles to surface plane. These micro surfaces reflect light not only specularly but also diffusely (at angles very close to the specular transmission), giving the surface a glossy appearance.





6.3 TYPES OF SHADING

In 3D graphics, a shading technique to compute a shaded surface based on the color and illumination at the corners of every triangle was developed. There are three traditional shading models, namely flat shading, Gouraud shading and Phong shading. Global illumination shading models such as recursive ray tracing and radiosity takes into account the interchange of light between all surfaces.

Gouraud shading is the simplest rendering method and is computed faster than Phong shading. It does not produce shadows or reflections. The surface normals at the triangle's points are used to create RGB values, which are averaged across the triangle's surface.

Flat Shading :-

Flat surface rendering or constant shading is the simplest rendering format that involves some basic surface properties such as colour distinctions and reflectivity. This method produces a rendering that does not smooth over the faces which make up the surface. The resulting visualization shows an object that appears to have surfaces faceted like a diamond.

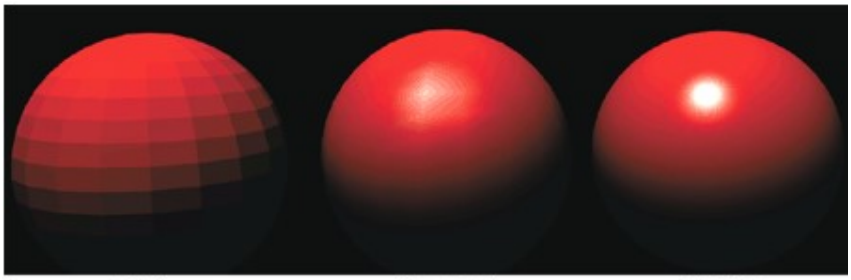
Rendering only requires the computation of a colour for each visible face. The whole face is filled with this colour.

Gouraud Shading :-

Named after its inventor, Henri Gouraud who developed this technique in 1971 (yes, 1971). It is by far the most common type of shading used in consumer 3D graphics hardware, primarily because of its higher visual quality versus its still-modest computational demands. This technique takes the lighting values at each of a triangle's three vertices, then interpolates those values across the surface of the triangle. Gouraud shading actually first interpolates between vertices and assigns values along triangle edges, then it interpolates across the scan line based on the interpolated edge crossing values. One of the main advantages to Gouraud is that it smoothes out triangle edges on mesh surfaces, giving objects a more realistic appearance. The disadvantage to Gouraud is that its overall effect suffers on lower triangle-count models, because with fewer vertices, shading detail (specifically peaks and valleys in the intensity) is lost. Additionally, Gouraud shading sometimes loses highlight detail, and fails to capture spotlight effects.

The Gouraud Shading method applies the illumination model on a subset of surface points and interpolates the intensity of the remaining points on the surface.

In the case of a polygonal mesh the illumination model is usually applied at each vertex and the colors in the triangles interior are linearly interpolated from these vertex values.



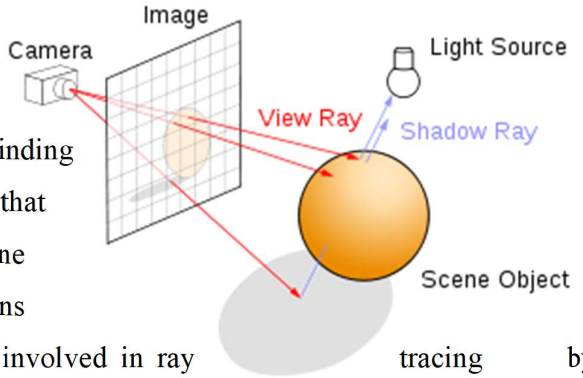
Phong Shading

It was developed in 1975 by Phong Biu-Tuong in which the surface normal is linearly interpolated across polygonal facets.

A Phong shader assumes the same input as a Gouraud shader, which means that it expects a normal for every vertex. The illumination model is applied at every point on the surface being rendered, where the normal at each point is the result of linearly interpolating the vertex normals defined at each vertex of the triangle. Phong shading will usually result in a very smooth appearance, however, evidence of the polygonal model can usually be seen along silhouettes.

Phong shading overcomes the limitation of Gouraud shading by incorporating specular reflection into the scheme.

6.4 RAY TRACING



"Ray Tracing" is a method of following the light from the eye to the light source.
Whereas ray casting only concerns itself with finding the visible surfaces of objects, ray tracing takes that a few steps further and actually tries to determine will cost your processor time spent in calculations you can understand the level of calculations involved in ray tracing by considering this example, Let's say we are rendering (that is, ray tracing) a scene at a resolution of 320 pixels wide by 240 pixels high, for a total of 76,800 pixels. Let it be of low complexity, with only 20 objects. That means, over the course of creating this picture, the ray tracer will have done 20 intersection tests for each of those 76,800 pixels, for a total of 1,536,000 intersection tests Ray tracing allows you to create several kinds of effects that are very difficult or even impossible to do with other methods. These effects include three items common to every ray tracer: reflection, transparency, and shadows.

Applications of Ray Tracing

Ray tracing finds a varied area of it applications. A few are as follows :-

• simulation of real-world phenomena for vision research,

• medical (radiation treatment planning),

• seismic (density calculations along a ray),

• mechanical engineering (interference checking),

• plant design (pipeline interference checking),

• hit-testing in geometric applications, and impact and penetration studies

• widely used in entertainment.

Computer animation is the use of computers to create animations. Motion can bring the simplest of characters to life. Even simple polygonal shapes can convey a number of human qualities when animated: identity, character, gender, mood, intention, emotion. A movie is a sequence of frames of still images. For video, the frame rate is typically 24 frames per second. For film, this is 30 frames per second.

There are a few different ways to make computer animations. One is 3D animation. One way to create computer animations is to create objects and then render them. This method produces perfect and three dimensional looking animations. Another way to create computer animation is to use standard computer painting tools and to paint single frames and composite them. These can later be either saved as a movie file or output to video. One last method of making computer animations is to use transitions and other special effects like morphing to modify existing images and video. Computer graphics are any types of images created using any kind of computer. There is a vast amount of types of images a computer can create. Also, there are just as many ways of creating those images. Images created by computers can be very simple, such as lines and circles, or extremly complex such as fractals and complicated rendered animations

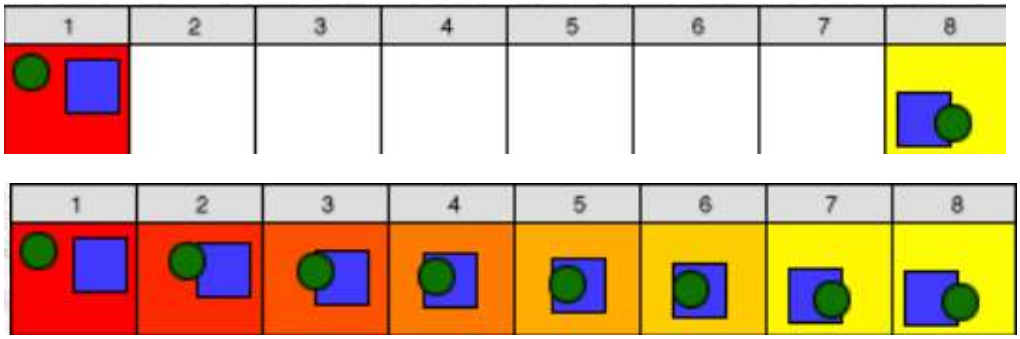
In general, animation may be achieved by specifying a model with n parameters that identify degrees of freedom that an animator may be interested in such as • polygon vertices, • spline control, • joint angles, • muscle contraction, • camera parameters

Modeling and animation are loosely coupled. Modeling describes control values and their actions. Animation describes how to vary the control values. There are a number of animation techniques,

Anti-aliasing is the process of bluring sharp edges in pictures to get rid of the jagged edges on lines. After an image is rendered, some applications automatically anti-alias images. The program looks for edges in an image, and then blurs adjacent pixels to produce a smoother edge. In order to anti-alias an image when rendering, the computer has to take samples smaller than a pixel in order to figure out exactly

where to blur and where not to. For example, if the computer finds that one pixel is on the edge of thwo objects, it then takes sub-pixel samples and checks about how many of them showed the fron obejct, and how many showed the back one. Lets say that the computer took 8 sub-samples, and 4 of them were on object one and the other 4 on object two. The computer then takes the resulting color values from the subsamples and averages them into a resulting blurred pixel, when viewed from a distance gives a smoother edge effect. In the below example, you can see an image without anti-aliasing and with anti-aliasing and enlargements of these two pictures.

When someone creates a 3D animation on a computer, they usually don't specify the exact position of any given object on every single frame. 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.



What will be the colour of a blue rose when it is viewed in red light? Give reasons in support of your answer. ……
……
…… 3) If the source of light is very far from the object what type of rays you expect from the source? What will happen to the type of rays if source is quite close to the object?

If no variation in the intensity of reflection light is observed in any direction, then what can you say about the smoothness of the surface? Also specify what type of reflection you expect from such surface.

What are merits & demerits of Ground shading and Phong shading?

How Ambient, Diffused and Specular reflection contributes to the resulting intensity of reflected ray of light?

APPENDIX -A

LAB PROGRAMS OF COMPUTER GRAPHICS IN C LANGUAGE

EXPERIMENT NO : 1 BRESENHAM'S LINE ALGORITHM

Aim : To write a C program to draw a line using Bresenham's Algorithm.

ALGORITHM :

Step 1 : Start.
Step 2 : Initialize the graphics header files and functions.
Step 3 : Declare the required variables and functions.
Step 4 : Get the four points for drawing a line namely x1,x2,y1,y2.
Step 5 : Draw the line using the algorithm.
Step 6 : Display the output.
Step 7 : stop.

PROGRAM : else

```
#include "stdio.h"
#include "conio.h"
#include "math.h"
#include "graphics.h"

main()
{

   int gd=DETECT,gm;
   int xa,xb,ya,yb;
   int dx,dy,x,y,xend,p;
   initgraph(&gd,&gm,"c:\\tc\\bgi");
   printf("Enter The Two Left
    endpoints(xa,ya):\n");
   scanf("%d%d",&xa,&ya);
   printf("Enter The Two Right
   endpoints(xb,yb):\n");
   scanf("%d%d",&xb,&yb);
   dx=abs(xa-xb);
   dy=abs(ya-yb);
   p=2*dy-dx;
   if(xa>xb)
   {
     x=xb;
     y=yb;
     xend=xa;

   }
{
   x=xa;
    y=ya;
    xend=xb;
}

    putpixel(x,y,6);
   while(x<xend)

    {
    x=x+1;
    if(p<0)
    {
    p=p+2*dy;
    }

    else
    {
    y=y+1;
    p=p+2*(dy-dx);
    }
    putpixel(x,y,6);
    }
  getch();
  return(0);
}
```

EXPERIMENT NO : 2 BRESENHAM'S CIRCLE ALGORITHM

Aim : Bresenham's Circle Drawing Algorithm Using C Programming :-

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
int gd=DETECT,gm;
int x,y,r;
void cir(int,int,int);
printf("Enter the Mid points and Radious:");
scanf("%d%d%d",&x,&y,&r);
initgraph(&gd,&gm,"");
cir(x,y,r);
getch();
closegraph();
}
void cir(int x1,int y1,int r)
{
int x=0,y=r,p=1-r;
void cliplot(int,int,int,int);
```

```
cliplot(x1,y1,x,y);
while(x<y)
{
x++;
if(p<0)
p+=2*x+1;
else
{
y--;
p+=2*(x-y)+1;
}
cliplot(x1,y1,x,y);
}
}
void cliplot(int xctr,int yctr,int x,int y)
{
putpixel(xctr +x,yctr +y,1);
putpixel(xctr -x,yctr +y,1);
putpixel(xctr +x,yctr -y,1);
putpixel(xctr -x,yctr -y,1);
putpixel(xctr +y,yctr +x,1);
putpixel(xctr -y,yctr +x,1);
putpixel(xctr +y,yctr -x,1);
putpixel(xctr -y,yctr -x,1);
getch();
}
```

EXPERIMENT NO : 3 - POLYGON FILLING ALGORITHM

3A :-- Aim : To implement FloodFill Algorithm Using C Programming :-

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

void flood_fill(int x, int y, int ncolor, int ocolor)
{
if (getpixel(x, y) == ocolor) {
putpixel(x, y, ncolor);
flood_fill(x + 1, y, ncolor, ocolor);
flood_fill(x + 1, y - 1, ncolor, ocolor);
flood_fill(x + 1, y + 1, ncolor, ocolor);
flood_fill(x, y - 1, ncolor, ocolor);
flood_fill(x, y + 1, ncolor, ocolor);
flood_fill(x - 1, y, ncolor, ocolor);
flood_fill(x - 1, y - 1, ncolor, ocolor);
flood_fill(x - 1, y + 1, ncolor, ocolor);
}
}
```

```
void main()
{
int x, y, ncolor, ocolor;
clrscr();
printf("Enter the seed point (x,y): ");
scanf("%d%d", &x, &y);
printf("Enter old color : ");
scanf("%d", &ocolor);
printf("Enter new color : ");
scanf("%d", &ncolor);
int gd = DETECT, gm = DETECT;
initgraph(&gd, &gm, "");
cleardevice();
circle(300, 200, 50);
flood_fill(x, y, ncolor, ocolor);
getch();
}
```

3B ---- Aim : To implement BoundaryFill Algorithm Using C Programming :-

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void boundary_fill(int x, int y, int fcolor, int bcolor)
{
if ((getpixel(x, y) != fcolor) && (getpixel(x, y) != bcolor)) {
putpixel(x, y, fcolor);
boundary_fill(x + 1, y, fcolor, bcolor);
boundary_fill(x - 1, y, fcolor, bcolor);
boundary_fill(x, y - 1, fcolor, bcolor);
boundary_fill(x, y + 1, fcolor, bcolor);
boundary_fill(x + 1, y - 1, fcolor, bcolor);
boundary_fill(x + 1, y + 1, fcolor, bcolor);
boundary_fill(x - 1, y - 1, fcolor, bcolor);
boundary_fill(x - 1, y + 1, fcolor, bcolor);
}
}
void main()
{
int x, y, fcolor, bcolor;
clrscr();
printf("Enter the seed point (x,y) : ");
scanf("%d%d", &x, &y);
printf("Enter boundary color : ");
scanf("%d", &bcolor);
printf("Enter new color : ");
scanf("%d", &fcolor);
int gd = DETECT, gm = DETECT;
initgraph(&gd, &gm, "");
```

```
cleardevice();
boundary_fill(x, y, fcolor, bcolor);
getch();
}
```

EXPERIMENT NO : 4 - TRANSFORMATION ALGORITHMS

4A :- Aim : To implement 2D Translation Program Using C Programming

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

int x1,y1,x2,y2,x3,y3,mx,my;
void draw();
void tri();

void main()
{
   int gd=DETECT,gm; int c;
   initgraph(&gd,&gm,"d:\\tc\\bgi ");
   printf("Enter the 1st point for the
   triangle:"); scanf("%d%d",&x1,&y1);
   printf("Enter the 2nd point for the
   triangle:"); scanf("%d%d",&x2,&y2);
```

```
   printf("Enter the 3rd point for the
   triangle:"); scanf("%d%d",&x3,&y3);
    cleardevice();
   draw();
   getch();
   tri();
   getch();
}

void draw()
{
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
}
void tri()
{
   int x,y,a1,a2,a3,b1,b2,b3;
   printf("Enter the Transaction
   coordinates"); scanf("%d%d",&x,&y);
   cleardevice();
   a1=x1+x;
   b1=y1+y;
   a2=x2+x;
   b2=y2+y;
   a3=x3+x;
   b3=y3+y;
   line(a1,b1,a2,b2);
   line(a2,b2,a3,b3);
   line(a3,b3,a1,b1);
}
```

4B :- Aim : To implement 2D Scaling Program Using C Programming

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

int x1,y1,x2,y2,x3,y3,mx,my;
void draw();
void scale();

void main()
{
   int gd=DETECT,gm;
   int c;
   initgraph(&gd,&gm," ");
   printf("Enter the 1st point for the triangle:");
   scanf("%d%d",&x1,&y1);
   printf("Enter the 2nd point for the triangle:");
   scanf("%d%d",&x2,&y2);
```

```
  printf("Enter the 3rd point for the triangle:");
  scanf("%d%d",&x3,&y3);
  draw();
  scale();
}

void draw()
{
  line(x1,y1,x2,y2);
  line(x2,y2,x3,y3);
  line(x3,y3,x1,y1);
}
void scale()
{
  int x,y,a1,a2,a3,b1,b2,b3;
  int mx,my;
  printf("Enter the scalling coordinates");
  scanf("%d%d",&x,&y);
  mx=(x1+x2+x3)/3; my=(y1+y2+y3)/3;
  cleardevice();
  a1=mx+(x1-mx)*x;
  b1=my+(y1-my)*y;
  a2=mx+(x2-mx)*x;
  b2=my+(y2-my)*y;
  a3=mx+(x3-mx)*x;
  b3=my+(y3-my)*y;
  line(a1,b1,a2,b2);
  line(a2,b2,a3,b3);
  line(a3,b3,a1,b1);
  draw();
  getch();
}
```

4C:- Aim : To implement 2D Rotation Program Using C Programming

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

void TriAngle(int x1,int y1,int x2,int y2,int x3,int y3);
void Rotate(int x1,int y1,int x2,int y2,int x3,int y3); void
main()
{
   int gd=DETECT,gm; int
   x1,y1,x2,y2,x3,y3;
   initgraph(&gd,&gm," ");
    printf("Enter the 1st point for the triangle:");
   scanf("%d%d",&x1,&y1);
   printf("Enter the 2nd point for the triangle:");
   scanf("%d%d",&x2,&y2);
```

```
    printf("Enter the 3rd point for the triangle:");
    scanf("%d%d",&x3,&y3);
    TriAngle(x1,y1,x2,y2,x3,y3);
    getch();
    cleardevice();
    Rotate(x1,y1,x2,y2,x3,y3);
     setcolor(1);
    TriAngle(x1,y1,x2,y2,x3,y3);
    getch();
}

 void TriAngle(int x1,int y1,int x2,int y2,int x3,int y3)
{
   line(x1,y1,x2,y2);
   line(x2,y2,x3,y3);
   line(x3,y3,x1,y1);
}

 void Rotate(int x1,int y1,int x2,int y2,int x3,int y3)
{
    int x,y,a1,b1,a2,b2,a3,b3,p=x2,q=y2;
    float Angle;
    printf("Enter the angle for rotation:");
    scanf("%f",&Angle);
    cleardevice();
    Angle=(Angle*3.14)/180;
    a1=p+(x1-p)*cos(Angle)-(y1-q)*sin(Angle);
    b1=q+(x1-p)*sin(Angle)+(y1-q)*cos(Angle);
    a2=p+(x2-p)*cos(Angle)-(y2-q)*sin(Angle);
    b2=q+(x2-p)*sin(Angle)+(y2-q)*cos(Angle);
    a3=p+(x3-p)*cos(Angle)-(y3-q)*sin(Angle);
    b3=q+(x3-p)*sin(Angle)+(y3-q)*cos(Angle);
    printf("Rotate"); TriAngle(a1,b1,a2,b2,a3,b3);
}
```

EXPERIMENT NO : 5 - CLIPPING ALGORITHMS

Aim :- Implementation of Cohen-Sutherland Line Clipping Algorithm

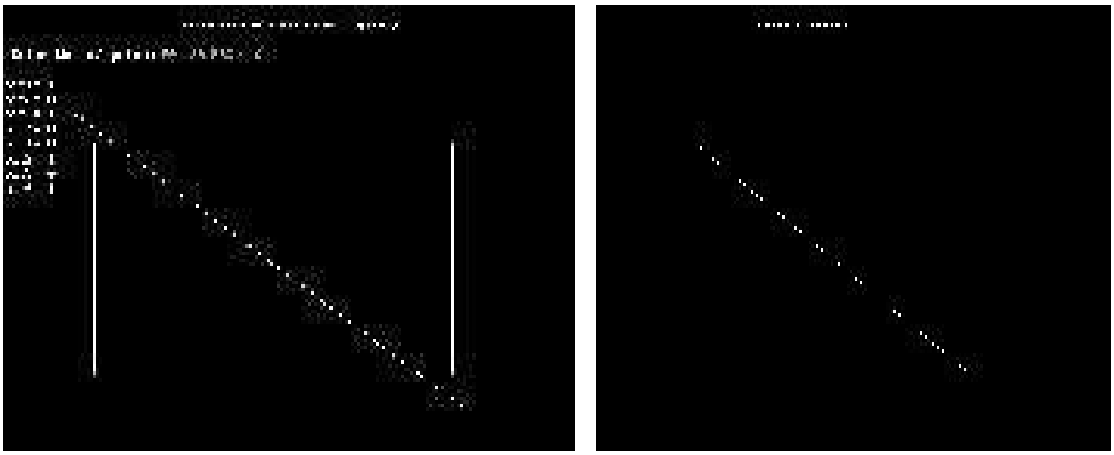
```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void clip(float,float,float);
int i,j=0,n;
int rx1,rx2,ry1,ry2;
float x1[8],y1[8];
void main()
{
int gd=DETECT,gm;
int i,n;
float x[8],y[8],m;
clrscr();
```

```
initgraph(&gd,&gm,"");
printf("coordinates for rectangle : ");
scanf("%d%d%d%d",&rx1,&ry1,&rx2,&ry2);
printf("no. of sides for polygon : ");
scanf("%d",&n);
printf("coordinates : ");
for(i=0;i<n;i++)
{
scanf("%f%f",&x[i],&y[i]);
}
cleardevice();
outtextxy(10,10,"Before clipping");
outtextxy(10,470,"Press any key....");
rectangle(rx1,ry1,rx2,ry2);
for(i=0;i<n-1;i++)
line(x[i],y[i],x[i+1],y[i+1]);
line(x[i],y[i],x[0],y[0]);
getch();
cleardevice();
for(i=0;i<n-1;i++)
{
m=(y[i+1]-y[i])/(x[i+1]-x[i]);
clip(x[i],y[i],m);
}
clip(x[0],y[0],m);
outtextxy(10,10,"After clipping");
outtextxy(10,470,"Press any key....");
rectangle(rx1,ry1,rx2,ry2);
for(i=0;i<j-1;i++)
line(x1[i],y1[i],x1[i+1],y1[i+1]);
getch();
}


void clip(float e,float f,float m)
{
while(e<rx1 e>rx2 f<ry1 f>ry2)
{
if(e<rx1)
{
f+=m*(rx1-e);
e=rx1;
}
else if(e>rx2)
{
f+=m*(rx2-e);
e=rx1;
}
if(f<ry1)
{
e+=(ry1-f)/m;
f=ry1;
}
```

```
else if(f>ry2)
{
e+=(ry2-f)/m;
f=ry2;
}
x1[j]=e;
y1[j]=f;
j++;
}
}
```

OUTPUT:



EXPERIMENT NO : 6 - Bezier Curve Generation

Aim :- C program to implement Bezier Curve Drawing Algorithm :-

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
int x,y,z;
void main()
{
float u;
int gd,gm,ymax,i,n,c[4][3];
for(i=0;i<4;i++) { c[i][0]=0; c[i][1]=0; }
printf("\n\n Enter four points : \n\n");
for(i=0; i<4; i++)
{
printf("\t X%d Y%d : ",i,i);
scanf("%d %d",&c[i][0],&c[i][1]);
}
c[4][0]=c[0][0];
c[4][1]=c[0][1];
detectgraph(&gd,&gm);
initgraph(&gd,&gm,"e:\\tc\\bgi");
ymax = 480;
setcolor(13);
for(i=0;i<3;i++)
{
line(c[i][0],ymax-c[i][1],c[i+1][0],ymax-c[i+1][1]);
}
setcolor(3);
n=3;
for(i=0;i<=40;i++)
{
u=(float)i/40.0;
bezier(u,n,c);
if(i==0)
{ moveto(x,ymax-y);}
else
{ lineto(x,ymax-y); }
getch();
}
getch();
}
bezier(u,n,p)
float u;int n; int p[4][3];
{
int j;
float v,b;
float blend(int,int,float);
x=0;y=0;z=0;
for(j=0;j<=n;j++)
{
b=blend(j,n,u);
```

```
x=x+(p[j][0]*b);
y=y+(p[j][1]*b);
z=z+(p[j][2]*b);
}
}
float blend(int j,int n,float u)
{
int k;
float v,blend;
v=C(n,j);
for(k=0;k<j;k++)
{ v*=u; }
for(k=1;k<=(n-j);k++)
{ v *= (1-u); }
blend=v;
return(blend);
}
C(int n,int j)
{
int k,a,c;
a=1;
for(k=j+1;k<=n;k++) { a*=k; }
for(k=1;k<=(n-j);k++) { a=a/k; }
c=a;
return(c);
}
```

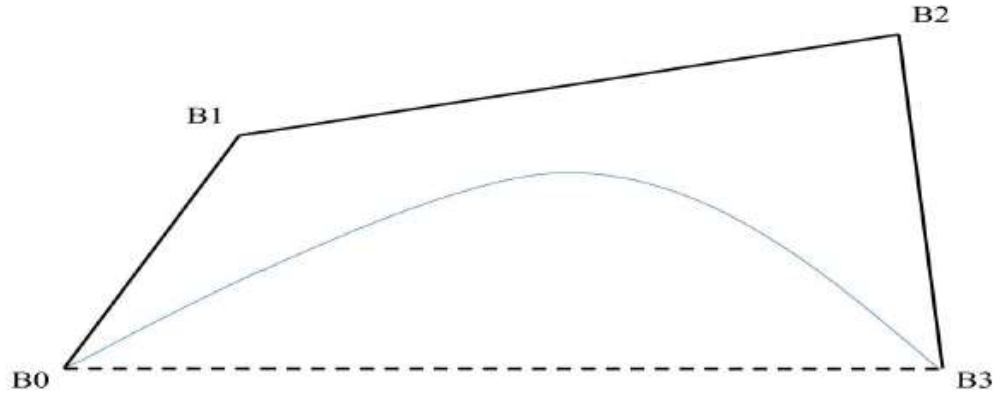


Fig : Bezier Curve and its Defining Polygon

EXPERIMENT NO : 7 - Animation Generation

Aim:-Write a C program for animation.

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
int i,j,loop=-40,loopl,k;
int gdriver=DETECT, gmode;
initgraph(&gdriver,&gmode,"");//Intialize Graphics Drivers
setcolor(GREEN);
/*Draw Steps*/
for(i=0,j=100;j<420;i+=40,j+=20)
{
line(i,j,i+40,j);
line(i+40,j,i+40,j+20);
}
for(loopl=100;loopl<420;loopl+=20)
{
loop+=40;
for(k=0;k<20;k++)
{
setcolor(RED);
circle(20+loop+k,loopl-20,18);
delay(10);
setcolor(BLACK);
circle(20+loop+k,loopl-20,18);
}
}
closegraph();
getch();
}
```