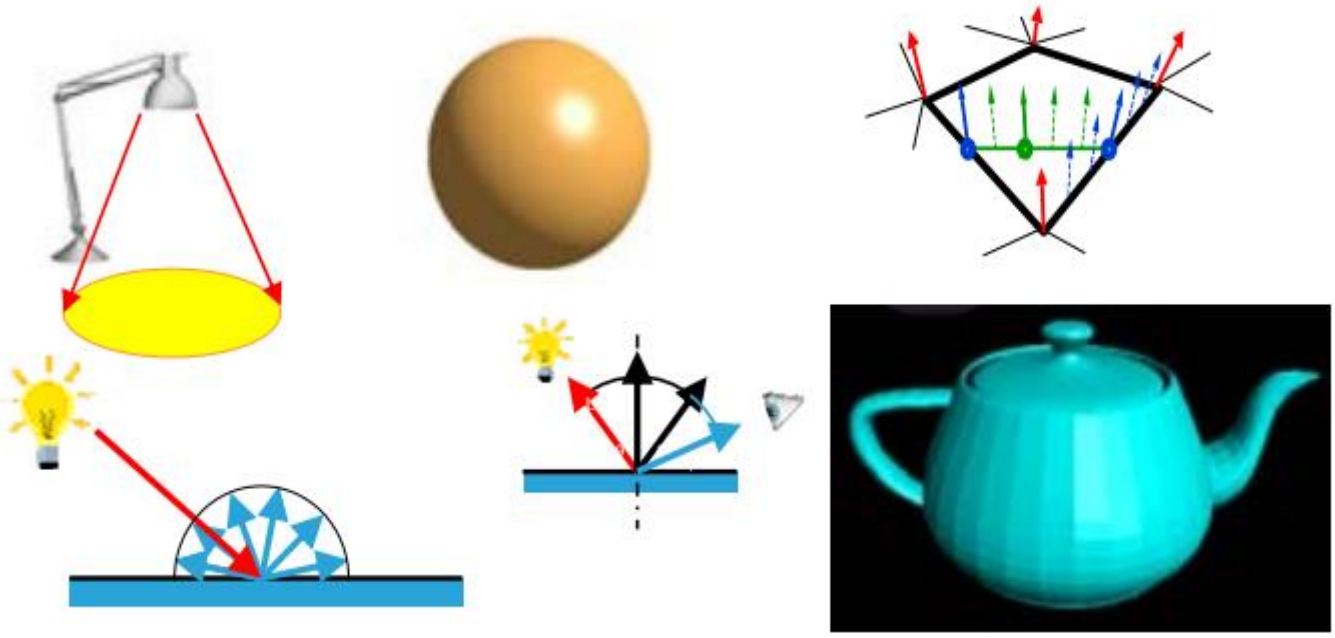
Computer Graphics (CSC. 209)

(B.Sc.CSIT, T.U.)



By

Keshav Bdr. Dhami

keshav.dhami@sagarmatha.edu.np

Chapter 1

Introduction to Computer Graphics

1.1 Introduction

- Started early in 1960s by Ivan Sutherland (MIT).
- Computer graphics is a field that is concerned with all aspects of producing pictures or images using a computer.
- It includes the creation, storage, and manipulation of images of objects.
- It is related to the generation and the representation of graphics by a computer using specialized graphic hardware and software.
- The graphics can be photographs, drawings, movies, or simulations etc.
- It is also defined as:
  - Data structures + Graphics algorithm + Graphical Languages=Computer Graphics
    - **Data structures:**
      - Data structures are used to store object attributes such as coordinate values, color consideration, depth etc.
      - Data structures that are suitable for computer graphics such as octree, quad tree, meshes etc.
    - **Graphics algorithms:**
      - Methods/procedures for picture generation, and transformations.
      - Graphics algorithms include scan line algorithms (for drawing points, lines, circles, ellipses etc.), clipping algorithms, fill algorithms, hidden surface removal algorithms etc.
    - **Graphics Languages:** Higher level languages for generation of graphics objects/pictures such as C, C++, Java, DirectX, QuickDraw, Display PDF and OpenGL.
- These objects come from diverse fields such as physical, mathematical, engineering, architectural, abstract structures and natural phenomenon.
- In short, Computer graphics refers:
  - Pictures, scenes that are generated by a computer.
  - Tools used to make such pictures, software and hardware, input/output devices.
  - The whole field of study that involves these tools and the pictures they produce.

1.2 CG tools

❖ CG tools include hardware and software tools.

- **Hardware Tools:**
  - Output devices video monitor, printer etc.
  - Input devices such as keyboard, mouse, touch panel etc.
  - Graphics cards.
- **Software tools:**
  - Operating system
  - Complier
  - Editor
  - Debuggers
  - Graphics libraries:
    - Functions/ routine to draw line or circle etc.
    - OpenGL.

1.3 Graphics System

There are five major elements in graphics system:

1. Input devices
2. Processor (display controller)
3. Memory
4. Frame buffer
5. Output devices

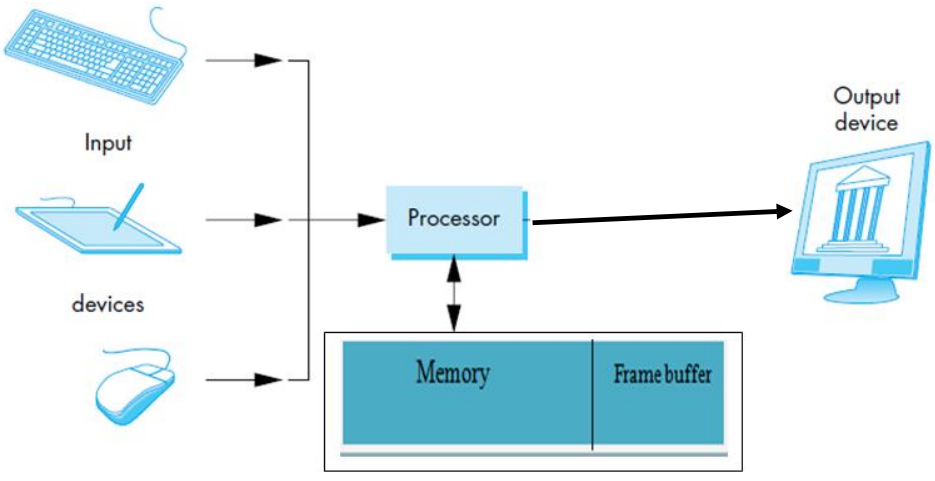


Figure 1.1: A graphics system.

Inside the frame buffer the image definition is stored. Display controller passes the contents of the frame buffer to the monitor.

1.4 Types of Computer Graphics

There are two kinds of computer graphics:

1.4.1 Bit Mapped or Raster Graphics

- Bit mapped graphics are graphics that are stored in the form of a bitmap in the frame buffer.
- They are a sequence of bits that get drawn onto the screen.
- We can create bit mapped graphics using a painting program. We cannot edit the components of a bitmapped image once we drew them.
- Enlargement of raster graphics reduces the clarity of image.

1.4.2 Object Oriented or Vector Graphics

- Vector graphics are graphics that are stored as a collection of drawing commands in the frame buffer.
- Graphics that are created using a drawing program are examples of vector graphics.
- When we create objects in a drawing program, we can still edit them after we have done something else. For example, if we draw a circle in one place, then draw a rectangle in another place, we can still select the circle and change its size and location.
- Enlargement of vector graphics does not affect the clarity of image.

1.5 Classification of Computer Graphics

1.5.1 On the basis of Interaction

I. Interactive Computer Graphics

- In interactive computer graphics user have some control over the picture i.e. user can make any change in the produced image. One example of it is the ping pong game.
- Computer graphics today is largely interactive, i.e. the user controls the contents, structure, and appearance of images of the objects by using input devices, such as keyboard, mouse, or touch-sensitive panel on the screen.

II. Passive Computer Graphics

- A computer graphics operation that transfers automatically and without operator intervention.
- Non-interactive computer graphics involves one way communication between the computer and the user.
- Picture is produced on the monitor and the user does not have any control over the produced picture.

1.5.2 On the basis of type of Objects

- Two Dimensional Computer Graphics(2D)
    - Generation of 2D objects such as lines, triangles, circles, rectangles, gray-scale images etc.
- Three Dimensional Computer Graphics(3D)

- Generation of 3D objects such as spheres, polygons, color images with various effects etc.

1.6 Application of Computer Graphics

1.6.1 Computer Aided Design (CAD)

- One of the major uses of computer graphics is in design processes.
- CG is used to design components and systems of mechanical, electrical, electrochemical, and electronic devices, including structures such as buildings, automobile bodies, airplane and ship hulls, very large scale integrated (VLSI) chips, optical systems and telephone and computer networks.
- These designs are more frequently used to test the structural, electrical, and thermal properties of the systems.
- Architects use computer graphics to layout floor plans that shows positioning of rooms, doors, windows, stairs, shelves and other building features. Electrical designers then try out arrangements for wiring, electrical outlets and other system to determine space utilization on a building.

1.6.2 Presentation Graphics

- Another major application area of computer graphics is the Presentation Graphics.
- Presentation Graphics is commonly used to summarize financial, statistical, mathematical, scientific and economic data for research reports, managerial reports and other types of reports.
- Typical examples are bar charts, line graphs, surface graphs, pie charts and other displays showing relationship between multiple variables.
- The 3D graphics are usually used simply for effects; they can provide a more diagrammatic or more attractive presentation of data relationship.

1.6.3 Computer Art & Commerce

- Computer graphics is used to generate arts.
- They are widely used in both fine art and commercial art applications. Fine art is drawn by artist hand and this kind of art is perfect to the artist skill. Artist use a variety of computer methods including special purpose hardware, artist's paints brush program, other paint packages, specially developed software.
- Also used for advertising purpose.

1.6.4 Entertainment

- Computer graphics methods are now commonly used in making motion pictures, music videos and TV shows.
- Disney movies such as Lion Kings and The Beauty of Beast, and other scientific movies like Jurassic Park, The lost world etc. are the best example of the application of computer graphics in the field of entertainment.
- Also used to develop in gaming applications such as Prince of Persia, Far Cry etc.

1.6.5 Education & Training

- Computer graphics is used in education and training for making it more effective and more illustrative.
- Computer generated models of physical, financial, and economic systems are often used as educational aids. A student can learn surgery using data gloves and realistic computer graphics.

1.6.6 Visualization

- Visualization is the process of visually representing the data. To visualize large amount of information graphical computer systems are used.
- Generating computer graphics for scientific, engineering, and medical data sets is termed as scientific visualization whereas business visualization is related with the non-scientific data sets such as those obtained in economics.
- Some methods generate very large amount of data/information, analysis the property of the whole amount of data is very difficult. Visualization simplifies the analysis of data by using graphical representation. Visualization makes easier to understand the trends and patterns inherent in the huge amount of data sets.

1.5.7 Image Processing

- Image can be created using simple point program or can be fed into computer by scanning the image. These picture/ images need to be changed to improve the quality.
- Form image/pattern recognition systems, images need to be changed in specified format so that the system can recognize the meaning of the picture.
- Currently computer graphics is widely used for image processing.

1.6.8 User Interface (GUI)

- Most applications have user interfaces that rely on desktop windows systems to manage multiple simultaneous activities, and on point-and click facilities to allow users to select menu items, icons and objects on the screen. These activities fall under computer graphics.
- Typing is necessary only to input text to be stored and manipulated. For example, Word processing, spreadsheet, and desktop-publishing programs are the typical examples where user-interface techniques are implemented.
- GUI provides point-and-click facilities to allow users to select menu items, icons, and objects on the screen.

1.6.9 Office Automation and Electronic Publishing

- Computer graphics has facilitated the office automation and electronic publishing which is also popularly known as desktop publishing, giving more power to the organizations to print the meaningful materials in-house.

- Office automation and electronic publishing can produce both traditional printed (Hardcopy) documents and electronic (softcopy) documents that contain text, tables, graphs, and other forms of drawn or scanned-in graphics.

1.6.10 Simulation and Modeling

- Mapping the real-world scenarios into virtual world using computer graphics.
- E.g.; Robot Operation Simulation, Training.
- Once graphics systems evolved to be capable of generating sophisticated images in real time, engineers and researchers began to use them as simulators.
- One of the most important uses has been in the training of pilots. Graphical flight simulators have proved both to increase safety and to reduce training expenses.

1.6.11 Cartography

- Cartography is a subject, which deals with the making of maps and charts.
- Computer graphics is used to produce both accurate and schematic representations of geographical and other natural phenomena from measurement data.
- Examples include geographic maps, oceanographic charts, weather maps, contour maps and population-density maps. Surfer is one of such graphics packages, which is extensively used for cartography.

1.7 Color Models

- A color space or color model is a method by which we can specify, create and visualize color.
- A color model or color space is a method for explaining the properties or behavior of color within particular context.
- It is a mathematical way of representing a set of colors.
- A color model is an orderly system for creating a *whole range of colors from a small set of primary colors.*
- There are several established color models used in computer graphics **RGB, CMY, CMYK, HSV, XYZ** etc.
- But the two most common are the RGB model (Red-Green-Blue) for computer display and the CMY, & CMYK model (Cyan-Magenta-Yellow-Key Black) for printing.

SN	Color Model	How a Color is represented?	Application
1	***RGB***: R = Red, G= Green, B = Blue.	1. Red, green, and blue are mixed which creates a variety of different colors. 2. A color is represented by three tuple (R, G, B) 3. A new color can be formed by varying the values of R, G, and B from 0-255. 4. ***Example***: (R= 255, G=255, B = 255) = WHITE. (R = 0, G = 0, B = 0) = BLACK. Similar, other colors can be formed.	Digital Camera, Color TV, Monitor etc.
2	***CMYK***: C=Cyan, M= Magenta, Y= Yellow, K = Key-Black.	1. The colors cyan, magenta, yellow and black are used as primary colors which are combined in different extents to get different colors. 2. A color is represented by four tuple(C, M, Y, K).	Used in printers.
3	***HSV:*** *H = Hue* *S = Saturation* *V = Value*	1. A color is a combination of Hue, Saturation, and Value. 2. Hue = Primary color component, Saturation = percentage of the color, Value = intensity of the color.	Converting grayscale images to RGB color. Human's way of perceiving the color.

Chapter 2

Hardware and Software Concept

2.1 Input Devices

Mouse, Touch Screen, Light Pen, Data Glove, Tablet (Digitizer), Bar Code Reader

Input device are used to feed data or information into a computer system. They are usually used to provide input to the computer upon which reaction, outputs are generated. Data input devices like keyboards are used to provide additional data to the computers whereas pointing and selection devices like mouse, light pens, touch panels are used to provide visual and indication-input to the application.

Keyboard

Keyboard is a primary serial input device. For each key press, a keyboard senses specific codes(American standard code for Information Interchange, ASCII) to the computes. Keyboards are also capable of sending/coding combinations of key press and special keys like function keys. It is useful for expert users in giving exact commands to the computer. Cursor controlled keys and function keys are used for doing operations in a single keystroke.

Mouse (Mechanical and optical)

A mouse is a small hand-held box used to position the screen cursor. Mouse is a serial input device used to select object movement in graphics system. According to working mechanism two types of mouse are there. They are:

Mechanical Mouse

A mechanical mouse uses wheels or rollers at the bottom surface, which rotates as the mouse is moved along a flat surface to move the cursor. These wheels or rollers record the amount and the direction of the movement.

Optical Mouse

An optical mouse uses an optical sensor to detect the movement across a special mouse pad that has a grid of horizontal and vertical lines. The optical sensor detects movement across the lines in the grid.

Z-Mouse

Z-mouse includes three buttons. A thumb wheel on the side, a track ball on the top, and a standard mouse ball underneath. This design provides ***six degrees of freedom*** to select an object from the spatial position. With this we can pick up an object, rotate it and we can move it in any direction. Used in virtual reality and CAD systems.

Trackball and Space ball

A **trackball** is a ball that can be rotated with the fingers or palm of the hand to produce screen cursor movement.

- It is a two dimensional positioning device.
- Potentiometer's attached to the ball, measure the amount and direction of rotation. Trackballs are often mounted on keyboards.

Space ball is a three dimensional positioning and pointing device that provides six degrees of freedom.

- Strain gauges are used to measure the amount of pressure applied to the space ball to provide the input.
- It is used in virtual-reality systems, modeling, animation, CAD, and other applications.

Data glove

Constructed with a series of sensors that can detect hand and finger motions. The transmitting and receiving antennas can be structured as a set of three mutually perpendicular cols, forming a three dimensional Cartesian coordinates system. Electromagnetic coupling between the three pairs of coil is used to provide information about the position and orientation of hand.

Digitizers

A common device for drawing, painting, or interactively selecting coordinate positions on an object is digitizer. It is used to scan over a drawing or object and to input a set of discrete coordinate positions. And it can be used in both 2D and 3D graphics. ***Graphics tablet*** is an example.

Light Pens

Light pens are pencil shaped devices used to select screen positions by detecting the light coming from points on the CRT screen. They are sensitive to the short burst of light emitted from the phosphor coating as the instant electron beam strikes a particular point. Other light sources, such as background light in the room are usually not detected by a light pen.

Joystick

A joystick is a device that moves in all directions and controls the movement of the screen cursor. It consists of a small, vertical lever mounted on a base that is used to steer the screen cursor. Joysticks are mainly used for computer games, controlling industrial robots, and for other applications such as flight simulators, training simulators etc.

Bar Code Reader

A bar code is a machine-readable code in the form of a pattern of parallel vertical lines. They are commonly used for labeling gods that are available in supermarkets, numbering books in libraries etc. These codes are sensed and read by a photoelectric device called bar code reader that reads the code by means of reflected light. The information recorded in a bar code reader is fed into the computer, which recognizes the information from the thickness and spacing of bars.

Touch-Panel

Touch panels are a sensitive surface that is used to point directly. The panel can be touched by finger or any other object like stylus. Transparent touch panels are integrated with computer monitor for the manipulation of information display. A basic touch panel senses voltage drop when a user touches the panel. It knows where the voltage has dropped and accordingly calculates the touch position.

2.2 Output Devices

2.2.1 Cathode Ray Tube

The primary output device in a graphical system is the video monitor. The main element of a video monitor is the Cathode Ray Tube (CRT), shown in the following illustration. The cathode ray tube (CRT) is an evacuated tube containing one or more electron guns (a source of electron) and a phosphor coated screen used to view images.

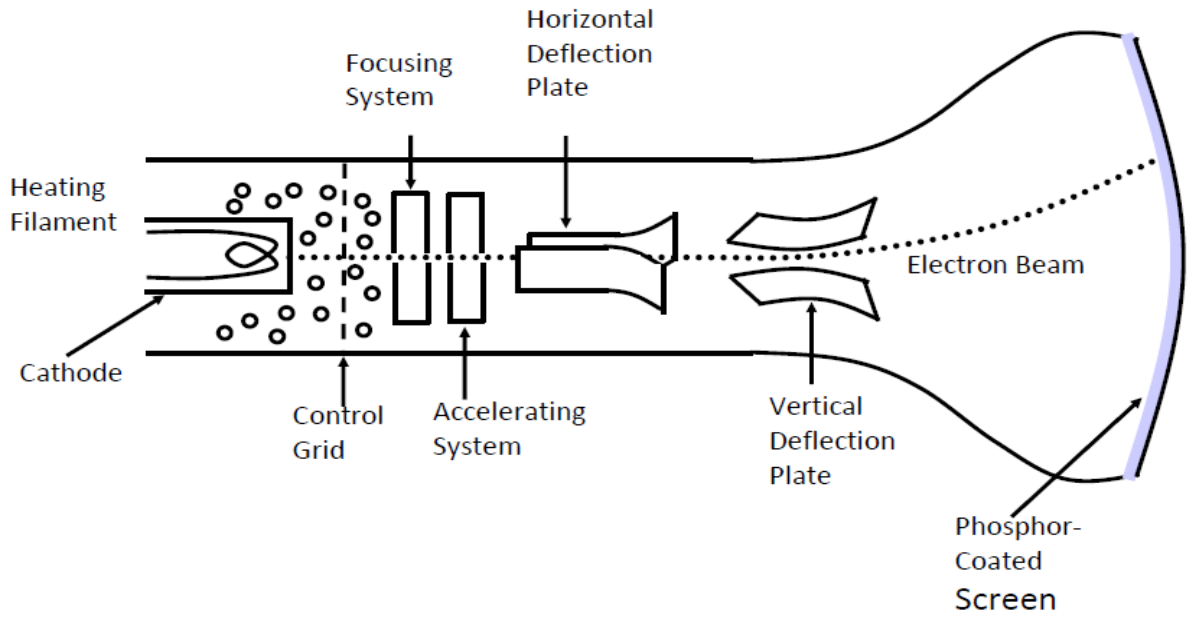


Figure 2.1: Cathode Ray Tube

Main Components of CRT

An electron gun

- The primary components of an electron gun in a CRT are the heated metal cathode and a control grid.
  - **Heated metal cathode:** Heat is supplied to the cathode by directing the beam through a coil of wire called the filament inside the cylindrical cathode structure.
  - **Control grid:** Intensity of the electron beam is controlled by setting the voltage levels on the control grid, which is a metal cylinder that fits to the cathode.

Focusing System & Accelerating Anode

- The focusing system in a CRT is needed to force the electron beam to converge into a small spot as it strikes the phosphor.
- And the accelerating anode is used to accelerate electron beam towards the phosphor coated screen. Otherwise, the electron beam would not reach to the screen.

Deflection System

- It is used to control the vertical and horizontal scanning of the electron beam.

The operation of CRT

- The electron gun emits a beam of electrons (cathode rays).
- The electron beam passes through focusing and deflection systems that direct it towards specified positions on the phosphor-coated screen.
- When the beam hits the screen, the phosphor emits a small spot of light at each position contacted by the electron beam. The glowing positions are used to represent the picture in the screen.
- The amount of light emitted by the phosphor coating depends on the number of electrons striking the screen. **The brightness of the display is controlled by varying the voltage on the control grid.**
- Because the light emitted by the phosphor decays very rapidly with time. So, it redraws the picture by directing the electron beam back over the same screen points quickly. Thus, also referred to as a refresh CRT.

Properties of CRT

1. Persistence

- Persistence is the one of the major property of phosphor used in CRT's.
- It means how long phosphors continue to emit light after the electron beam is removed.
- Persistence is defined as the time it takes the emitted light from the screen to decay to one-tenth of its original intensity.

- Lower persistence phosphors require higher refresh rates to maintain a picture on the screen. A phosphor with lower persistence is useful for animation and a higher–persistence phosphor is useful for displaying highly complex static picture.
- Graphics monitor are usually constructed with the persistence 10 to 60 microseconds.

2. Resolution

- The maximum number of points (pixel) that can be displayed without overlap on a CRT is referred to as the resolution.
- It is usually denoted as width × height, with the units in pixels.
- It is also defined as maximum number of points displayed horizontally and vertically without overlap on a display screen.
- It represents number of dots per inch (dpi/pixel per inch) that can be plotted horizontally and vertically.
- Resolution of 1280 x 720 means that there are 1280 columns and 720 rows.
- Resolution of 1024 x 768 means the width is 1024 pixels and the height is 768 pixels.

3. Aspect Ratio

- Another property of video monitors is aspect ratio.
- This number gives the ratio of total number of vertical pixel lines to total horizontal pixel lines i.e., the ratio of the width to the height of an image or screen.
- *Aspect ratio* = Width/Height.

774. Refresh Rate

- Light emitted by phosphor fades very rapidly, so to keep the drawn picture glowing constantly; it is required to redraw the picture repeatedly by quickly directing the electron beam back over the some point. This process is called refresh operation.
- The no of times/sec the image is redrawn to give a feeling of non-flickering pictures is called refresh-rate. Refresh rates are described in units of cycles per second, or Hertz (Hz), where a cycle corresponds to one frame.
- If Refresh rate decreases, flicker develops.
- Refresh rate above which flickering stops and steady it may be called as critical fusion frequency (CFF).

5. Bit Depth/Color Depth

- Number of bits used to store color information about a pixel.
- Represented as bits/pixel.
- A system with 1 bit/pixel can display two colors.
- Similarly, a system with 8 bits/pixel can display 256 colors.

2.2.2 Types of Refresh CRT's

Two types of Refresh CRT's:

a. *Raster-Scan Displays*

b. *Random-Scan Displays*

2.2.2.1 Raster-Scan Displays

- The most common type of graphics monitor employing a CRT is the raster-scan display.
- In raster scan approach, the viewing screen is divided into a large number of discrete phosphor picture elements, called pixels.
- Row of pixels is called the scan line. The matrix of pixels or collection of scan lines constitutes the raster.
- *As electron beam* moves across each row, the beam intensity is turned on and off to create a pattern of illuminated spots.
- Picture definition is stored in a memory called frame buffer or refresh buffer. Frame buffer holds all the intensity value for each screen point.
- In monochromatic CRT's (i.e., black-and-white system) with **one bit per pixel**, the frame buffer is commonly called a **bitmap**. For systems with multiple bits per pixel, the frame buffer is often referred to as a **pixmap**.
- Stored intensity values are then retrieved by the display processor from the frame buffer and "painted" on the screen, pixel-by-pixel, one row (scan line) at a time.

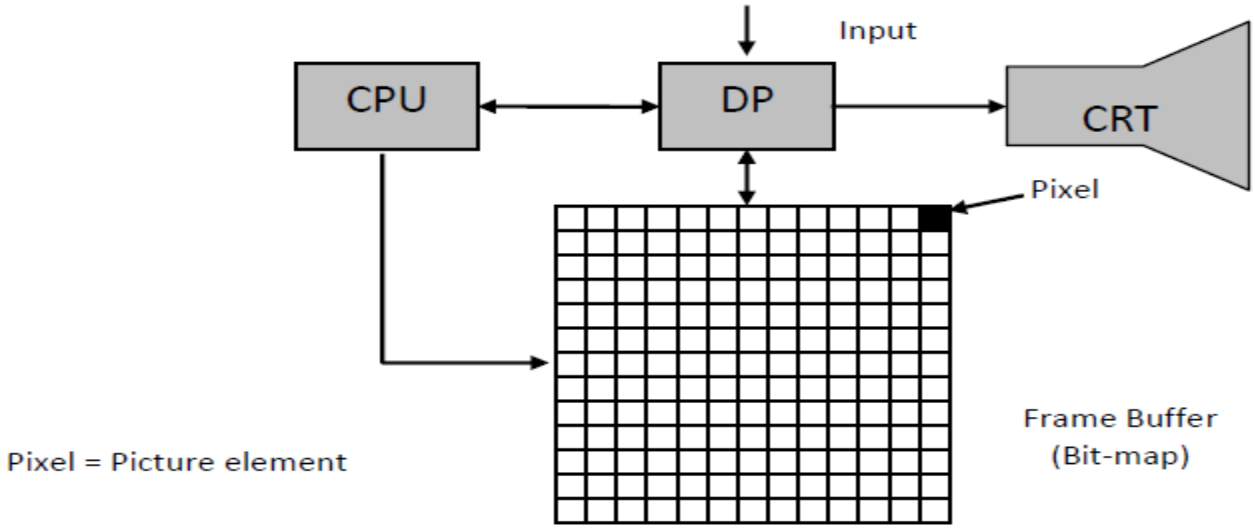


Figure 2.2: Raster Scan display system

- **Horizontal Retrace and Vertical Retrace:**
  - At the end of each scan line the electron beam returns to the left side of the screen to begin the displaying the next scan line. The return to the left of the screen, after refreshing each scan line, is called ***horizontal retrace*** of the electron beam.

- And at the end of each frame, the electron beam returns to the top left corner of the screen to begin the next frame, it is called ***vertical retrace***.

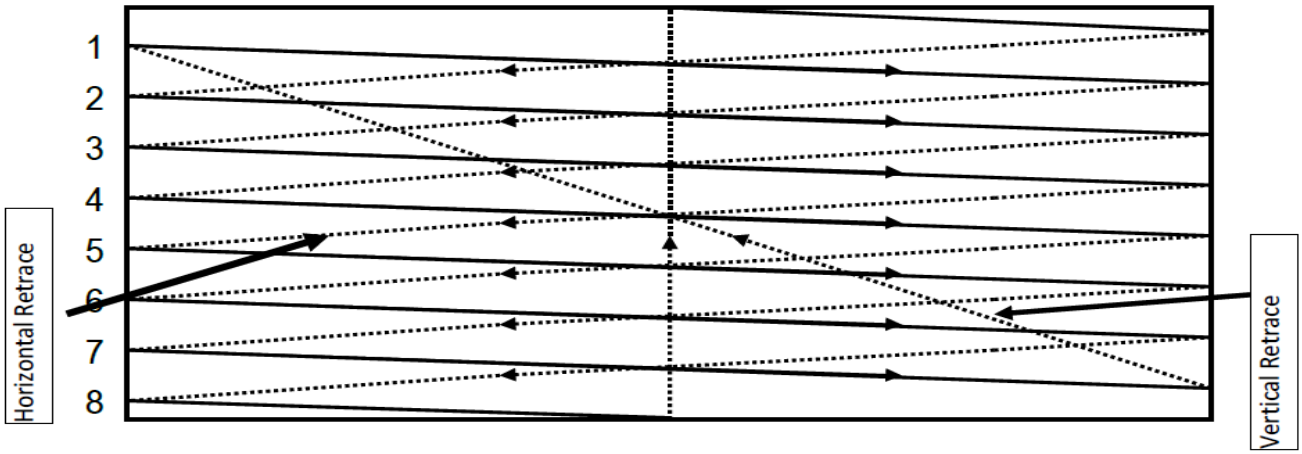


Figure 2.3: Raster Scan display system

❖ Two types of Raster-scan systems:

- Non-interlaced Raster-Scan System or Progressive Raster Scan System
- Interlaced Raster-Scan System

A. Non-interlaced Raster-Scan System or Progressive Scanning

- ❖ Follows non-interlaced refresh procedure.
- ❖ In non-interlaced refresh procedure, electron beam sweeps over entire scan lines in a frame from top to bottom in one pass.
- ❖ That is, content displays both the even and odd scan lines (the entire video frame) at the same time (fig 2.4).

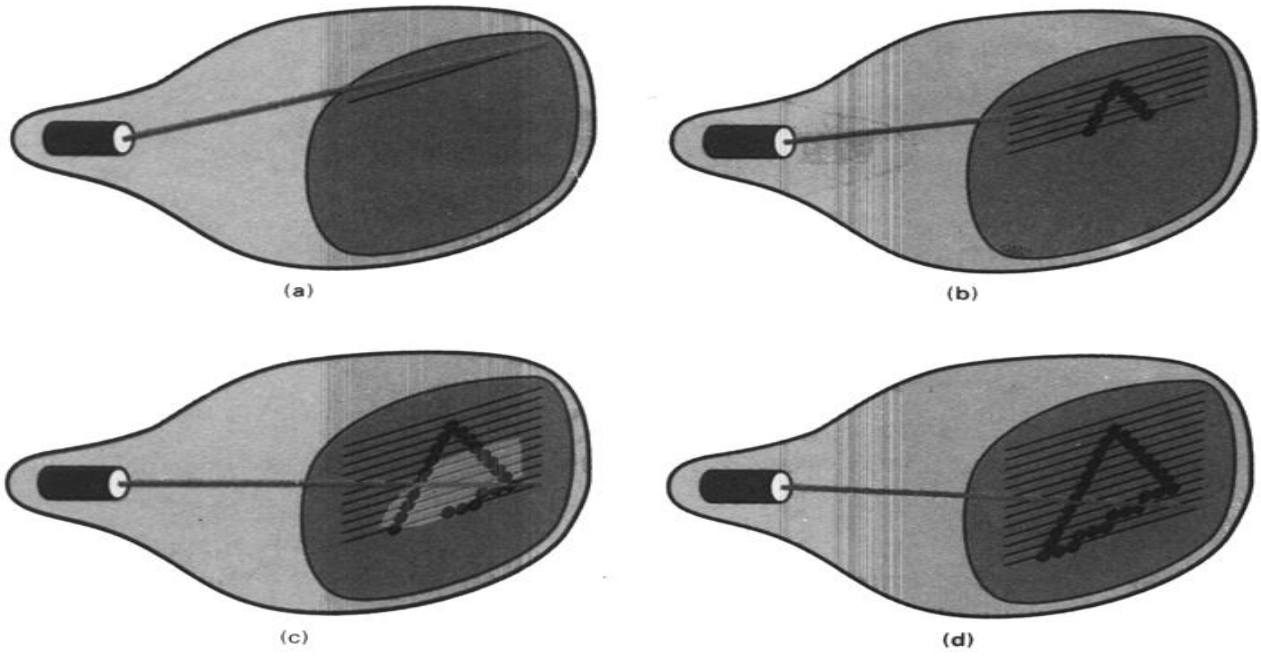


Figure 2.4: A raster-scan system displays an object as a set of discrete points across each scan line.

B. Interlaced Raster-Scan System

- On some raster-scan systems (and in TV sets), each frame is displayed in two passes using an interlaced refresh procedure.
- In interlaced scan, each frame is displayed in two passes. First pass for even scan lines and then after the vertical re-trace, the beam sweeps out the remaining scan lines i.e., odd scan lines (shown in *Fig.2.5)*. Interlacing is primarily used with slower refreshing rates. This is an effective technique for avoiding screen flickering.

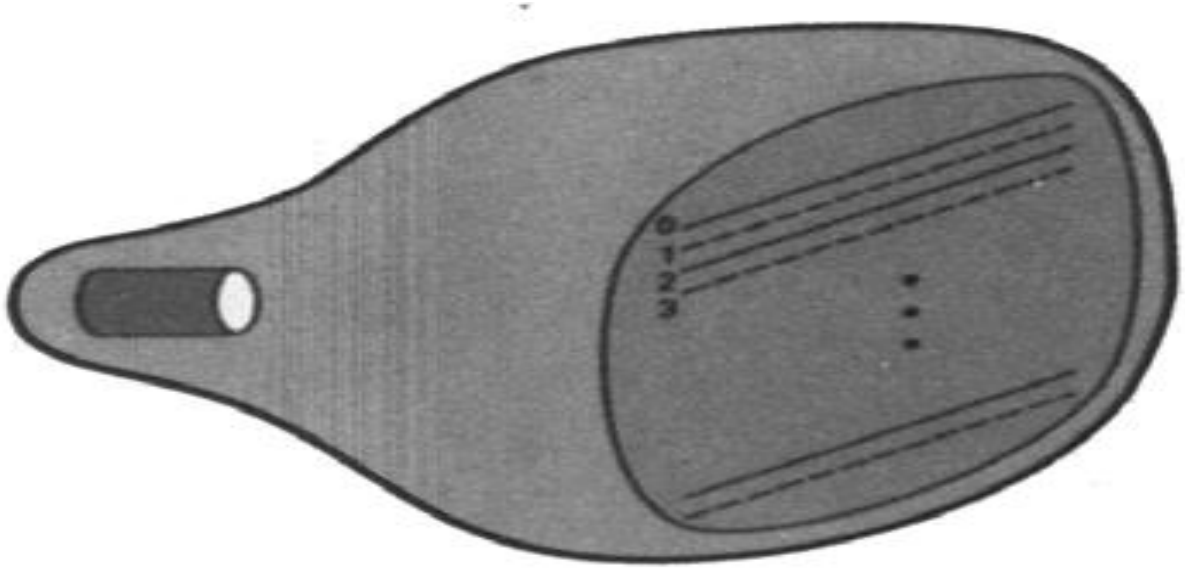


Figure 2.5: Interlacing Scan lines on a raster-scan display. First, all points on the even-numbered (solid) scan lines are displayed; and then all points along the odd-numbered (dashed) lines are displayed.

Refresh Rates in Raster Scan displays

Refreshing on raster scan display is carried out at the rate of 60 to 80 frames per second. Sometimes, refresh rates are described in units of cycles per second, or ***Hertz (Hz)***, where a cycle corresponds to one frame.

Examples

- *Monitors, Home television, printers.*

Applications

- For the realistic display of scenes containing subtle shading and color patterns.

2.2.2.2 Random-Scan/Calligraphic displays/Vector display system

- *In a random scan display unit, electron beam directed towards only to the parts of the screen where a picture is to be drawn.*
- Random-scan monitors draw a picture one line at a time and for this reason are also referred to as vector displays (or stroke-writing or calligraphic displays).

- Random scan system uses an electron beam which operates like a pencil to create a line image on the CRT.
- Picture definition is stored as a set of line drawing instructions in an area of memory called the refresh display file (*Display list or display file*).
- To display a picture, the system cycles through the set of commands (line drawing) in the display file. After all commands have been processed, the system cycles back to the first line command in the list.
- The component line can be drawn or refreshed by a random scan display system in any specified order (shown in *figure 2.6.*).

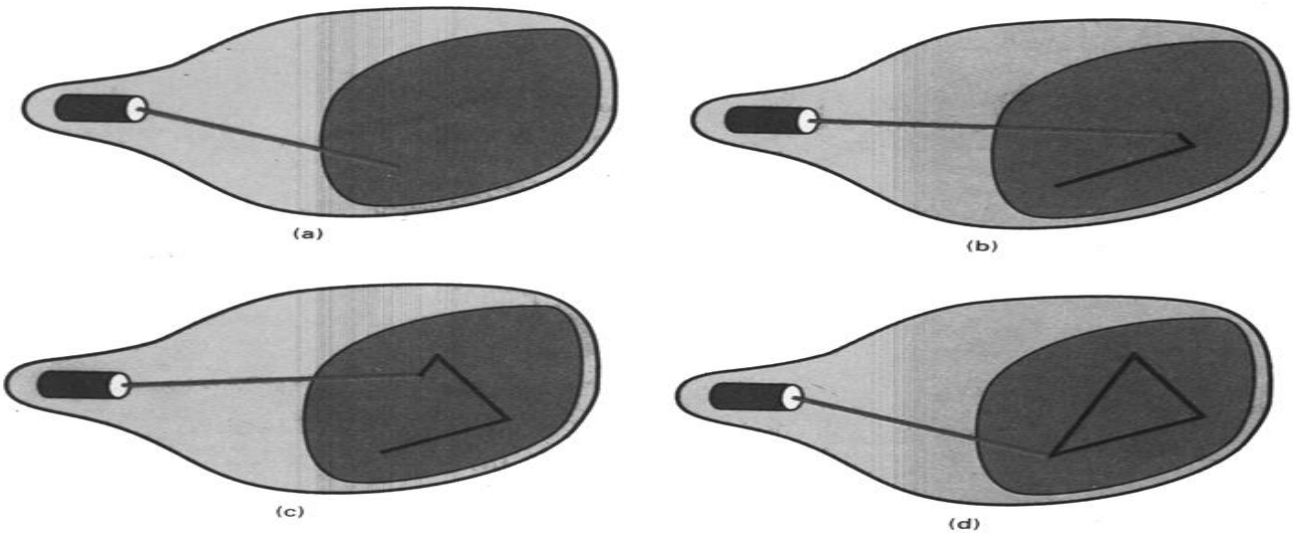


Figure 2.6: A random-scan system draws the component lines of an object in any order specified.

- The refresh rate of vector display depends upon the no. of lines to be displayed for any image.
- Random scan systems are designed for line-drawing applications and cannot display realistic shaded scenes. Since CRT beam directly follows the line path, the vector display system produce smooth line.

Example: A pen plotter.

Refresh Rates in Random-Scan displays:

- Generally, refreshing on random-scan display is carried out at the rate of 30 to 60 frames per second.

Applications:

- Random-scan systems are used in line-drawing applications.
- Vector displays generally used to produce graphics with higher resolution.

Exercises:

1. **A system with 24 bits per pixel and resolution of 1024 by 1024. Calculate the size of frame buffer (in Megabytes).**

   Frame size in bits= 24*1024*1024 bits

   Frame size in bytes= 24*1024*1024 / 8 bytes (since, 8 Bits = 1 Byte)

   Frame size in kilobytes= 24*1024*1024 / 8 * 1024 kb (since, 1024 Bytes = 1 KB)

   So, Frame size in megabytes= 24*1024*1024 / 8*1024*1024 MB (since, 1024 KB = 1 MB)

   = 3 MB.

2. **How Many k bytes does a frame buffer needs in a 600 x 400 pixel?**

   Solution: Resolution is 600 x 400.

   Suppose, n bits are required to store 1 pixel.

   Then, the size of frame buffer = Resolution * bits per pixel

   = (600 * 400) * n bits

   = 240000 n bits

   = $\frac{240000\ n}{1024 * 8}$ kb (as 1kb = 1024 bites)

   = 29.30 n k bytes.

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

   We know that, Aspect ratio = Width / Height

   = $\frac{8 \times 100}{10 \times 100}$ = 4 / 5

   So, aspect ratio = 4 : 5.

4. **Consider three different raster systems with resolutions of 640 x 480, 1280 x 1024, and 2560 x 2048.**

   a) What size is frame buffer (in bytes) for each of these systems to store 12 bits per pixel?

   b) How much storage (in bytes) is required for each system if 24 bits per pixel are to be stored?

5. **Suppose RGB raster system is to be designed using on 8 inch X 10 inch screen with a resolution of 100 pixels per inch in each direction. If we want to store 6 bits per pixel in the frame buffer, how much storage (in bytes) do we need for frame buffer?**

6. **If you are supposed to create an animated movie of 20 minutes and your video is of 30fps (*frames/second*). Calculate the number of frames should be in this video.**

7. **How long would it take to load a 640 by 480 frame buffer with 12 bits per pixel if 10^5 bits can be transferred?**
8. **A raster system has a resolution of 1280 by 1024. How many pixels in frame buffer needs to be accessed per second by a display controller that refreshes the screen at a rate of 60 frames per second?**
9. **How much time spent scanning across each row of pixels during screen refresh on a raster system with resolution 1280 by 1024 and a refresh rate 60 frame per second?**
10. **Calculate the total memory required to store a 10 minutes video in a SVGA system with 24 bit true color, and 25 fps.**

2.2.3 Color CRT

A CRT monitor displays color pictures by using a combination of phosphors that emit different-colored light. By combining the emitted light from the different phosphors, a range of colors can be generated. Two basic techniques for producing color displays with CRT are:

1. ***Beam-penetration method***
2. ***Shadow-mask method***

2.2.3.1 Beam Penetration Method

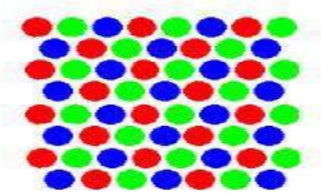
- *This method is commonly used for random scan display or vector display. In random scan display CRT, the two layers of phosphor usually red and green are coated on CRT screen. Display color depends upon how far electrons beam penetrate the phosphor layers.*
- Slow electron beam excites only outer red layer so that we can see red color displayed on the screen pixel where the beam strikes.
- Fast electron beam excites green layer penetrating the red layer and we can see the green color displayed at the corresponding position.
- At Intermediate beam speeds or an average electron beam gives the combination of red and green color. That is yellow and orange.
- The speed of the electrons and hence the screen color at any point, is controlled by control grid by controlling the beam-acceleration voltage.
- *Beam-penetration is an inexpensive way to produce color in random-scan monitors, but quality of pictures is not as good as other methods since only 4 colors are possible.*

2.2.3.1 Shadow Mask Method

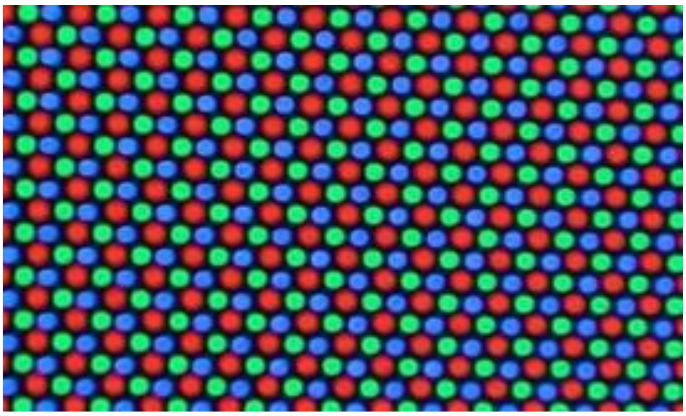
Shadow-mask methods are commonly used in raster-scan systems (including color TV) because they produce a much wider range of colors than the beam-penetration method.

- A shadow Mask CRT has three phosphor color dots at each pixel location. One phosphor dot emits a red light, another emits green light and the last one emits a blue light.
- *These phosphor color dots are arranged in a precise geometric pattern. There are two primary variations.*
    1. ***The stripe pattern***
        - Color dots are arranged in stripes pattern for a single pixel.
    2. ***The delta pattern***
        - Color dots are arranged in a delta pattern i.e. in a triangular pattern.

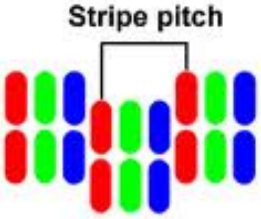
Delta Pattern



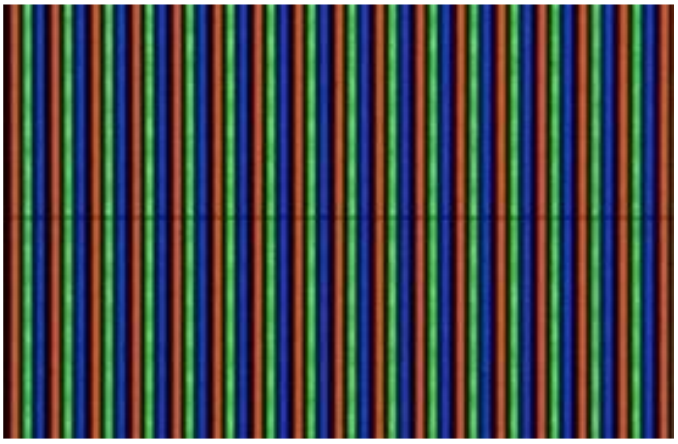
Monitor view



Stripe Pattern



Monitor view



- This type of CRT also has three electron guns one for each red, green, and blue color dot.

- In phosphor coating there may be either strips one for each primary color, for a single pixel or there may be three dots one for each pixel in delta fashion.
- In the shadow-mask CRT, special metal plate with small holes -called a shadow mask is placed just behind the screen.
- The shadow mask—ensures that an electron beam excites only phosphors of the proper color.
- Depending on the pattern of coating of phosphor, two types of raster scan color CRT are commonly used using shadow mask method.

Delta-Delta Shadow Mask CRT

- In Delta-Delta CRT, phosphor color dots are arranged in triangular groups called triads, each triad consisting of three phosphors, one of each primary.
- The three electron beams are deflected and focused as a group onto the shadow mask, which contains a series of very fine holes aligned with the phosphor dot patterns.
- The phosphor dots in the triangles are arranged so that each electron beam can activate only its corresponding color dot when it passes through the shadow mask.
- When the three beams pass through a hole in the shadow mask, they activate a dot triangle, which appears as a small color spot on the screen.
- Color variation can be obtained by varying the intensity levels of the three electron guns.

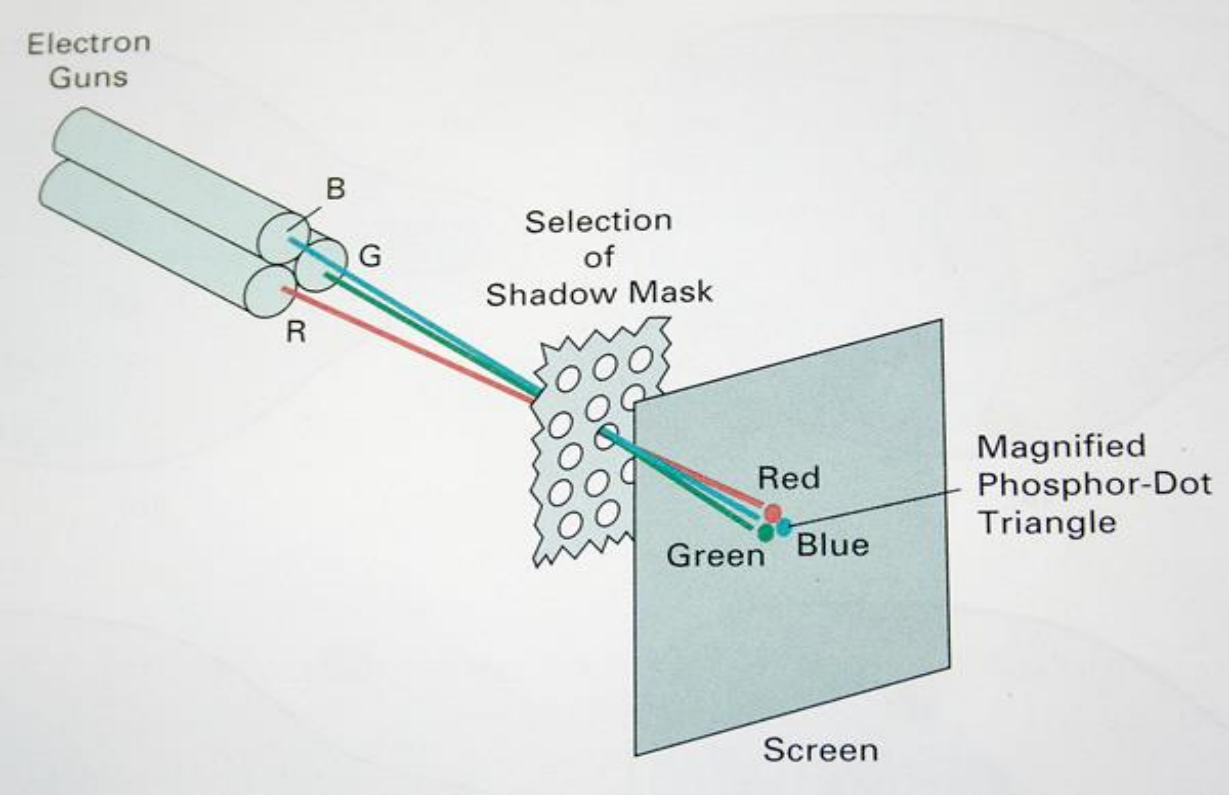


Figure 2.7: Operation of a delta–delta, shadow-mask CRT. Three electron guns, aligned with the triangular color-dot patterns on the screen, are directed to each dot triangle by a shadow mask.

Inline Color CRT

- This CRT uses stripe pattern instead of delta pattern.
- Three strips one for each R, G, and B colors are used for a single pixel along a scan line so called inline.
- Three beams simultaneously expose three inline phosphor dots along scan line.

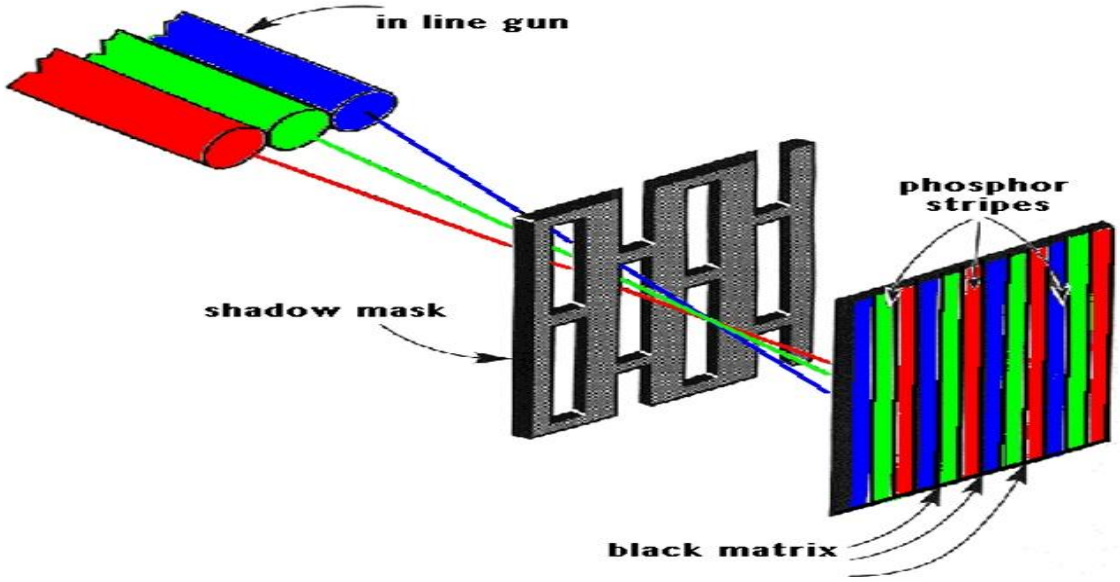


Figure 2.8: Inline Color CRT

2.2.4 Direct-view Storage tubes (DVST)

- This is an alternative method for maintaining a screen image.
- It sores the picture information inside the CRT instead of refreshing the screen.
- A direct-view storage tube (DVST) stores the picture information as a charge distribution just behind the phosphor-coated screen.
- Two electron guns are used in a DVST. One, the primary gun, is used to store the picture pattern; the second, the flood gun, maintains the picture display.
- **Pros:**
  - Since no refreshing is needed complex pictures can be displayed in high-resolution without flicker.
- **Cons:**
  - Ordinarily do not display color

2.3 Architecture of Raster and Random Scan System

- In this section, various elements of raster scan systems and random scan systems and there organization will be discussed.

2.3.1 Architecture of Raster-Scan System

- The raster graphics systems typically consist of several processing units. CPU is the main processing unit of computer systems.

- Besides CPU, graphics system consists of a special buffer processor called video controller or display controller. It is used to control the operation of the display device.
- In addition, to the video controller, raster scan systems can have other processors as co-processors which are called graphic controller or display processors or graphics processor unit (GPU).
- Image is generated by CPU/GPU and write (or load) into frame buffer. Image in frame buffer is read out by video controller to display on the screen.

A. Simple Raster display System

- The organization of simple raster system (without additional processors i.e., display processor) is shown in the figure below.
- Here, the frame buffer can be anywhere in the system memory, and the video controller accesses the frame buffer to refresh the screen.

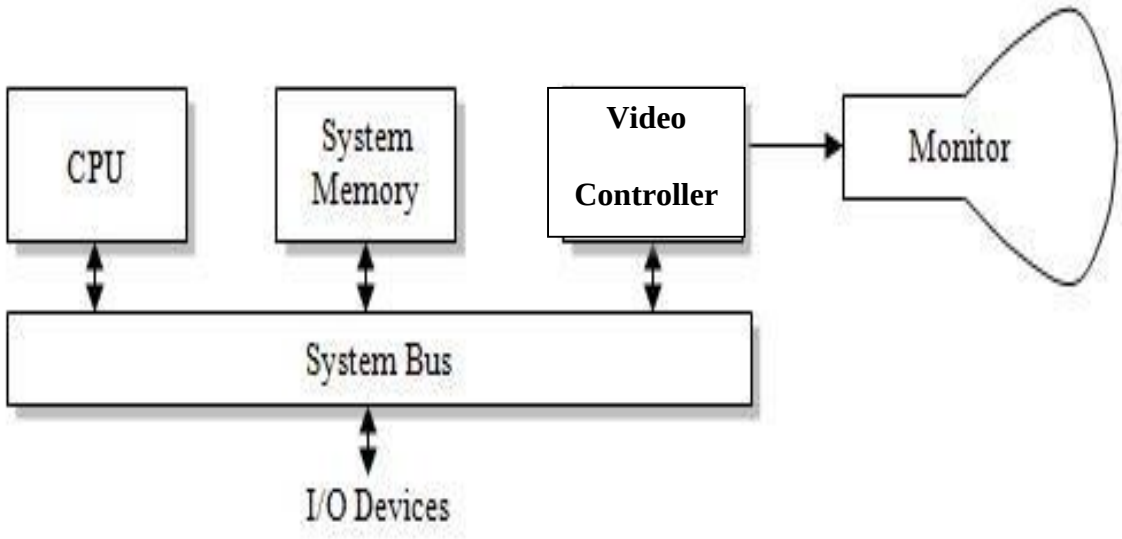


Fig 2.9: Simple raster display system architecture, frame buffer may be anywhere in system memory, the video controller accesses the frame buffer via the system bus.

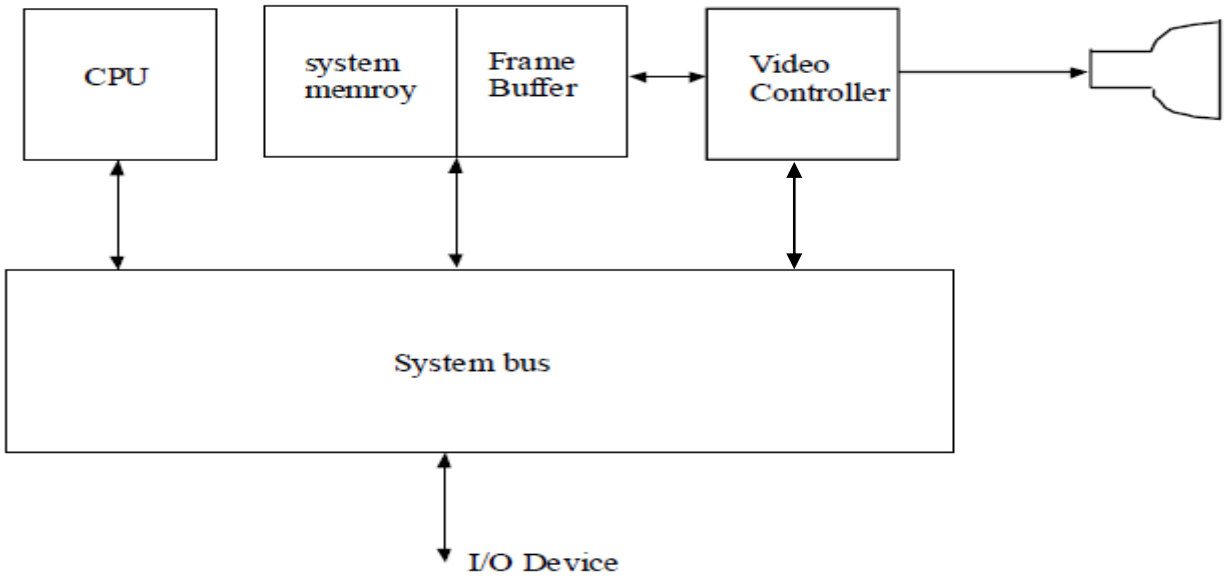
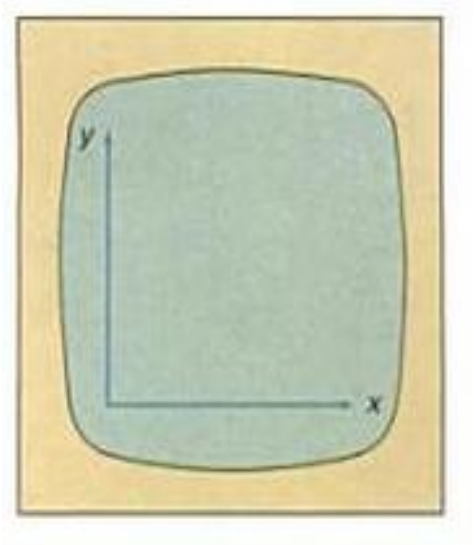


Fig 2.10: A common raster display system architecture. A dedicated portion of the system memory is dual-ported (*the memory has two independent access paths*)

- A common architecture in which a fixed area of the system memory is reserved for the frame buffer, and the video controller is given access to the frame buffer memory.
- The video controller cycles through the frame buffer, one scan line at a time. The content of frame buffer is used to control the CRT beam's intensity or color.

Working Mechanism of Video Controller:



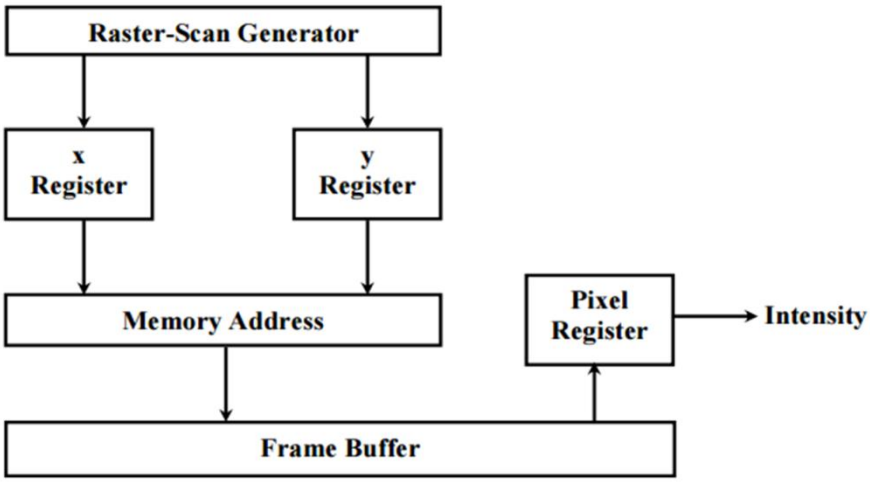


Fig 2.11: The coordinate origin of the graphics monitor is defined as the lower left screen corner

Fig 2.12: Basic video controller Refresh operation.

- The video controller is organized as in figure above. The raster-scan generator produces deflection signals that generate the raster scan and also controls the X and Y address registers, which in turn defines memory location to be accessed next.

- Two registers are used to store the coordinates of the screen pixels, x register is used to store x-coordinate and y register is used to store y-coordinate.
- **The intensity value stored in the frame buffer for this pixel position is then retrieved, and used to set the intensity of the CRT beam.**
- Initially, the x-register is set to 0 and the y register is set to y_{max}. Then the x-register is incremented by 1, and the process repeated for the next pixel on the top scan line until $x = x_{max}$.
- This process is repeated for each pixel along the scan line. After the last pixel on the top scan line has been processed, the x-register is reset to 0, and the y-register is decremented by 1.
- This process is repeated for each successive scan line until the bottom scan line where y=0.

Disadvantages:

- Slow, since graphics function such as retrieving coordinates, and *scan conversion* (Digitizing a picture definition into a set of pixel intensities, reading screen coordinates) is done by the CPU.
- In this type of architecture, CPU digitizes a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer. ***This digitization process is called scan conversion.***

B. Raster Display System with Peripheral Display Processor:

- The raster scan with a peripheral display processor is a common architecture that avoids the disadvantage of simple raster scan system.
- Raster system architecture with a peripheral display processor is shown in figure below.

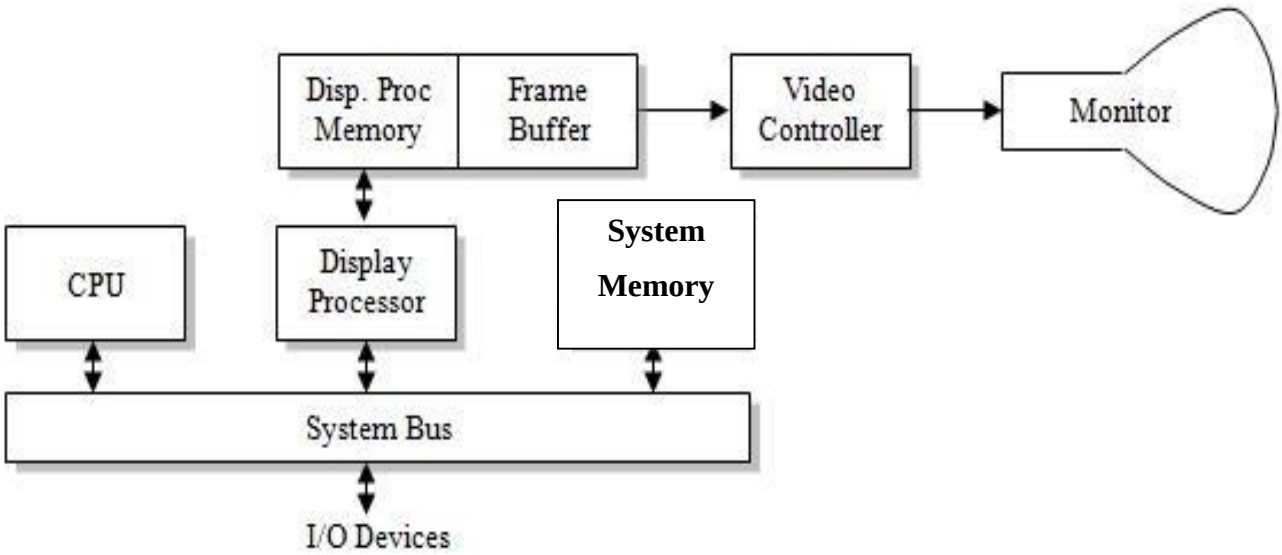


Fig 2.13: Architecture of raster graphics system with a display processor

- It includes a separate graphics processor called display processor or also called graphic controller or display co-processor. The purpose of the display processor is to free the CPU from graphics loads.

- Display processor performs **graphics functions such as scan conversion and retrieving screen coordinates**. A major task of the display processor is digitizing a picture definition given in an application program into a set of pixel intensity values for storage in the frame buffer. This digitization process is called scan conversion.
- The display processor has its own separate memory called display processor memory. Display processor memory holds ***data plus the program that perform scan conversion***.

2.3.2 Architecture of Random-scan (Vector) Systems

- A typical organization of a vector system is shown below.

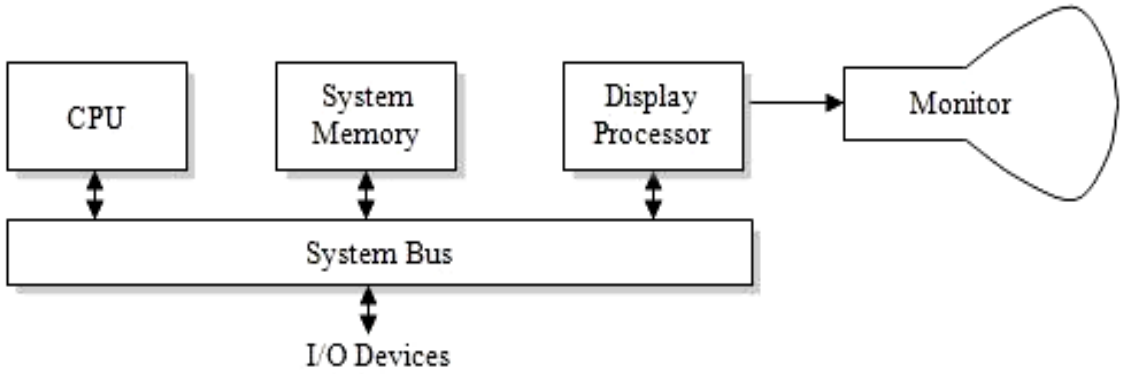


Fig 2.14: Architecture of random-scan system.

- Vector display system consists of additional processing unit along with CPU which is called the *display processor or graphics controller.*
- Picture definition is stored as a set of line drawing commands in the memory, called a display list. A display list (or display file) contains a series of graphics commands that define an output image.

2.4 Flat-Panel Displays

- Flat panel display refers to a class of video device that have reduced volume, weight and power requirement compared to a CRT.
- These emerging display technologies tend to replace CRT monitors. Current uses of flat-panel displays include TV monitors, calculators, pocket video games, laptops, displays in airlines and ads etc.
- Two categories of flat-panel displays:
  - **Emissive displays:**
    - Convert electrical energy into light.
    - ***Example: Plasma panels, electroluminescent displays and light-emitting diodes.***
  - **Non-emissive displays:**
    - Use optical effects to convert sunlight or light from other sources into graphics patterns.
    - ***Example: liquid-crystal displays.***

Plasma Panels

- Also called gas-charge displays. Plasma panels are constructed by filling the region between two glass plates with a mixture of gases usually neon.
- A series of vertical conducting ribbons is placed on one glass panel, and a set of horizontal ribbons is built into other glass panel.
- Firing voltage is applied to a pair of horizontal and vertical conductors. It causes the gas at the intersection of the two conductors to break down into glowing pattern.
- Picture definition is stored in the refresh buffer.

Light-emitting Diode (LED)

- In LED, a matrix of diodes is arranged to form the pixel positions in the display and picture definition is stored in a refresh buffer.
- Information is retrieved from the refresh buffer and converted to voltage levels that are applied to the diodes to produce the light patterns in the display.
- LED's have the screen refresh rate of 60 frames per second.

Liquid-crystal Displays (LCDs)

- Liquid crystal displays are the devices that produce a picture by passing light from light source through a liquid crystal material that transmit the light.
- Two glass plates are used. Rows of horizontal transparent conductors are built into one glass plate, and columns of vertical conductors are put into the other plate.
- Liquid crystal material is placed between these two glass plates. And intersection of two conductors defines the pixel position. Picture definition is stored in the refresh buffer, and the screen is refreshed at the rate of 60 frames per second. Liquid-crystal displays (LCDs) are commonly used in small systems, such as calculators and portable, laptop computers.

2.5 Graphics Software

Interactive graphics allow users to make change over the displayed objects. Several graphics software packages are now available. There are two general classifications for graphics software:

I. ***General programming packages***

II. ***Special-purpose application packages***

General programming packages: *contain graphics functions that can be used with high level programming languages such as C, FORTRAN, Java etc.* **Example** Open GL (Graphics library). A general-purpose graphics package provides users with a variety of functions for creating and manipulating pictures. These graphic functions include tools for generating picture components, setting color, selecting views, and applying transformations.

Special-purpose application packages: *specifically designed for particular applications. Maya, CINEMA 4D are particularly used for animations, different types of CAD applications are designed for medical and business purposes. These are primarily oriented to non-programmers.*

2.5.1 Graphics Standard

Graphics standards are documents produced by International Standards Organization (ISO) which define a common interface to interactive computer graphics for application programs. Many problems have been encountered due to the plate-form dependency of the graphic software's.

Why Graphics standards?

- So primary goal of standardized graphics software is portability. When graphics packages are designed with *standard graphics functions* (set of specifications that is independent of any programming languages), software can be easily moved from one H/W system to another. And it can be used in different implementations and applications.
- Production and manipulation of 2D-pictures in a way that does not depend on the system or graphical device used.

Primarily there are three graphics standards:

I. ***Graphical Kernel System (GKS):*** GKS was the first graphics software standard adopted by the international standards organization (ISO). It was originally designed as a 2-dimensional graphics package. GKS supports the grouping of logically related primitives such as; lines, polygons, character strings.

II. ***Programmer's Hierarchical Interactive Graphics System (PHIGS):*** It is an extension of GKS. Increased capabilities in object modeling, color specifications, surface rendering, and picture manipulations are provided in PHIGS. PHIGS include all primitives supported by GKS, in addition it also includes geometric transformations (***Scaling, Translation, and Rotation***).

III. ***PHIGS+:*** Extension of earlier PHIGS. 3D surface shading capabilities are added to the PHIGS.

Chapter 3

Scan conversion Algorithms

3.1 Introduction

- The basic building blocks for pictures are referred to as output primitives.
- Output primitives are the geometric structures that used to describe the shapes and colors of the objects.
- They include character strings, and geometric entities such as points, straight lines, curved lines, polygons, circles etc.
- Points and straight line segments are the most basic components of a picture.
- In raster scan systems, a picture is completely specified by the set of intensities for the pixel positions in the display. The process that converts picture definition into a set of pixel intensity values is called scan conversion. This is done by display processor.

3.2 Points and Lines

- With raster-scan system, a point can be plotted, by simply turning on the electron beam at that point.
  - putpixel(20, 20, RED)
- And a random-scan system stores the point plotting instructions in the display list file.
  - LDXA 100 Load data value 100 into the X register.
    LDYAP 450 Draw point at(100, 450)
- In raster scan systems line drawing is accomplished by calculating the intermediate positions along the line path between two specified endpoints.
- In random scan systems, line drawing is accomplished by retrieving line drawing commands from the display list.
- Scan lines are numbered consecutively from 0, starting at the bottom of the screen; and pixel columns are numbered from 0, from left to right across each scan line.

3.3 Line Drawing Algorithms

Three most widely used line drawing algorithms:

1. Direct use of Line Equation
2. Digital Differential Analyzer Algorithm(DDA)
3. Bresenham's Line Drawing Algorithm(BLA/BSA)

3.3.1 Direct use of Line Equation

The Cartesian slope equation of a straight line can be written as,

$y = m.x+b$ …….(i)

Where, m represents the slope of the line and b as the y–intercept.

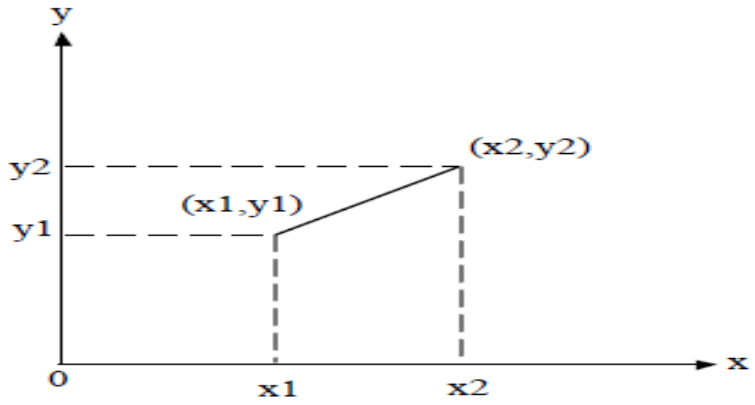


Fig 3.1: Line path between two endpoint(x_1, y_1), and (x_2, y_2).

Suppose (x_1, y_1) and(x_2,y_2) are the two end points of a line segment as shown in figure 3.1.

Then,

$m = (y_2-y_1)/(x_2-x_1)$ ……..(ii)

For any point (x_k, y_k), (i) becomes;

$y_k = m.x_k+b$ ……..(iii)

For any point (x_{k+1}, y_{k+1}), (i) becomes;

$y_{k+1} = m.x_{k+1}+b$ ……..(iv)

Now, subtracting (iii) from (iv);

$y_{k+1}- y_k = m(x_{k+1}-x_k)$ ……(v)

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

where Δx and Δy are x and y-increment respectively.

Therefore, $m = \Delta y/\Delta x$ ……..(vi)

This equation forms the basis for determining intermediate pixel positions. Calculate m from equation (ii) and test for following three cases:

Case I: For $|m| < 1$,

- $\triangle x$ can be set to small increment (Generally 1) and the corresponding $\triangle y$ calculated from equation (vi).

Case II: For $|m| > 1$,

- $\triangle y$ can be set to small increment (Generally 1) and the corresponding $\triangle x$ calculated from equation (vi).

Case III: For $|m| = 1$,

- Set both $\triangle x$ and $\triangle y$ to small increment i.e., $\triangle x = \triangle y$ = to small increment (Generally 1).

3.3.2 Digital Differential Analyzer (DDA) Algorithm

- The digital differential analyzer(DDA) is a scan conversion line drawing algorithm based on calculating Either x-increment or y-increment.
- This algorithm samples the line at unit interval in one-coordinate and determines corresponding integer values nearest the line path for the other co-ordinate.

Procedure:

The equation of the line is,

$Y = m.x+b$ ……(i)

$m = y_2-y_1/x_2-x_1$ ……(ii)

For any interval $\triangle x$, corresponding interval is given by $\triangle y = m.\triangle x$.

Case I:

If $|m| \leq 1$, we sample at unit x intervals i.e $\triangle x = 1$.

$x_{k+1} = x_k+1$ ……..(iii)

Then we compute each successive y-values, by setting $\triangle y = m$ (since, $m=\triangle y/\triangle x$ and $\triangle x=1$).

So, $y_{k+1} = y_k + m$ ……..(iv)

Case II:

If $|m| > 1$, we sample at unit y-intervals i.e $\triangle y = 1$ and compute each successive x-values.

Therefore, $1 = m.\triangle x$, and $\triangle x = 1/m$ (since, $m=\triangle y/\triangle x$ and $\triangle y=1$).

$x_{k+1} = x_k + 1/m$……..(v)

$y_{k+1} = y_k + 1$…….(vi)

The above equations are under the assumption that the lines are processed from left to right i.e. left end point is starting. For the reverse process i.e. if the line is to be processed form right to left then,

Case III:

$\triangle x = -1$ for line $|m| \leq 1$

Therefore,

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

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

Case IV:

$\triangle y = -1$ for line $|m| > 1$

Therefore,

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

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

Therefore, in general,

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

$x_{k+1} = x_k \pm 1$ for $|m| < 1$ and

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

$x_{k+1} = x_k \pm 1/m$ for $|m| >= 1$

Algorithm:

Step 1: Input the line endpoints and store the left endpoint in (x_1, y_1) and right endpoint in (x_2, y_2).

Step 2: Calculate the values of dx and dy, $dx = x_2 - x_1$, $dy = y_2 - y_1$.

Step 3: if(abs(dx) > abs(dy))

- *steplength* = abs(dx)

else

- *steplength* = abs(dy)

Step 4: Calculate the values of x-increment and y-increment.

- xIncrement = dx/ *steplength*
- yIncrment = dy/ *steplength*

Step 5: Set $x = x_1$ and $y = y_1$

Step 6: Plot(x, y).

Step 7: for k=1 to *steplength* do

- x = x + xIncrement
- y = y + yIncrement
- Perform round off, and plot each calculated (x, y) i.e. Plot(round(x), round(y)).

Step 8: End

Pros & Cons of DDA:

- Simple method, involves only integer additions.
- But involves, continuous round offs which can cause the calculated pixel positions to drift away from the actual line path.
- Furthermore, continuous rounding operations are time consuming.

Example: ***Digitize the line with endpoints (1, 5) and (7, 2) using DDA algorithm.***

Here, dx = 7-1=6, and dy = 2-5 = -3,

So, steplength = 6 (since abs(dx)>abs(dy)).

Therefore, xIncrement = dx/steplength = 6/6 = 1, and

yIncrement = dy/steplength = -3/6 = -1/2 = -0.5

Based on these values the intermediate pixel calculation is shown in the table below.

k	x_{k+1}	y_{k+1}	(x_{k+1}, y_{k+1})	Plot in screen (x_{k+1}, y_{k+1})
1	2	4.5	(2, 4.5)	(2, 5)
2	3	4	(3, 4)	(3, 4)
3	4	3.5	(4, 3.5)	(4,4)
4	5	3	(5,3)	(5,3)
5	6	2.5	(6,2.5)	(6,3)
6	7	2	(7,2)	(7,2)

Exercises:

1. ***Digitize the line with endpoints (1, -6) and (4, 4) using DDA algorithm.***
2. ***Digitize the line with endpoints (1, 6), (6, 10) using DDA algorithm.***
3. ***Trace DDA algorithm for line with endpoints (1, 2), (5, 6).***
4. ***Trace DDA algorithm for endpoints (1, 7), (6, 3).***

3.3.3 Bresenham's Line Drawing Algorithm (BSA)

An accurate and efficient raster line-drawing algorithm, developed by Bresenham's. It only uses integer calculations so can be adapted to display circles and other curves. It introduces the integer decision parameter to select the next appropriate pixel point.

Bresenham's complete algorithm:

1. Input the two end points, and store in (x_0, y_0), and (x_n, y_n) respectively.
2. Compute $\triangle x = |x_n - x_0|$ and $\triangle y = |y_n - y_0|$
3. If $x_n - x_0 < 0$ and $y_n - y_0 > 0$ or $x_n - x_0 > 0$ and $y_n - y_0 < 0$.
   Then set a = -1,
   else a = 1
4. If $\triangle x \geq \triangle y$
   i. If $x_0 > x_n$ then,
      $t = x_0$; $x_0 = x_n$; $x_n = t$
      $t = y_0$; $y_0 = y_n$; $y_n = t$
   ii. Find initial decision parameter $p_0 = 2\triangle y - \triangle x$
   iii. Plot the first pixel (x_0, y_0).
   iv. Repeat the following, starting from k=0.
      If $p_k < 0$ then,
         -plot pixel (x_k+1, y_k)
         -set $p_{k+1} = p_k + 2\triangle y$
      else,
         -plot pixel $(x_k+1, y_k + a)$
         -set $p_{k+1} = p_k + 2\triangle y - 2\triangle x$
   v. Repeat iv, $\triangle x$ times.
5. Else
   i. Check if $(y_0 > y_n)$ then,
      $t = x_0$; $x_0 = x_n$; $x_n = t$
      $t = y_0$; $y_0 = y_n$; $y_n = t$
   ii. Find initial decision parameters. $p_0 = 2\triangle x - \triangle y$
   iii. Plot the first point (x_0, y_0)
   iv. Repeat the following, starting from k=0.
      If $p_k < 0$ then,
         -plot pixel (x_k, y_k+1)
         -set $p_{k+1} = p_k + 2\triangle x$.
      Else,

-plot pixel (x_k + a, y_k+1)

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

v. Repeat iv, $\triangle y$ times.

6. END.

Pros and Cons of BSA

- ❖ Involves simple integer calculations, so does not need to perform round off operation.
- ❖ It can be used to generate circles and other curves.
- ❖ Additional parameter i.e., decision parameter must be calculated at each step.

Exercises:

1. Digitize the line with endpoints (1, -6) and (4, 4) using BSA algorithm.
2. Digitize the line with endpoints (1, 6), (6, 10) using BSA algorithm.
3. Trace BSA for line with endpoints (15, 15), (10, 11).
4. Trace BSA for line with endpoints (20, 10), (30, 18).
5. Trace BSA for endpoints (5, 10), (10, 7).

3.4 Circle

- ❖ A circle is defined as a set of points that are all at a given distance 'r' from the center position (x_c, y_c).
- ❖ The general circle equation can be written as;

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

3.4.1 Properties of circle

Symmetry in quadrants.

- ➢ The shape of the circle is similar in each quadrant. Thus by calculating points in one quadrant we can calculate points in other three quadrants.

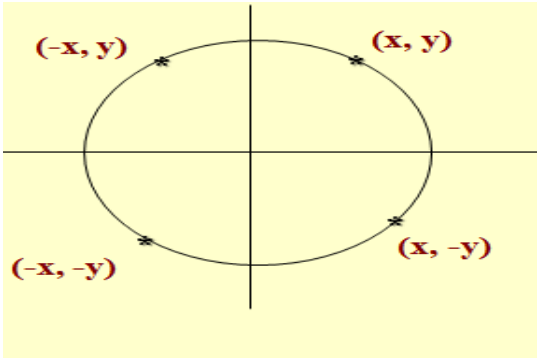


Fig: Symmetry of a circle in its quadrants.

Symmetry in octants.

- The shape of the circle is similar in each octant. Thus by calculating points in one octant we can calculate points in other seven octants. If the point (x, y) is on the circle, then we can trivially compute seven other points on the circle.
- Therefore, we need to compute only one 45° segment to determine the circle, as shown in ***figure below***.
- By taking advantage of circle symmetry in octants, we can generate all pixel positions around a circle by calculating only the points within the sector from x = 0 to y = x.

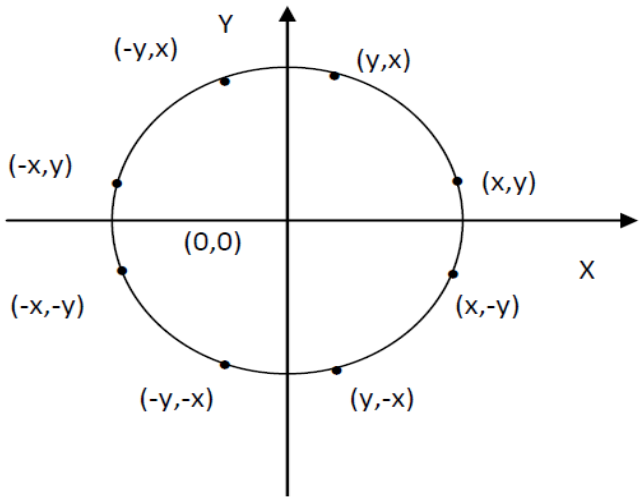


Fig: Symmetry of a circle in its octants (8 way symmetry).

3.4.2 Methods to draw circle:

- Direct method
- Trigonometric method

Mid-point Circle method.

3.4.2.1 Direct method

- Use circle equation to calculate the positions of points on a circle circumference. Increase x-values by 1 from $x_c - r$ to $x_c + r$ and calculate associated y-value using;

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

3.4.2.2 Trigonometric method

- Use circle equation in polar form

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

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

❖ To draw circle using these polar co-ordinates approach, just increment angle starting from 0 to 2π. Compute (x, y) position corresponding to increment angle.

3.4.2.3 Midpoint Circle Algorithm/Bresenham's Circle Drawing Algorithm

1. Input radius r and circle center (x_c, y_c).
2. Obtain the first point on the circumference by assuming that circle is centered on the origin i.e., $(x_0, y_0) = (0, r)$.
3. Calculate the initial decision parameter as $p_0 = 5/4 - r$. (If **r is integer, then set $p_0 = 1-r$.**)
4. Repeat untill $x \geq y$, at each x_k position, starting at k=0, performing the following
   - If $p_k < 0$
     - select pixel (x_k+1, y_k)
     - set $p_{k+1} = p_k + 2x_{k+1} + 1$

     else
     - select pixel (x_k+1, y_k-1)
     - set $p_{k+1} = p_k + 2x_{k+1} - 2y_{k+1} + 1$
   - Determine symmetry points in the other seven octants.
   - Move each calculated pixels positions (x, y) in to circle path centered at (x_c, y_c) and plot pixels as;
     - $x = x_c + x$,
     - $y = y_c + y$.
5. END

Example:

Digitize a circle $(x-2)^2 + (y-3)^2 = 25$

Solution:

Center $C(x_c, y_c) = (2, 3)$

Radius $(r) = 5$

1^{st} pixel $(0, 5)$

$p_0 = 1-r$

$= 1-5$

$= -4$

Successive decision parameter, and pixel calculations are shown in the table below.

k	**p_k**	**$(x_{k+1}\ y_{k+1})$**	**$2x_{k+1}$**	**$2y_{k+1}$**
0	-4	(1, 5)	2	10
1	-1	(2, 5)	4	10
2	4	(3, 4)	6	8
3	3	(4, 3)	8	6

Now, we need to determine symmetry points, and actual points when circle center is at (2, 3).

Determined pixel	**Symmetry Points**	**Actual pixel (x_c+x, y_c+y)**
(1, 5)	(1, 5)	(3, 8) (x, y)
	(-1, 5)	(1, 8) (-x, y)
	(-1, -5)	(1, -2) (-x, -y)
	(1,-5)	(3, -2) (x, -y)
	(5, 1)	(7, 4) (y, x)
	(5, -1)	(7, 2) (y, -x)
	(-5, -1)	(-3, 2) (-y, -x)
	(-5, 1)	(-3, 4) (-y, x)
(2, 5)	(2, 5)	(4, 8)
	(-2, 5)	(0, 8)
	(-2,-5)	(0, -2)
	(2, -5)	(4, -2)
	(5, 2)	(7, 5)
	(5, -2)	(7, 1)
	(-5, -2)	(-3, 1)
	(-5, 2)	(-3, 5)

(3, 4)	(3, 4)	(5, 7)
	(-3, 4)	(-1, 7)
	(-3, -4)	(-1, -1)
	(3, -4)	(5, -1)
	(4, 3)	(6, 6)
	(4, -3)	(6, 0)
	(-4, -3)	(-2, 0)
	(-4, 3)	(-2, 6)

(4, 3)	(4, 3)	(6, 6)
	(-4, 3)	(-2, 6)
	(-4, -3)	(-2, 0)
	(4, -3)	(6, 0)
	(3, 4)	(5, 7)
	(3, -4)	(5, -1)
	(-3, -4)	(-1, -1)
	(-3, 4)	(-1, 7)

Exercises:

1. ***Digitize $x^2 + y^2 = 100$ in first octant.***
2. ***Digitize a circle $(x-2)^2 + (y-3)^2 = 25$.***
3. ***Draw a circle having radius 3 units and center at (3, 2).***
4. ***Draw a circle with radius 10 units and center at origin.***
5. ***Draw a circle with radius 5 units and center at (4, -2).***

3.5 Ellipse

- An ellipse is an elongated circle.
- Therefore elliptical curves can be generated by modifying circle drawing procedures.
- The ellipse, like the circle, shows symmetry. An ellipse is symmetric in quadrants. So if one quadrant is generated then other three parts can be easily generated.
- An ellipse can be represented as:

$$\frac{(x-h)^2}{a^2}+\frac{(y-k)^2}{b^2}=1$$

Where (h, k) = ellipse center.

a = length of semi-major axis.

b = length of semi-minor axis.

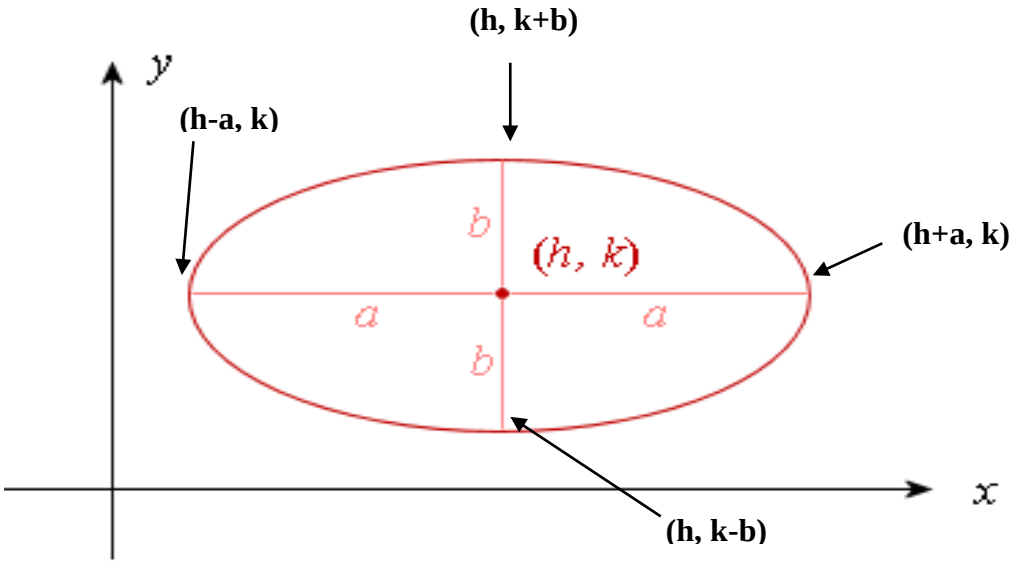


Fig: Ellipse centered at (h, k) with semi-major axis a and semi-minor axis b.

An ellipse centered at origin (0,0) can be represented as:

$$x^2/a^2 + y^2/b^2 = 1$$

3.5.1 Ellipse generating algorithms

- Direct Method
- Trigonometric Method
- Midpoint Ellipse Algorithm

1. Direct Method

An ellipse can be represented as:

$$\frac{(x-h)^2}{a^2}+\frac{(y-k)^2}{b^2}=1 \quad \text{.....1}$$

- Solving this equation, we get

$$y = k \pm b\sqrt{1-\frac{x-h^2}{a^2}} \quad \text{.....2}$$

- The value of x is incremented in units of 1 from (h-a) to (h+a), and corresponding y is evaluated from equation (2).
- Plotting these computed points we can get the ellipse.

Algorithm for direct method:

1. Input the center of ellipse (h, k), semi-major and semi-minor axis a and b respectively.
2. For each x position starting from h-a and stepping unit interval along x-direction, compute corresponding y positions as

$$y = k \pm b\sqrt{1-\frac{x-h^2}{a^2}}$$

3. Plot the point (x, y).
4. Repeat step 2 to 3 until $x \geq h+a$.

2. Trigonometric Method

Using polar coordinates an ellipse can be represented as;

$x = a \cos \theta + h$

$y = b \sin \theta + k$

Where, (x, y) = current coordinate

a = length of semi-major axis

b = length of semi-minor axis

θ = current angle, measured in radians from 0 to 2π.

(h, k) = ellipse center.

In this method, θ is incremented from 0 to 2π and we compute successive values of x and y.

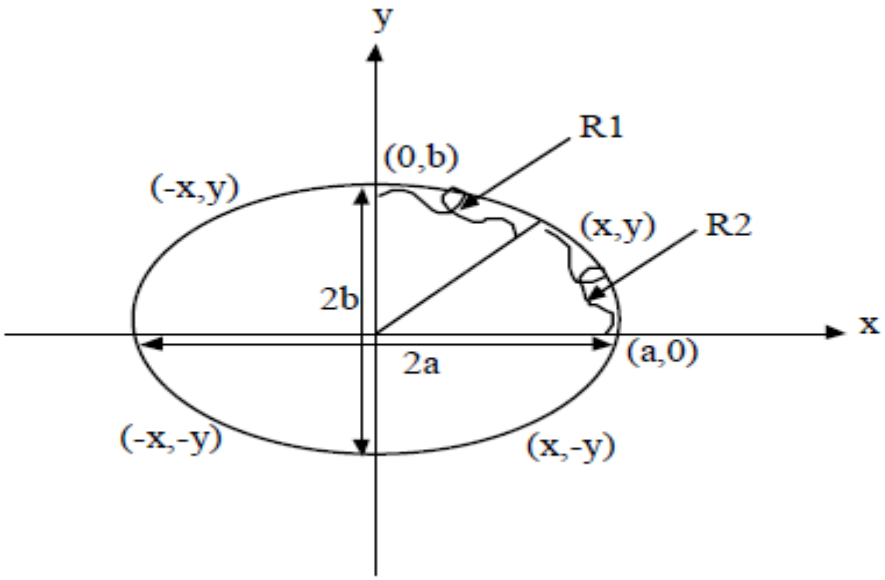
Algorithm for trigonometric method:

1. Input the center of ellipse (h, k), semi-major and semi-minor axis a and b respectively.
2. Starting from angle θ = 0 compute boundary point of ellipse as

   $x = a \cos \theta + h$

   $y = b \sin \theta + k$
3. Plot the point(x, y).
4. Repeat until $\theta \geq 2\pi$.

3. Midpoint Ellipse Algorithm



- The algorithm is applied throughout the 1st quadrant according to the slope of the ellipse.
- First quadrant is divided into two parts, region-1 (R1) and region-2 (R2). These regions are formed by considering the slope of the curve.
- For the region (R1) where the slope of the curve is less than -1. We process by taking unit steps in x-direction and find corresponding y.

- And for the R2 where the slope is greater than -1 we take unit steps in y direction and find corresponding x. For R2, the initial point is the last calculated point in R1.
- The ellipse slope is calculated from equation :

  $x^2/a^2 + y^2/b_2 = 1$

  Differentiating both sides w.r to x

  $2x/a^2 + 2y/b^2 . dy/dx = 0$

  $dy/dx = -2b^2x/2a^2y$

  At the boundary region R1 and region R2, $dy/dx = -1$ and $2b^2x = 2a^2y$ at the boundary region. Therefore, we move out of region 1 (R1) when $2b^2x \geq 2a^2y$.

Algorithm:

1. Input center (h, k), semi-major and semi-minor axis length as a and b.
2. Obtain the first point on an ellipse by assuming that ellipse is centered on the origin i.e.

   $(x_0, y_0) = (0, b)$.
3. Compute initial decision parameter for region R1 as,

   $p1_0 = b^2 - a^2b + a^2/4$.
4. Starting at k = 0, repeat the following till $2b^2x_k < 2a^2y_k$, at each x_k position in R1, perform the following test:
   - If $p1_k < 0$
     - Select pixel(x_k+1, y_k)
     - Set $p1_{k+1} = p1_k + 2b^2x_{k+1} + b^2$
   - Otherwise
     - Select pixel(x_k+1, y_k-1)
     - Set $p1_{k+1} = p1_k + 2b^2x_{k+1} - 2a^2y_{k+1} + b^2$
   - Determine the symmetry points in other 3 quadrants.
   - Move each calculated point (x_k, y_k) on to the centered (h, k) ellipse path as

$x_k = x_k + h;$

$y_k = y_k + k$

5. Calculate value of the initial decision parameter at region R2 using last calculated point say (x_0, y_0) in R1 as;

$p2_0 = b^2(x+1/2)^2 + a^2 (y-1)^2 – a^2b^2$

6. Repeat the following till y>0, at each x_k position in R2, starting at k = 0, perform the following test:

- If $p2_k>0$
    - Select pixel(x_k, y_k -1)
    - Set $p2_{k+1} = p2_k - 2a^2y_{k+1} + a^2$
- Otherwise
    - Select pixel(x_k+1, y_k-1)
    - Set $p2_{k+1} = p2_k + 2a^2x_{k+1} – 2b^2y_{k+1} + a^2$
- Determine the symmetry points in other 3 quadrants.
- Move each calculated point (x_k, y_k) on to the centered (h, k) ellipse path as

$x_k = x_k + h;$

$y_k = y_k + k$

Source Code:

```
#include<stdio.h>
 #include<conio.h>
 #include<graphics.h>
 void ellipse(int xc,int yc,int rx,int ry)
 {
   int gm=DETECT,gd;
   int x, y, p;
   initgraph(&gm,&gd,"C:\\TC\\BGI");
   x=0;
   y=ry;
   p=(ry*ry)-(rx*rx*ry)+((rx*rx)/4);
   while((2*x*ry*ry)<(2*y*rx*rx))
   {
       putpixel(xc+x,yc-y,WHITE);
       putpixel(xc-x,yc+y,WHITE);
```

```
    putpixel(xc+x,yc+y,WHITE);
    putpixel(xc-x,yc-y,WHITE);
    if(p<0)
    {
                    x=x+1;
                    p=p+(2*ry*ry*x)+(ry*ry);
    }
    else
    {
                    x=x+1;
                    y=y-1;
                    p=p+(2*ry*ry*x+ry*ry)-(2*rx*rx*y);
    }
}
p=((float)x+0.5)*((float)x+0.5)*ry*ry+(y-1)*(y-1)*rx*rx-rx*rx*ry*ry;   //calculating initial decision
                                                                        parameter for Region 2.


     while(y>=0)
{
    putpixel(xc+x,yc-y,WHITE);
    putpixel(xc-x,yc+y,WHITE);
    putpixel(xc+x,yc+y,WHITE);
    putpixel(xc-x,yc-y,WHITE);
    if(p>0)
    {
                    y=y-1;
                    p=p-(2*rx*rx*y)+(rx*rx);
    }
    else
    {
                    y=y-1;
                    x=x+1;
                    p=p+(2*ry*ry*x)-(2*rx*rx*y)-(rx*rx);
```

```
    }
  }
  getch();
  closegraph();
}
int main()
{
  int xc,yc,rx,ry;
  printf("Enter Xc=");
  scanf("%d",&xc);
  printf("Enter Yc=");
  scanf("%d",&yc);
  printf("Enter Rx=");
  scanf("%d",&rx);
  printf("Enter Ry=");
  scanf("%d",&ry);
  ellipse(xc,yc,rx,ry);
  getch();
  return 0; }
```

Example:

Digitize an ellipse $(x-2)^2/64 + (y+5)^2/36 = 1$, Using midpoint algorithm.

Solution:

Center(h, k) = (2,-5), a = 8, b = 6.

Let us assume that ellipse is centered on the coordinate origin(0, 0).

For region R1,

First pixel is (0, 6).

Now initial decision parameter,

$p1_0 = b^2 - a^2b + a^2/4$

$= 36 - 64 \times 6 + 64/4$

$p1_0 = -332$

Successive decision parameter values and pixel positions can be calculated as follows;

k	**p_k**	**(x_{k+1}, y_{k+1})**	**$2b^2x_{k+1}$**	**$2a^2y_{k+1}$**
0	-332	(1, 6)	72	768
1	-224	(2, 6)	144	768
2	-44	(3, 6)	216	768

3	-208	(4, 5)	288	640
4	-108	(5, 5)	360	640
5	288	(6, 4)	432	512
6	244	(7, 3)	504	384

We now move out of region 1, since $2b^2x_k < 2a^2y_k$.

For region R2,

The initial point is $(x_0, y_0) = (7, 3)$.

And the initial decision parameter is

$$p2_0 = b^2(x+1/2)^2 + a^2(y-1)^2 - a^2b^2$$
$$= 36(7+1/2)^2 + 64(3-1)2 - 36\times 64$$
$$= 36\times 225/4 + 64\times 4 - 36\times 64$$
$$p2_0 = -151$$

The remaining positions along the ellipse path in the first quadrant are then calculated as follows;

k	**p_k**	**(x_{k+1}, y_{k+1})**	**$2b^2x_{k+1}$**	**$2a^2y_{k+1}$**
0	-151	(8, 2)	576	256
1	233	(8, 1)	576	128
2	745	(8, 0)	-	-

Thus we have obtained points along the ellipse path in the first quadrant. By using the property of symmetry we can determine the symmetry points in other 3 quadrants.

Finally, for each (x, y) we need to perform;

$$x = x + 2$$
$$y = y - 5.$$

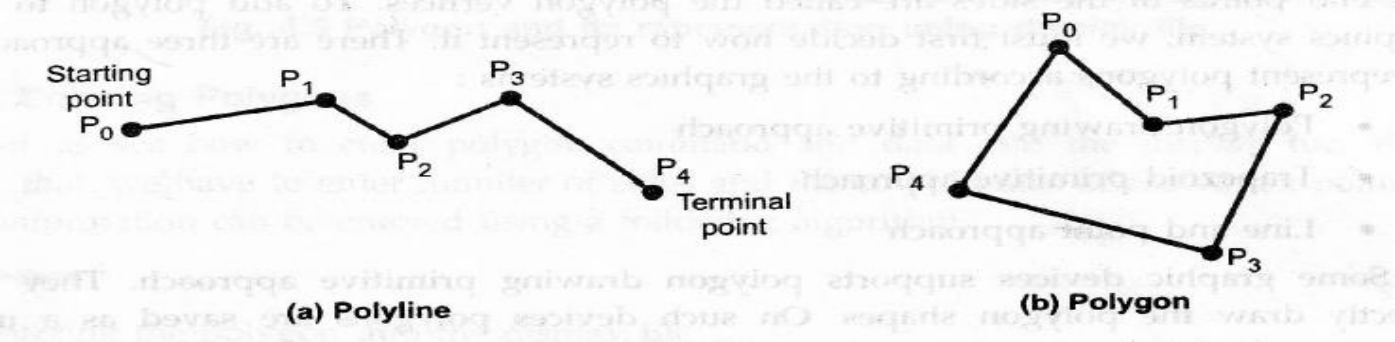
3.6 Filled Area Primitives

In this part, we will focus on study of different types of polygons, their representation and filling algorithms for them. Filling is the process of "coloring in" a fixed area or region.

3.6.1 Introduction

A polyline is a chain of connected line segments. It is specified by giving the vertices (nodes) P0, P1, P2... and so on. The first vertex is called the initial or starting point and the last vertex is called the final or terminal point, as shown in the figure(a).

When starting point and terminal point of any polyline is same, i.e. when polyline is closed then it is called polygon. This is illustrated in figure (b).



Triangle is the simplest form of polygon having three sides and three vertices.

3.6.2 Types of Polygons

❖ The classification of polygons is based on where the line segment joining any two points within the polygon is going to lie. There are two types of polygons:

- Convex
  - If the line segment joining any two interior points of the polygon lies completely inside the polygon then it is called convex polygon.

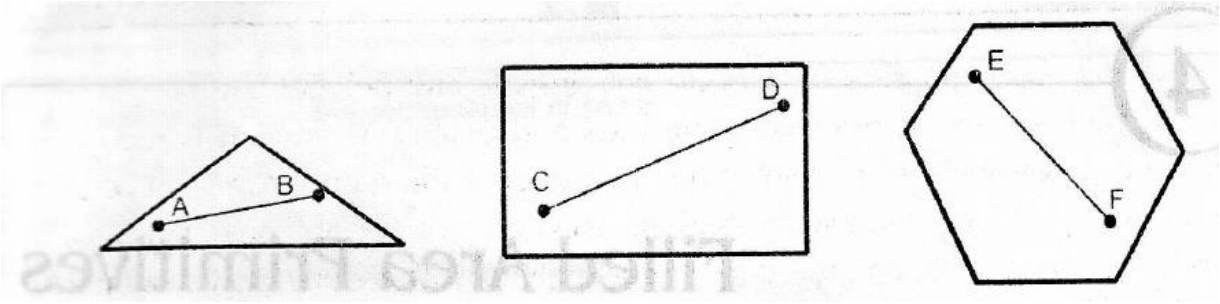


Figure: Convex Polygon

- Concave.
  - The polygons that are not convex are called concave polygons i.e., if the line segment joining any two interior points of the polygon is not completely inside the polygon then it is called concave polygon.

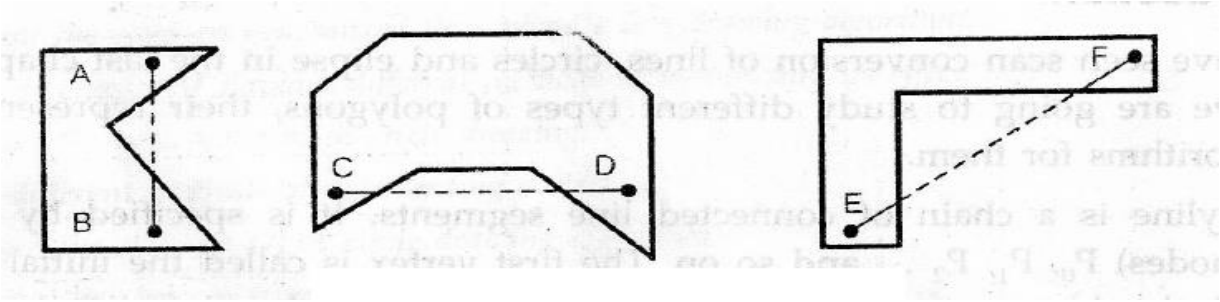


Figure: Concave Polygons

3.6.3 Representation of Polygons

There are various methods to represent polygons in a graphics system. Some graphic devices support polygon drawing primitive approach. They can directly draw the polygon shapes.

In C, drawpoly(n, int *points) function is used to draw polygons i.e. *triangle, rectangle, pentagon, hexagon* etc. Here n is the number of vertices in a polygon.

Example:

```
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};
drawpoly(4, points);
```

3.6.4 Polygon Filling

- *Polygon filling is the process of highlighting all the pixels which lie inside the polygon with any color other than background color.*
- *In other words, it is the process of coloring the area of a polygon.* An area or a region is defined as a collection of pixels.
- For area filling polygons are standard output primitive in general graphics package since polygons are easier to process/fill since they have linear boundaries.
- **There are two basic approaches used to fill the polygon**.
  - One way to fill a polygon is to start from a given "seed", point known to be inside the polygon and highlight outward from this point i.e. neighboring pixels until we encounter the boundary pixels. This approach is called **seed fill** because color flows from the seed pixel until reaching the polygon boundary. Suitable for complex objects.
  - Another approach to fill the polygon is to apply the inside test i.e. to check whether the pixel is inside the polygon or outside the polygon and then highlight pixels which lie inside the polygon. This approach is known as **scan-line algorithm**. It avoids the need for a seed pixel but it requires some additional computation. Suitable for simple objects, polygons, circles etc.

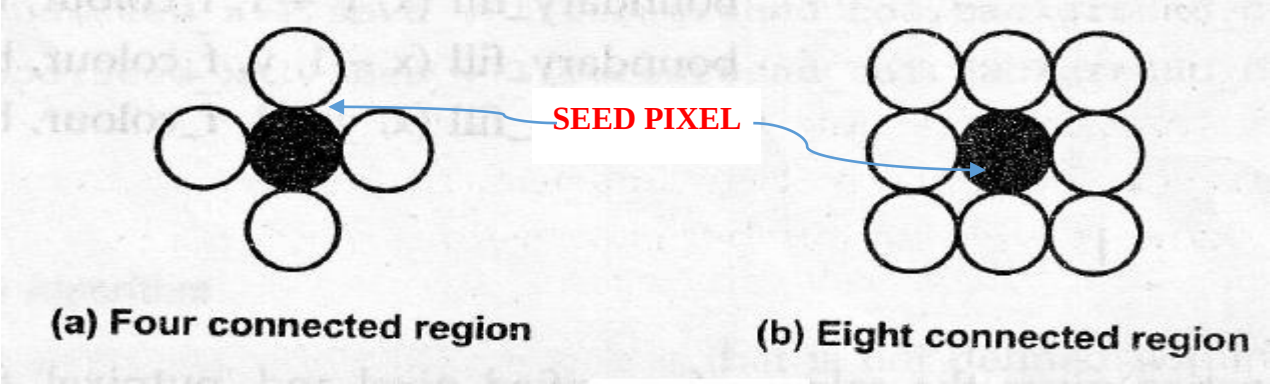
3.6.5 Seed Fill Algorithm

- **Principle:** "Start at a sample pixel called as a seed pixel from the area, fill the specified color value until the polygon boundary".
- The seed fill algorithm is further classified as **flood fill algorithm and boundary fill algorithm**.

Boundary-fill Algorithm

- This is a recursive algorithm that begins with a starting interior pixel called a seed, and continues painting towards the boundary. The algorithm checks to see if this pixel is a boundary pixel or has already filled. If not it fills the pixel and makes a recursive call to itself using each and every neighboring pixel as a new seed.
- 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 fill color that should be different for this algorithm to work. In this method, edges of the polygons are drawn first **in a single color**.
- An area filling starts from a random pixel called as seed, inside a region and continues painting towards the boundary. This process continues outwards pixel by pixel until the boundary color is reached.

- A boundary-fill procedure accepts as input the co-ordinates of an interior point (x, y), a fill color, and a boundary color.
- Starting from (x, y), the procedure tests neighboring positions to determine whether they are of boundary color or has already filled. If not, they are painted with the fill color, and their neighbors are tested.
- This process continues until all the pixels up to the boundary are tested.
- There are two methods for proceeding to the neighbouring pixels from the seed pixel:
  - 4 adjacent pixel may be tested (4-connected)
    - 4 neighboring pixels i.e., left, right, up and down are tested.
  - 8 adjacent pixel may be tested (8-connected)
    - 8 neighboring pixels are tested i.e., left, right, up, down, and diagonal pixels are also tested.



The following procedures illustrate the recursive method for filling 4-connected region and 8-connected with color specified in parameter fill color (fill_color) up to a boundary color specified with parameter boundary color (b_color).

Boundary fill 4-Connected

```
void Boundary_fill4(int x, int y, int b_color, int fill_color)
{
        int value = getpixel (x, y);
        if (value! =b_color && value!=fill_color)
        {
                putpixel (x, y, fill_color);
                Boundary_fill 4 (x-1, y, b_color, fill_color);
                Boundary_fill 4 (x+1, y, b_color, fill_color);
                Boundary_fill 4 (x,y-1, b_color, fill_color);
                Boundary_fill 4 (x,y+1, b_color, fill_color);
        }
}
```

Note: 'getpixel' function gives the color of specified pixel and 'putpixel' function draws the pixel with specified color.

In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm. This is illustrated in Figure. Here, a 4-connected algorithm produces the partial fill.

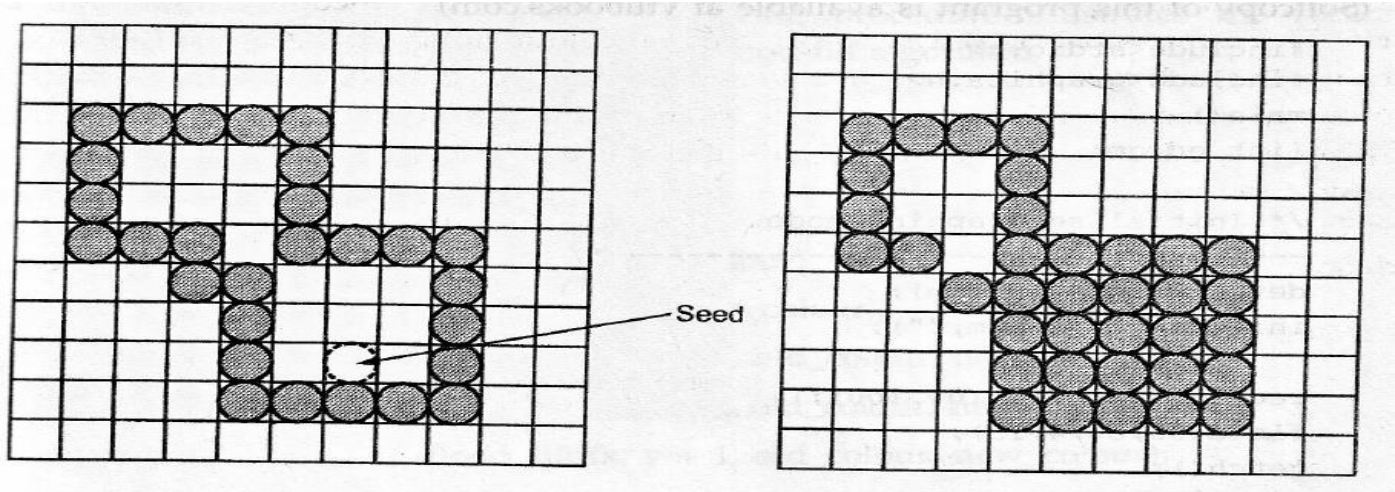


Figure: particular filling resulted using 4-connected algorithm

Boundary fill 8-connected

```
void Boundary-fill8(int x,int y,int b_color, int fill_color)
{
        Int current  = getpixel (x, y);
        if (current !=b_color && current!=fill_color)
        (       putpixel (x,y,fill_color);
                Boundary_fill8(x-1, y, b_color,fill_color);
                Boundary_fill8(x+1, y, b_color, fill_color);
                Boundary_fill8(x, y-1, b_color, fill_color);
                Boundary_fill8(x, y+1, b_color, fill_color);
                Boundary_fill8(x-1, y-1, b_color,fill_color);
                Boundary_fill8(x-1,y+1,b_color,fill_color);
                Boundary_fill8(x+1,y-1,b_color,fill_color);
                Boundary_fill8(x+1,y+1,b_color,fill_color);
        }
}
```

Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.

Flood Fill Algorithm

- Flood fill algorithm is used to fill the area bounded by different color boundaries i.e., it is required to fill in an area that is not defined within a single color boundary.
- Basic concept of a flood-fill algorithm is to fill areas by replacing a specified interior color by fill color instead of searching for a boundary 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.
- Here we start with some seed and examine the neighboring pixels.
- We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with desired fill-color.

- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.
- The following procedure illustrates the recursive method for filling 8-connected region using flood-fill algorithm.

Procedure for Flood-fill 8-connected:

```
void flood_fill (int x, int y, int old_color, int new_color)
{
        int value = getpixel (x, y);
        if (value ==old_color)
        {
                putpixel (x, y, new_color);
                flood_fill (x + 1, y, old_color, new_color);
                flood_fill (x - 1, y, old_color, new_color);
                flood_fill (x, y + 1, old_color, new_color);
                flood_fill (x, y - 1, old_color, new_colour);
                flood_fill (x + 1, y + 1, old_color, new_color);
                flood_fill (x - 1, y - 1, old_color, new_color);
                flood_fill (x + 1, y - 1, old_color, new_color);
                flood_fill (x - 1, y + 1, old-color, new_color);
```

```
    }
}
```

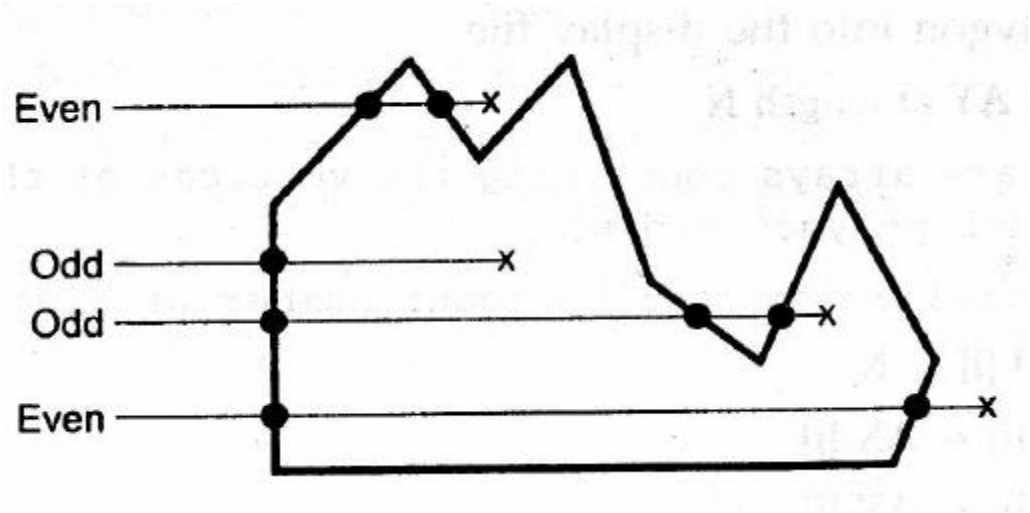
Note: this procedure is for 8-connected; similarly you can write procedure for Flood-fill 4-connected.

Problems with Seed-fill algorithms:

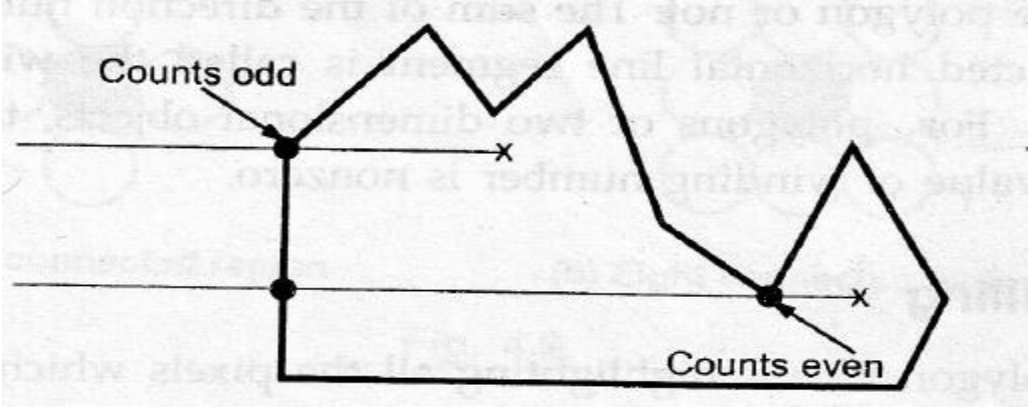
- Recursive algorithm for seed fill methods have got two difficulties:
  - The first difficulty is that if some inside pixels are already displayed in fill color then recursive branch terminates, leaving further internal pixels unfilled.
  - Another difficulty with recursive seed fill methods is that it cannot be used for large polygons. This is because recursive seed fill procedures require stacking of neighboring points and in case of large polygons stack space may be insufficient for stacking of neighboring points.

3.6.6 Inside and Outside Test of Polygon

- Area filling algorithms and other graphics processes often need to identify interior and exterior region of objects.
- To determine whether or not a point is inside of a polygon, even-odd method.
  - **Even-odd method(Odd-parity rule)**
    - In this method, we draw a line segment from a point X(point in question) and to a point known to be outside the polygon.
    - Now count the intersections of the line segment with the polygon boundary.
    - If there are an odd number of intersections then the point X is inside; otherwise it is outside.



    - If the intersection point is vertex of the polygon then we have to look at the other endpoints of the two segments which meet at this vertex.
    - If these points lie on the same side of the constructed line, then the point X counts as an even number of intersections.
    - If they lie on opposite sides of the constructed line, then the point is counted as a single intersection.



3.6.7 Scan-line Algorithm

- A scan-line algorithm fills horizontal pixels across scan lines, instead of proceeding to 4-connected or 8-connected neighboring points.
- This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections.
- **Approach:**
  - For each scan-line crossing a polygon, this algorithm computes the intersection points of the scan line with the polygon edges.
  - These intersection points are then sorted from left to right (i.e., increasing value of x coordinate), and the corresponding positions between each intersection pair are filled by the specified color (shown in figure below).
  - When scan line intersects an edge of polygon, start to color each pixel (because now we're inside the polygon), when intersects another edge, stop coloring.
  - This procedure is repeated until complete polygon is filled.

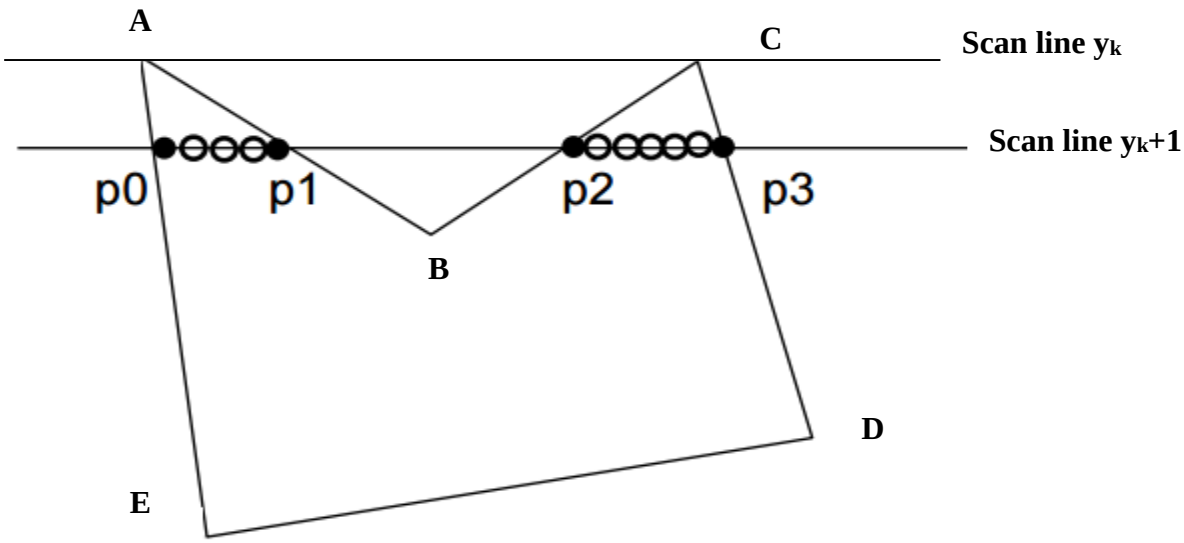


Figure: Showing intersection points p0, p1, p2, p3 and intersection pair (p0 –> p1, p2–> p3). The scan line algorithm first finds the largest and smallest y values of the polygon. It then starts with the largest y value and works its way down, scanning from left to right.

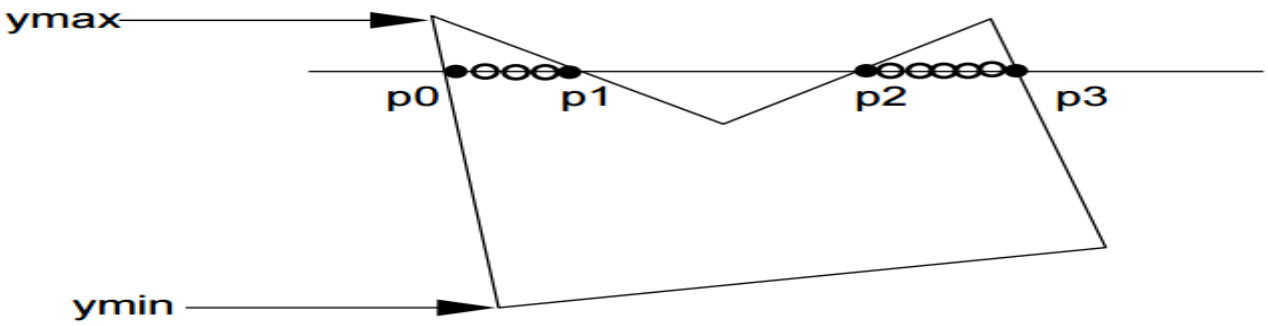
- The important task in the scan line algorithm is to find the intersection points of the scan line with the polygon boundary.
- To find intersection points;
  - Find the slope m for each polygon boundary (i.e., each edge) using vertex coordinates. If A(x_1, y_1) and E(x_2, y_2) then slope for polygon boundary AE can be calculated as follows;
    - $m = (y_2-y_1)/(x_2-x_1)$
  - We can find the intersection point on the lower scan line if the intersection point for current scan line is known. Let (x_k, y_k) be the intersection point for the current scan line y_k.
    - Scanning starts from top to bottom, y-coordinates between the two scan line changes by 1
      - i.e., $y_{k+1} = y_k - 1$.
    - And x coordinate can be calculated as,
      - $x_{k+1} = x_k - 1/m$

Algorithm:

Step1: Read the number of vertices of polygon, n.

Step2: Read the x and y coordinates of all vertices.

Step3: Find the minimum and maximum value of y and store as y_{min} and y_{max} respectively.



Step4: For y = y_{max} to y_{min}

- For each scan line
  - Find the intersection points p0, p1, p2, …, pn.
  - Find the coordinates of intersection points p0, p1, p2, …,pn.
  - Sort the intersection point in the increasing order of x-coordinate i.e.[p0, p1, p2,…,pn].
  - Fill all those pair of coordinates that are inside polygon, using even-odd method, and ignore the alternate pairs.

Step5: END.

3.6.8 Filling Rectangles

Scan line method can be used to fill a rectangle.

for ($y=y_{min}$ to $y= y_{max}$ of the rectangle) //by scan line

    for($x = x=$ to $x= x_{max}$) //by pixels

    putpixel(x, y, fill_color)

3.6.9 Filling Curved Boundary Areas

- Scan line fill of regions with curved boundaries requires more work than polygon filling, since intersection calculations now involve non-linear boundaries.
- For simple curves such as circles or ellipses, performing a scan line fill is straightforward process.
- We only need to calculate the two scan-line intersections on opposite sides of the curve.
- Then simply fill the horizontal spans of pixel between the boundary points on opposite side of curve.
- Symmetries between quadrants are used to reduce the boundary calculation.

Interior fill of an elliptical arc.

Chapter 4

2D Geometrical Transformations

4.1 What is Transformation?

The operations that are applied to geometrical description of an object to change its position, orientation, shape or size are called geometric transformations.

Types:

- **Rigid body transformation** (transformation without deformation in shape and size.)
  - *Translation*
  - *Rotation*
  - *Reflection.*

- **Non rigid body transformation** (transformation with change in shape/size.)
  - *Scaling*
  - *Shearing.*

4.2 Why Transformations?

In computer graphics, transformations of 2D objects are essential to many graphics applications. Transformations are needed to manipulate the initially created object and to display the modified object without redrawing it.

4.3 Basic Transformations:

1. *Translation*
2. *Rotation*
3. *Scaling.*

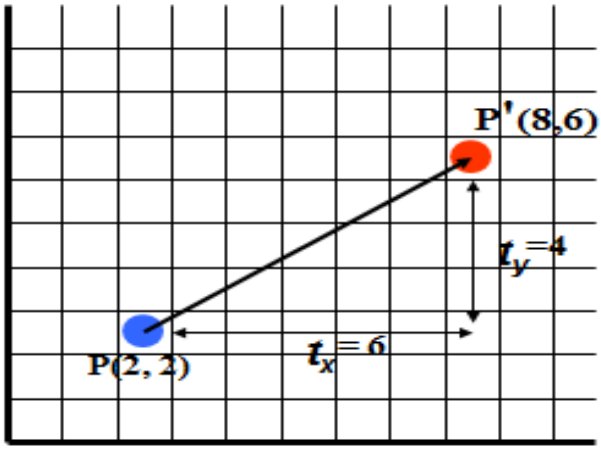
Other Transformation

1. *Reflection*
2. *Shearing*

Rotation, Translation and scaling are three major transformations that are extensively used by all graphical packages or graphical subroutines in applications. Other than these, reflection and shearing transformations are also used by some graphical packages.

4.3.1 Translation

Changing the co-ordinate position along a straight line is called translation. A translation moves all points in an object along the same straight-line path to new positions. The path is represented by a vector, called the *translation or shift vector*. If P(x, y) is translated to a position P'(x', y') by t_x units parallel to x-axis and t_y units parallel to y axis then,



$$x' = x + t_x$$

$$y' = y + t_y$$

In matrix form:

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

i.e. **P' = P + T,** where T is transformation matrix.

4.3.2 Rotation

Changing the co-ordinate position along a circular path is called rotation. A rotation repositions all points in an object along a circular path. *The point about to which object is to be rotated is called pivot point.*

About origin:

If P(x, y) is rotated to a new position P' (x, y') in anti-clockwise direction by an angle θ, then (x', y') can be calculated as following.

Let the angle made by line OP with x-axis is α and the radius of circulation path is r then,

$x = r\cos\alpha$

$y = r\sin\alpha$

Also,

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

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

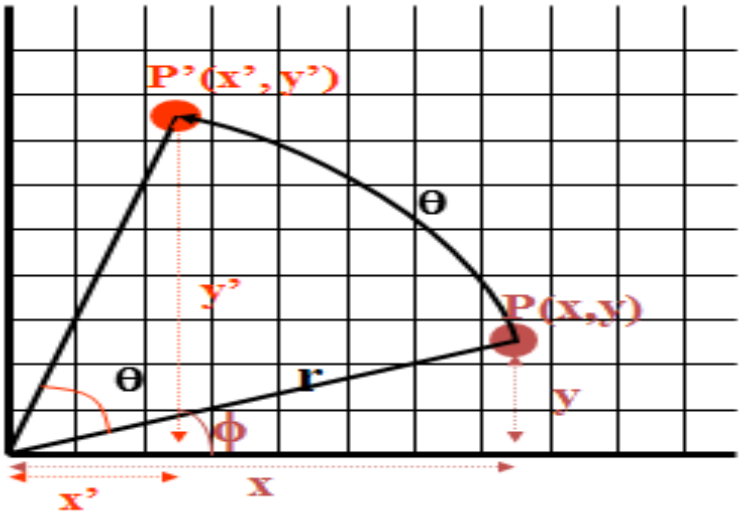
$x' = r(\cos\theta.\cos\alpha - \sin\theta.\sin\alpha)$

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

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

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

In matrix form:



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

P' = R • P, where R is a rotation matrix.

θ can be clockwise (-ve) or counterclockwise (+ve in this example).

4.3.3 Scaling

Scaling transformation alters the size of an object. Scaling changes the size of an object and involves two scale factors, s_x and s_y for the x- and y- coordinates respectively.

Scaling about the origin:

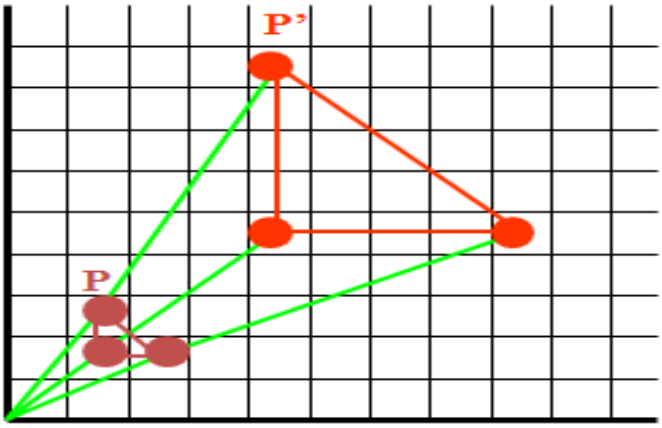
If P(x, y) be scaled to a point P'(x', y') by s_x times in x-coordinate and s_y times in y-coordinate then,

$x' = s_x \bullet x$

$y' = s_y \bullet y$

In matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



P' = S • P, where S is a scaling matrix**.**

- **If the scale factors are the same,**
  - That is, $s_x = s_y$ ➔ uniform scaling.
  - Only change in size.
- **If the scale factors are different**
  - That is, $S_x \neq S_y$ ➔ differential scaling.
  - Change in size and shape.

4.4 Matrix Representations and Homogeneous co-ordinates

Many graphics applications involve sequences of geometric transformations. An animation, for example, might require an object to be translated and rotated at each increment of the motion. In design and picture construction applications, we perform translations, rotations, and scaling's to fit the picture components into their proper positions. Here we discuss how such transformation sequences can be efficiently processed using matrix notation.

Homogeneous coordinate system:

- It is the way representing n-dimensional Cartesian coordinates into n+1 dimensional projective coordinates or projective plane.
- In homogeneous coordinate representation each 2D point (x, y) is represented as homogeneous coordinate triple (x_h, y_h, h), where $x = x_h/h$, $y = y_h/h$
- h is 1 for 2D, and 3D case.
- Example:
  - 2D point (2, -4) can be represented as (2, -4, 1).

Why Homogeneous Co-ordinates?

- The homogeneous co-ordinate system provides a uniform framework for handling different geometric transformations, simply as multiplication of matrices.
  - To express any two-dimensional transformation as a matrix multiplication.
  - To represent all the transformation equations as matrix multiplications.
  - To perform more than one transformation at a time.
  - To reduce unwanted calculations of intermediate steps, to save time and memory and produce a sequence of transformations.

Homogeneous Coordinate Representation For translation:

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

Therefore, P' = $T(t_x, t_y).P$

For inverse translation:

$T^{-1}(t_x, t_y) = T(-t_x, -t_y)$

Homogeneous Coordinate Representation for Rotation (about origin):

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

Therefore, P' = $R(\theta).P$

For inverse rotation:

$R^{-1}(\theta) = R(-\theta)$

Homogeneous Coordinate Representation For Scaling (about origin):

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

Therefore, P' = $S(s_x, s_y).P$

For inverse scaling:

$S^{-1}(s_x, s_y) = S(1/s_x, 1/s_y)$

4.5 Composite Transformation

In many cases we need to apply several transformations (scaling, rotation, translation) to one object. It is possible to set up a matrix for any sequence of transformations as a composite transformation matrix by calculating the matrix product of the individual transformations.

The basic purpose of composite transformations is gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after another.

Exercises:

I. Suppose we have a quadrilateral ABCD with vertices A(3,1), B(-7,-4), C(5,-8) and D(3,0). What is the coordinates of quadrilateral if it is to be doubled in size about origin. Then translate obtained coordinates with (-4, 3).

II. Rotate the triangle having co-ordinates (1, 2), (2, 3) (4, 5) by 60° about origin. Then translate with T(9, 0), and then scale about origin with scale factor S(2, 4).

Translations

What happens when a point goes through $T(t_{x1}, t_{y1})$ and then $T(t_{x2}, t_{y2})$? (***Two successive Translation***)

❖ **Prove that two successive translations are additive. i.e. $T(t_{x1}, t_{y1}).T(t_{x2},t_{y2}) = T(t_{x1} + t_{x2}, t_{y1}+ t_{y2})$.**

If two successive translation vectors (t_{x1}, t_{y1}) and (t_{x2}, t_{y2}) are applied to a coordinate position P, the final transformed location P' is calculated as: -

$$P' = T(t_{x2},t_{y2}) . \{T(t_{x1},t_{y1}) .P\}$$
$$= \{T(t_{x2},t_{y2}) . T(t_{x1},t_{y1})\} .P$$

Net transformation = $T(t_{x2}, t_{y2}).T(t_{x1}, t_{y1})$

$$= \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & t_{x1} \\ 0 & 1 & t_{y1} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= T(tx_1 + tx_2, ty_1 + ty_2)$$

$T(t_{x2},t_{y2}).T(t_{x1},t_{y1}) = T(t_{x1}+t_{x2}, t_{y1}+t_{y2})$, which demonstrates that two successive translations are additive.

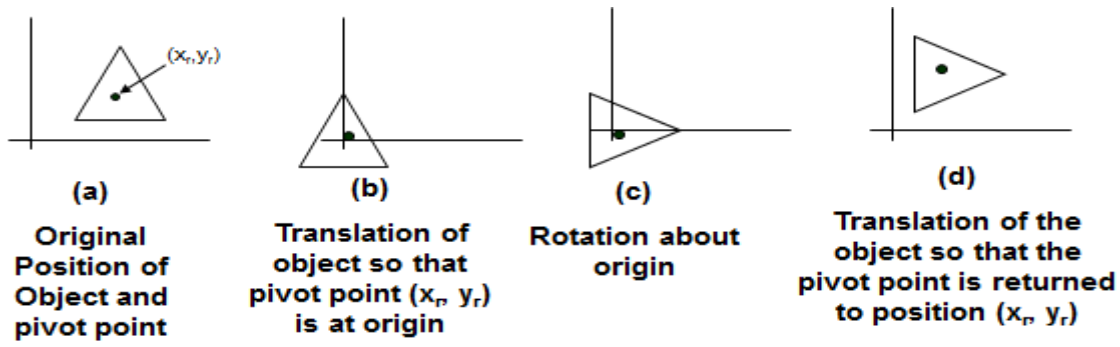
❖ **Similarly prove that:**

- Two successive rotations are additive i.e. $R(\theta_2) \cdot R(\theta_1) = R(\theta_1 + \theta_2)$.
- Two successive scaling are multiplicative i.e. $S(s_{x2}, s_{y2}).\ S(s_{x1},s_{y1}) = S(s_{x1}.s_{x2}, s_{y1}.s_{y2})$.

4.6 General pivot point rotation

To generate a rotation about any selected pivot point (x_r, y_r), we need to perform following sequence of translate-rotate-translate operations:

1. *Translate the object so that pivot-position is moved to the coordinate origin.*
2. *Rotate the object about the coordinate origin.*
3. *Translate the object so that the pivot point is returned to its original position.*

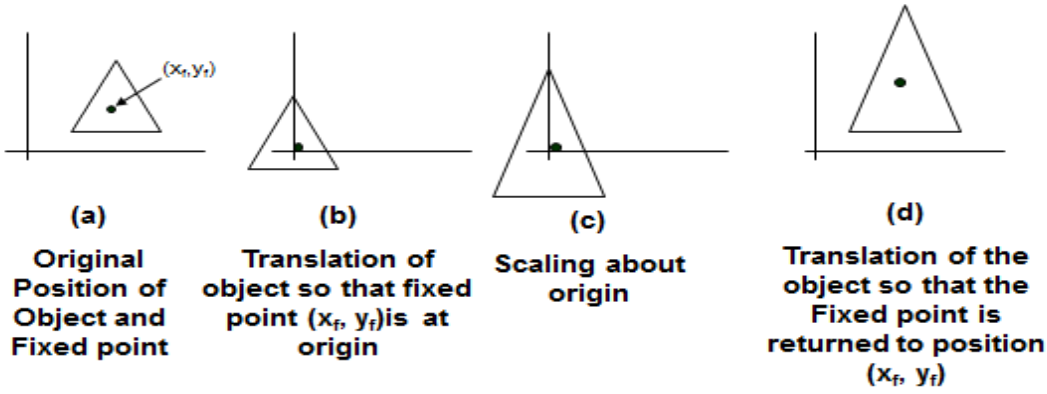


Composite Transformation: $T(x_r, y_r).R(\theta).(-x_r, -y_r)$

4.7 General fixed point scaling

A transformation sequence to produce scaling with respect to a selected fixed position (x_f, y_f) is illustrated below:

1. *Translate object so that the fixed point coincides with the coordinate origin.*
2. *Scale the object with respect to the coordinate origin.*
3. *Use the inverse translation to return the object to its original position.*



Composite Transformation: $T(x_f, y_f).S(s_x, s_y).T(-x_f, -y_f)$

Exercise: *Rotate a triangle A(4, 7), B(5, 5), and (3, 6) by 45 degree in clockwise direction about an arbitrary pivot point(3, 3).*

4.8 Other transformations

- ***Reflection***
- ***Shearing***

4.8.1 Reflection

Reflection is a transformation that produces a mirror image of an object. In 2D-transformation, reflection is generated relative to an axis of reflection. The reflection of an object to a relative axis of reflection is same as 180^o rotation about the reflection axis.

Following are examples of some common reflections:

1. ***Reflection about the line y=0, the X- axis.***
2. ***Reflection about the line x=0, the Y- axis.***
3. ***Reflection about the line passing through the coordinate origin.***
4. ***Reflection of an object w.r.t the straight line y=x.***
5. ***Reflection of an object w.r.t the straight line y= -x.***
6. ***Reflection of an object w.r.t. arbitrary axis y=mx+b.***

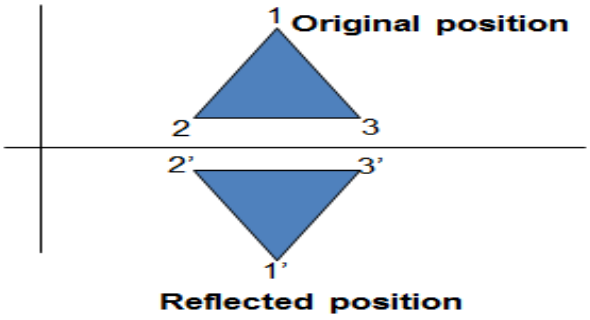
1. Reflection about the line y=0, the X- axis.

$x' = x$

$y' = -y$

In matrix from,

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



Therefore, P'= R_x.P, where R_x is a reflection matrix.

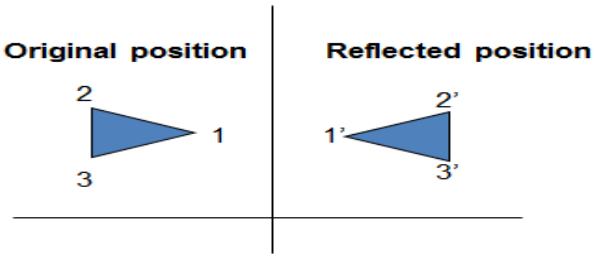
2. Reflection about the line x=0, the Y- axis.

$x' = -x$

$y' = y$

In matrix from,

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



Therefore, P' = R_y.P.

3. Reflection about the line passing through the coordinate origin.

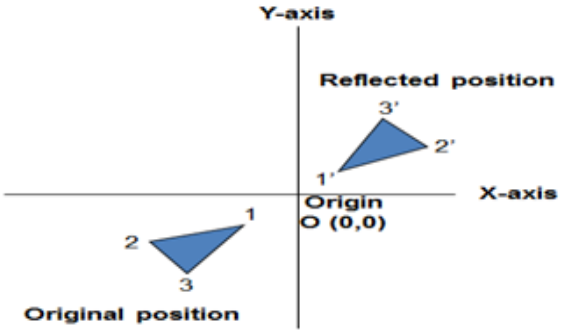
$x' = -x$

$y' = -y$

In matrix form,

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

Therefore, P' = R_0.P.



4. Reflection of an object w.r.t the straight line y=x.

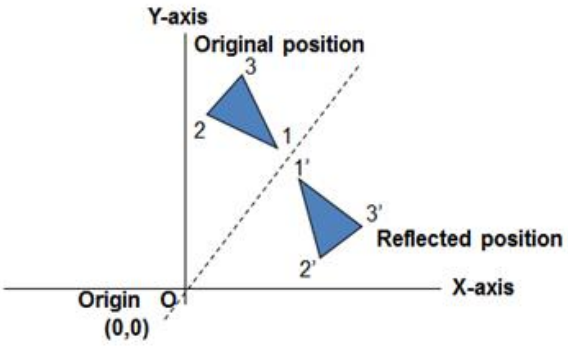
$x' = y$

$y' = x$

In matrix form,

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

Therefore, P' = R.P



5. Reflection of an object w.r.t the straight line y= -x.

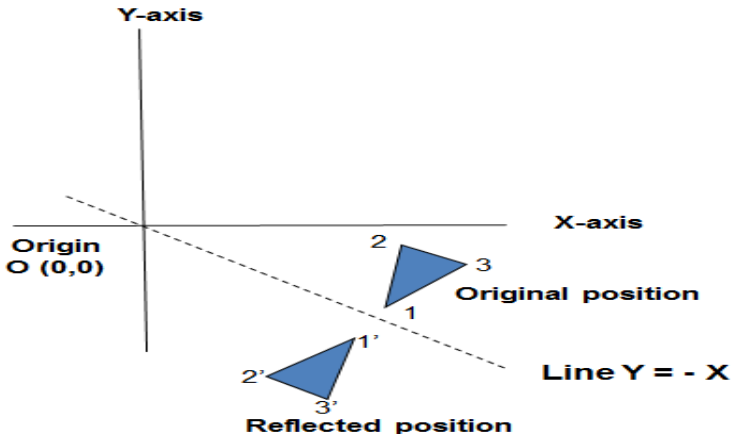
x = -y

y = -x

In matrix form,

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

Therefore, P' = R.P.



6. Reflection of an object w.r.t. arbitrary axis y=mx+b.

Basic Steps:

1. Translate the object so that reflection axis passes through origin i.e., apply T(0, -b).

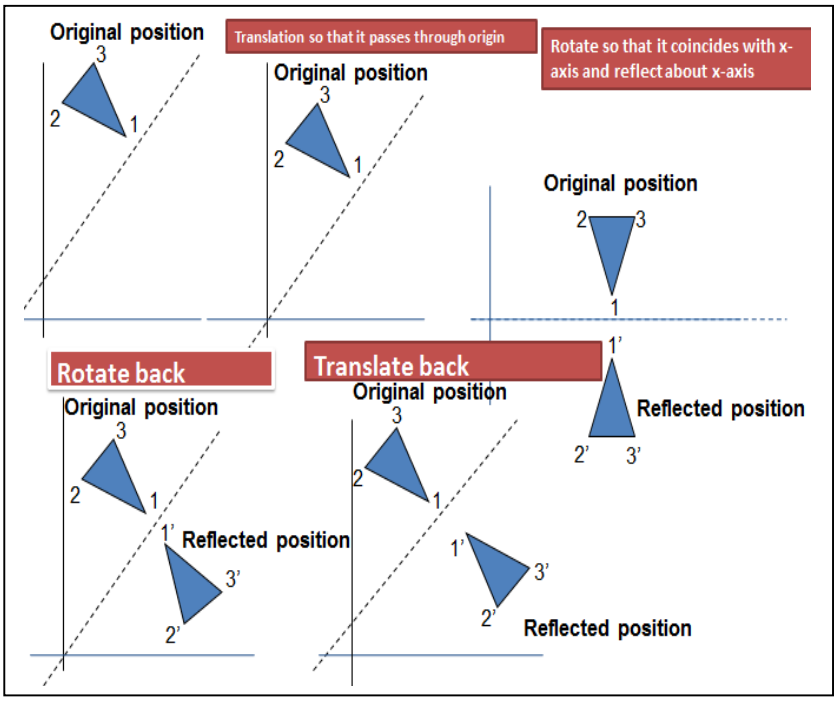
2. Rotate object such that reflection axis coincides with one of the coordinate axis, in this case x-axis i.e., apply R(-θ).

Here, $\theta = \tan^{-1}(m)$.

3. Reflect the object w.r.to that axis. In this case x-axis i.e., apply R_X.

4. Rotate back so that reflection axis regains its initial slope i.e., apply R(θ).

5. Translate back to return the object to its original position i.e., apply T(0, b).



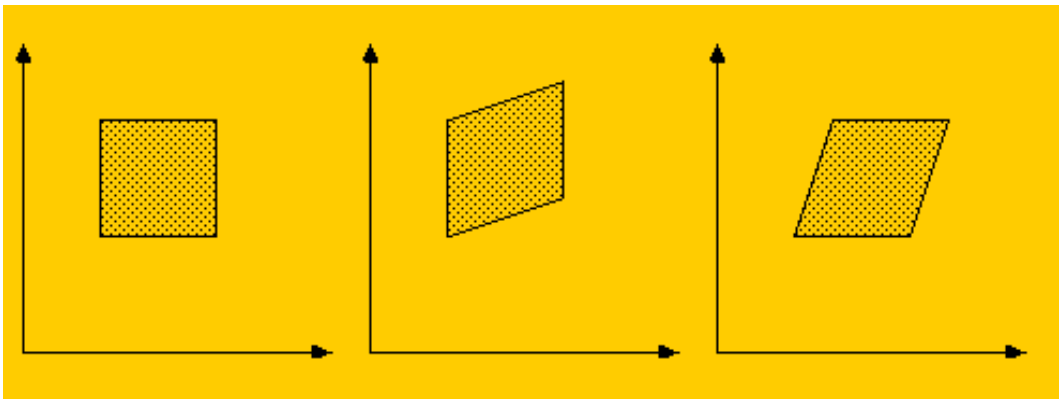
Net transformation

T(0, b).R(θ).Rx.R(-θ).T(0, -b)

4.8.2 Shearing

A shear is a transformation that distorts the shape of an object along coordinate axes. It distorts the shape of an object such that the transformed shape appears as if the object were composed of internal layers that had been caused to slide over each other. Two common shearing transformations are those that shift coordinate x-values and those that shift y-values. However, in both the cases, only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as ***Skewing***.

Example



Original Y-Shear X-Shear

For example, $Sh_x=2$

X-direction shear

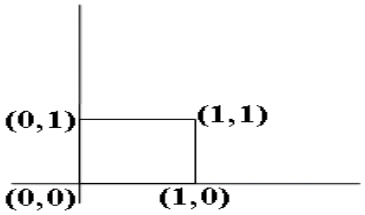
- The X-Shear preserves the y-coordinate and changes are made to x-coordinates.
- Shifts coordinate x-values horizontally by an amount proportionally to its y-coordinate.
- A horizontal shear (or shear parallel to the x axis) is a function that takes a generic point with coordinates (x, y) to the point ($x+sh_x.y$, y); where sh_x is a fixed parameter, called the shear factor.

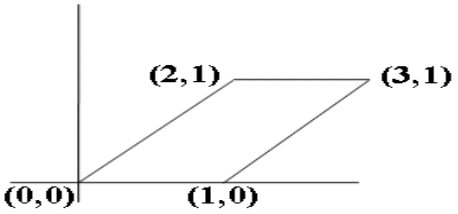
$y' = y$

$x' = x+sh_x.y$

In matrix form,

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





Where, Sh_x is shearing constant, can be any real number.

Y-direction shear

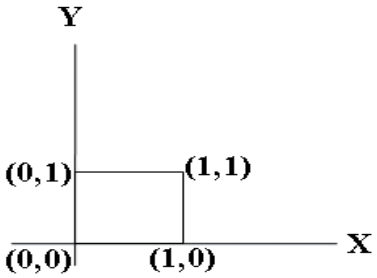
- The Y-Shear preserves the x-coordinates and changes the y-coordinates.
- It shifts coordinate y-values vertically by an amount proportionally to its x-coordinate.
- A vertical shear (or shear parallel to the y-axis) of lines is similar, except that the roles of x and y are swapped.

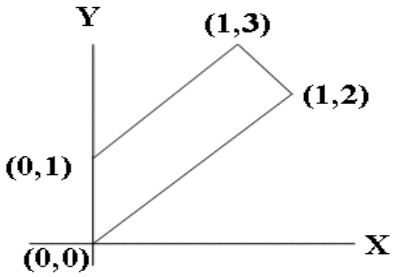
$x' = x$

$y' = y + sh_y.x$

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

For example, $Sh_y=2$





Where, Sh_y is shearing constant, can be any real number.

Similarly, we can determine XY-shear.

XY- shear

- Both x-coordinates and y-coordinates change.
- That is, it shifts both x and y coordinate values.
- Let P(x, y) be the point that is to be sheared along xy-direction with shear parameter Sh_x, Sh_y and P'(x', y') be the resultant point.

  Then,

  $x' = x + Sh_x.y$

  $y' = y + Sh_y.x$

Exercise:

1. A triangle A (4, 5), B (2, 1), and C (6, 1) is required to enlarge twice its initial size about the fixed point (4, 2). Find the final coordinates.
2. Suppose we have a quadrilateral ABCD with vertices A (3, 1), B (-7,-4), C (5,-8), and D(3, 0). What is the coordinates of quadrilateral if it is to be doubled in size about origin. Then translate obtained coordinates with (4, 3).
3. Transform a square with corners A (2, 2), B (4, 2), C (2, 4), D (4, 4) into a rectangle whose breadth is half of the breadth of square and length is double of it.
4. A triangle A (4, 5), B (2, 1), and C (6, 1) is required to enlarge with scaling factors S(2, 2), and S(2, 0.5). Find the new vertices of the triangle.
5. Rotate the △ABC by 45° clock wise about origin and scale it by (2, 3) about origin, where A(7,15), B(5,8), and C (10,10).
6. Represent all the transformations with homogeneous matrix representation. Consider a triangle ABC with vertices A(3,4), B(7, 4), and C(5, 8). What is the transformation if the triangle is to be doubled in size keeping A(3, 4) fixed? Obtain new vertices of the triangle.
7. Reflect a triangle ABC with vertices A(1, 9), B(3, 3), C(1, 6) about the line y = x+2.
8. Reflect a triangle ABC with vertices A(1, 4), B(3, -9), C(2, 2) about the line y = 2.

9. Determine transformation responsible for reflection of object about the line x = -3.
10. Translate the given triangle A(1, 3), B(-2, -1), C(6, -2) with translation vector T(-2, -6) and then reflect with y= x axis.
11. Determine transformation responsible for reflection of object about the line y + x = 2.
12. Prove that two successive rotations are additive.
13. Prove that two successive scaling are multiplicative.
14. Why homogeneous coordinate representation is necessary in computer graphics?

Chapter 5

Two dimensional Viewing (2D Viewing), and Clipping

5.1 Two dimensional Viewing (2D Viewing)

- It is a formal mechanism for displaying views of a picture on an output device.
- A graphics package allows the user to specify:
  - Which part of the defined picture is to be displayed(***Window***).
  - Where the part is to be placed on the display device(***Viewport***).
  - A **view** is selected by specifying a sub area of the total picture area.
- Much like what we see in real life through a small window or the view finder of a camera.
- In computer graphics several coordinate systems (reference frame) are used to construct, select views, and display a 2D scene or image.
  - **Modeling coordinate system**
    - It is used to define coordinates that are used to construct the shape of individual parts (*objects*) of a 2D scene.
  - **World Coordinate system**
    - Used to organize the individual objects (*points, lines, circles etc*) into a scene.
    - A scene is made up of a collection of objects.
    - 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.
  - **Viewing coordinate system**
    - Used to define particular view of a 2D scene. Translation, scaling, and rotation of the window will generate a different view of the scene.
    - For a 2-D picture, a view is selected by specifying a subarea of the total picture area.
  - Normalized viewing coordinates
    - These are viewing coordinates between 0 and 1. The lower left corner corresponds to (0, 0), and the upper right corner corresponds to (1, 1).
    - NVC's are used to make the viewing process independent of the output device (*monitor, mobile, hard copy devices*).
    - Clipping is usually and more efficiently done in these coordinates.

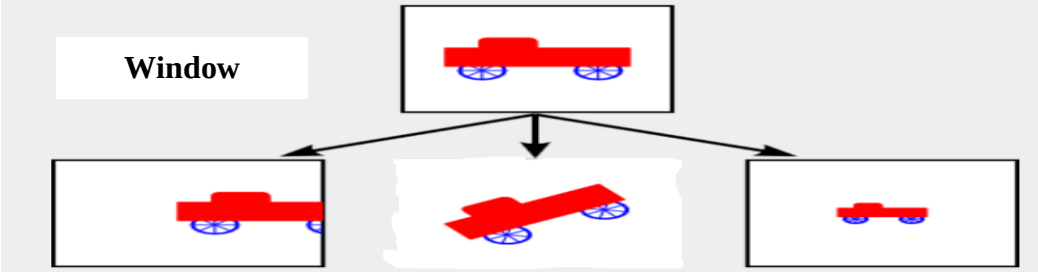


Fig: several view of the same car

❖ **Device Coordinate or Screen Coordinate system**

- Used to define coordinates in an output device.
- Device coordinates are integers within the range (0, 0) to (x_{max}, y_{max}) for a particular output device.
- Depends on screen resolution.
- **Normalized Device Coordinates:**
  - These are viewing coordinates between 0 and 1.
  - Device coordinates are specific to output device: printer page, screen display, etc. but Normalized Device co-ordinates are independent.

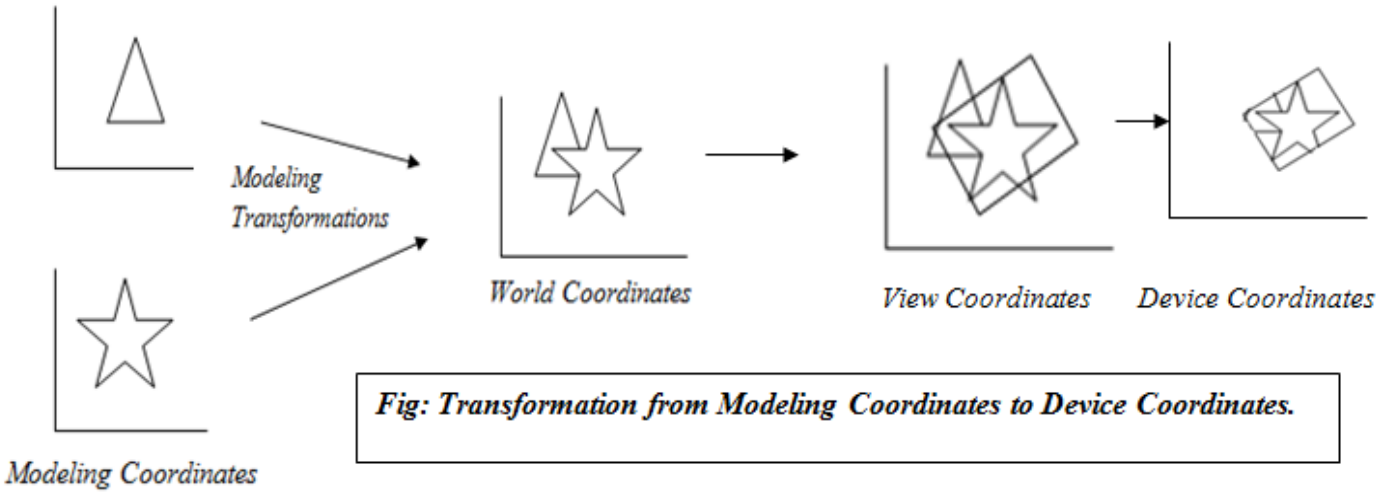


Fig: Transformation from Modeling Coordinates to Device Coordinates.

❖ Any convenient Cartesian coordinate system; referred to as the world coordinate reference frame can be used to define the picture. The picture parts within the selected areas are then mapped onto specified areas of the device co-ordinates.

5.2 Viewing Pipeline/2D-viewing transformation

❖ The process of mapping of a part of world coordinate scene to device coordinate is called the 2D viewing transformation.

❖ That is, it is a process of mapping the coordinates of the points and lines that form the picture into the appropriate coordinates on the device where the image is to be displayed. Sometimes also referred to as the *window-to-viewport transformation or the windowing transformation*.

❖ Transformations from world to device coordinates involve translation, rotation and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area(*clipping*).

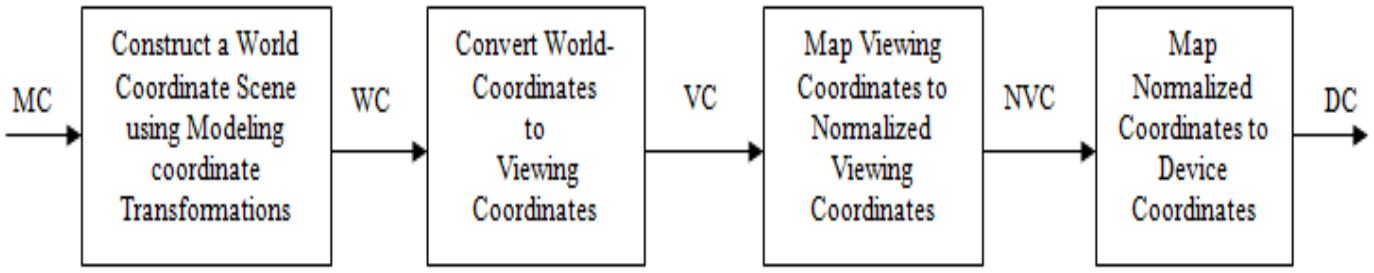


Fig: Two-dimensional viewing pipeline

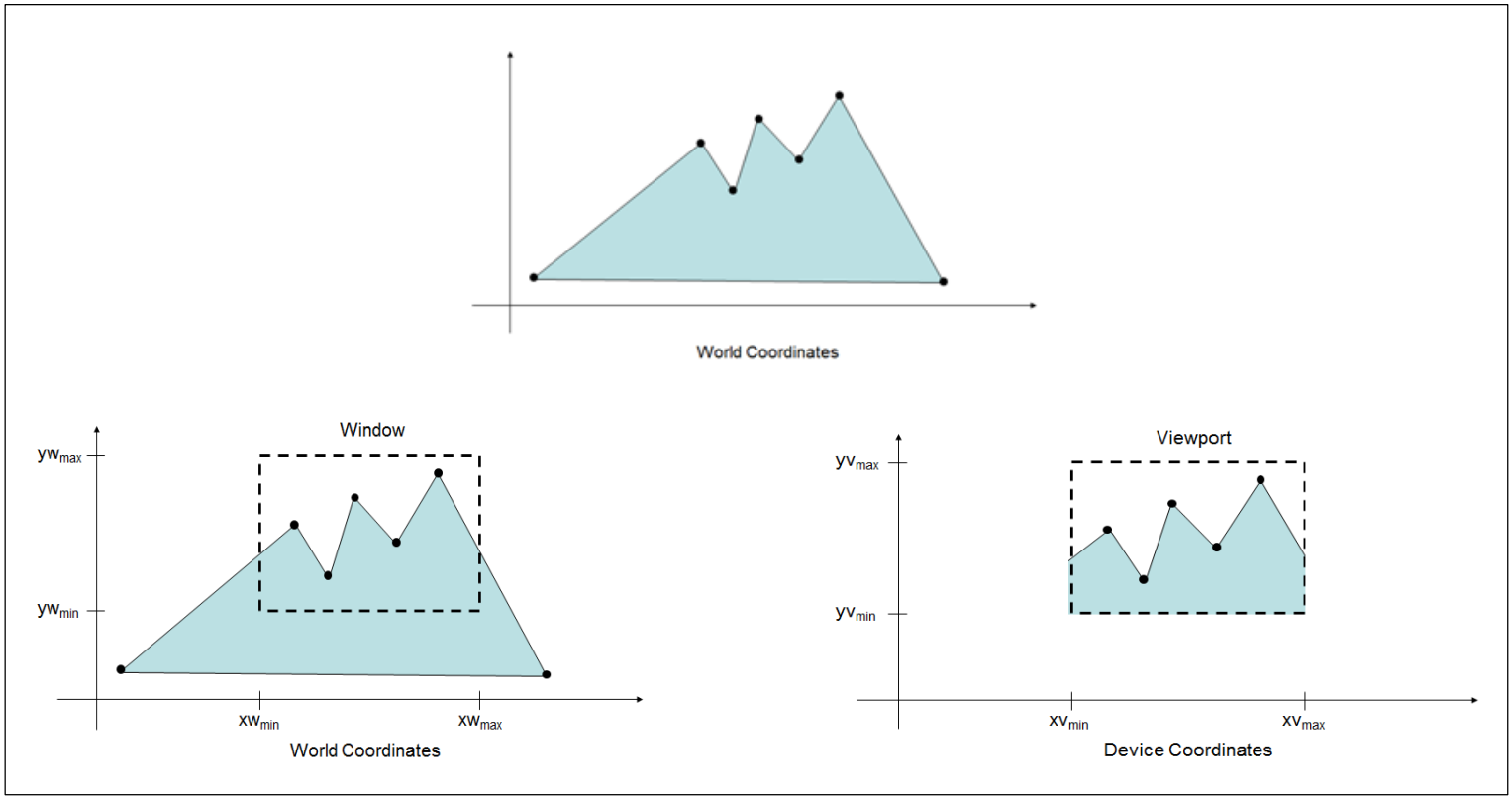
Window

- A world-coordinate area selected for display is called a window. That is, window is the section of the 2D scene that is selected for viewing.
- The window defines what is to be viewed.
- A window can be specified by four world coordinates: x_{wmin}, x_{wmax}, y_{wmin} and y_{wmax}.

Viewport

- An area on a display device to which a window is mapped is called a viewport.
- The viewport indicates where on an output device selected part will be displayed.
- A viewport can be described by four device coordinates: x_{vmin}, x_{vmax}, y_{vmin} and y_{vmax}.

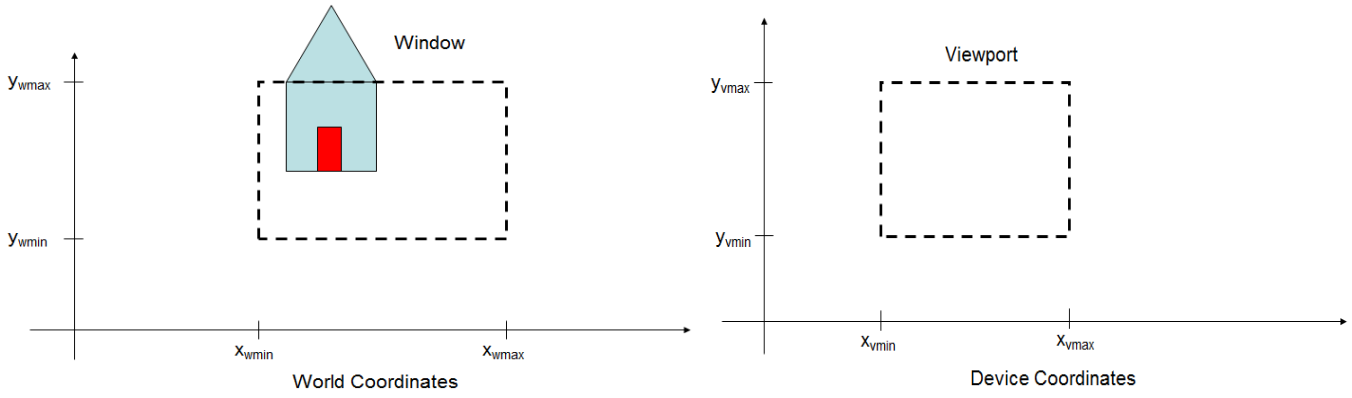
Figure given below illustrates the mapping of a 2D picture that falls within a rectangular window onto a designated rectangular viewpoint.



- *By changing the position of the viewport, we can view objects at different positions on the display area of an output device.*
- *Also by varying the size of viewports, we can change size of displayed objects.*
- *Zooming effects can be obtained by successively mapping different-sized windows on a fixed-sized viewport.*
- *Panning effects are produced by moving a fixed-size window across the various objects in a scene.*

5.3 Window-to-viewport coordinate transformation

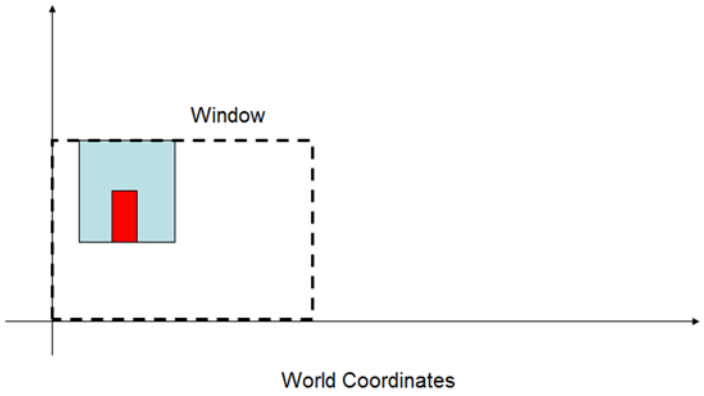
A window can be specified by four world coordinates: x_{wmin}, x_{wmax}, y_{wmin} and y_{wmax}. Similarly, a viewport can be described by four device coordinates: x_{vmin}, x_{vmax}, y_{vmin} and y_{vmax}.



Then, the window-to-viewport transformation can be explained in three steps:

1. Translate object along with window such that the lower left corner of the window is at the origin.

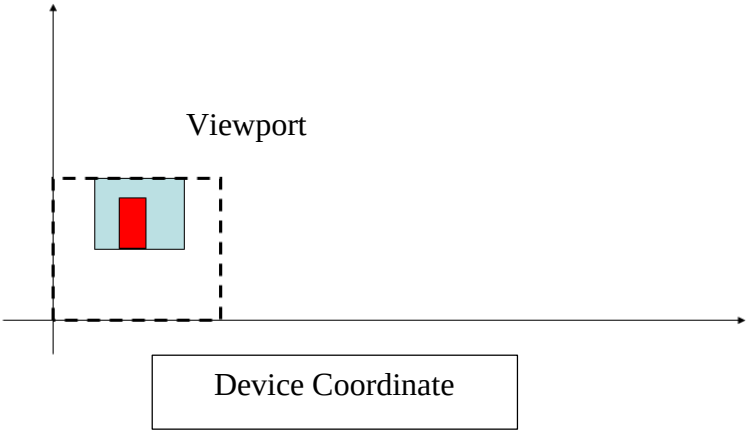
That is, apply T ($-x_{wmin}$, $-y_{wmin}$).



2. Scale the object and the window such that window has the same dimension as that of a viewport.

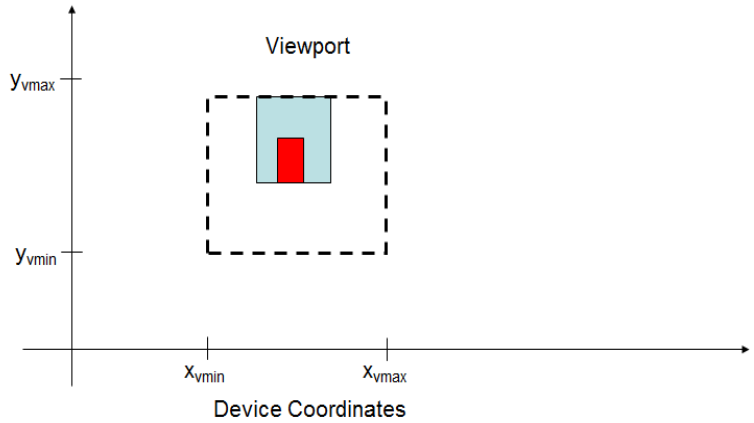
That is, apply S (s_x, s_y).

Here, sx, and sy can be obtained by using relative displacement along x-coordinate, and y-coordinate.



3. Finally, apply another translation to move the viewport to its original position on the screen.

That is, apply T(x_{vmin}, y_{vmin}).



Therefore, net transformation,

$$T_{WV} = T(x_{vmin}, y_{vmin}).\ S(s_x, s_y).T(-x_{wmin}, -y_{wmin}) \quad \ldots\ldots(1)$$

Here, s_x and s_y are scaling factors, where;

$$s_x = \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$$

$$s_y = \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$$

Now, substituting these values;

$$T_{wv} = \begin{bmatrix} 1 & 0 & x_{vmin} \\ 0 & 1 & y_{vmin} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} & 0 & 0 \\ 0 & \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{wmin} \\ 0 & 1 & -y_{wmin} \\ 0 & 0 & 1 \end{bmatrix}$$

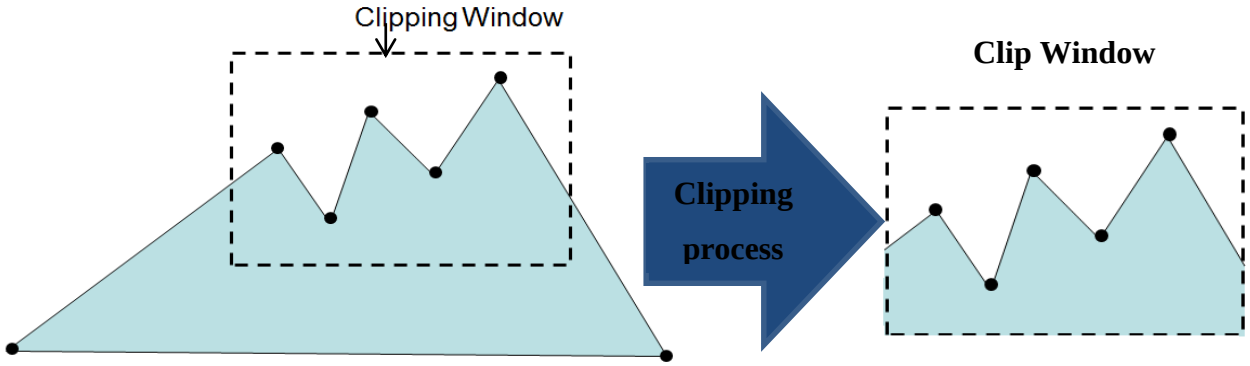
Is the required window-to-viewpoint transformation.

Let P(x, y) be the world coordinate point that is mapped onto the viewport point P'(u, v), then we must have;

$$P' = T_{wv}.P$$

5.4 Clipping

- The process of discarding (cutting off) those parts of a picture which are outside of a specified region is called *clipping.*
- Any procedure that identifies those parts of a picture that are either inside or outside of the specified region is called a *clipping algorithm*.
- The region against which the clipping operation is performed is called a clip window.



(**Before Clipping**) (**After Clipping**)

5.4.1 Why Clipping?

- Excludes unwanted graphics from the screen.
- Improves efficiency, as the computation dedicated to objects that appear off screen can be significantly reduced. Extracting part of a defined scene for viewing.
- Identifying visible surfaces in three-dimensional views.
- Creating objects using solid-modeling procedures.
- Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.

5.4.2 Applications

- Video editing
- Image Editing

5.5 Clipping Algorithms

5.5.1 Point Clipping

- Let W(x_{min}, x_{max}, y_{min}, y_{max}) denote a clip window.
- A point (x, y) is considered for drawing only if x lies between x_{min} and x_{max} and y lies between y_{min} and y_{max}. That is if following inequalities are satisfied:

$$x_{min} \leq x \leq x_{max},$$

$$y_{min} \leq y \leq y_{max}.$$

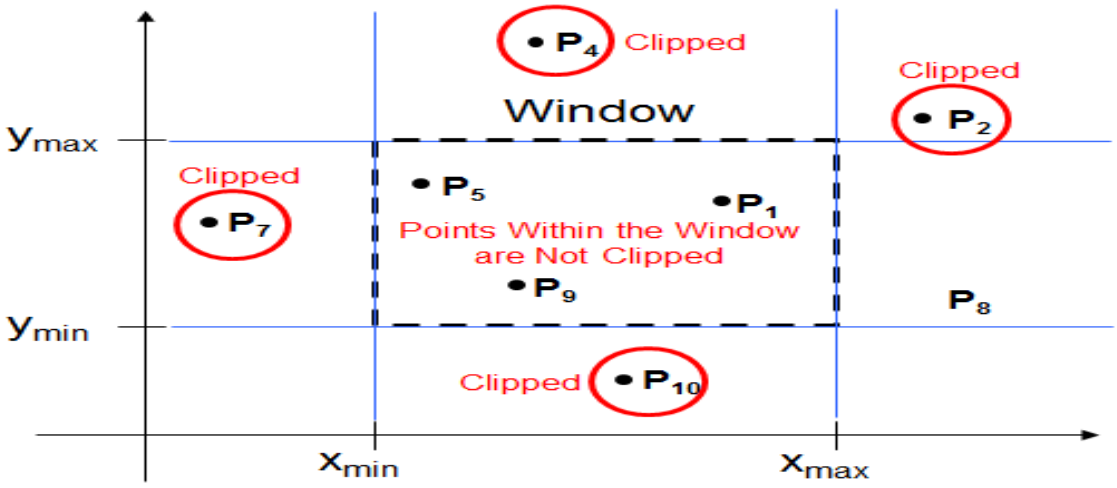
otherwise it is clipped.

- **The following algorithm reflects this fact:**

```
Algorithm PointClipping(x,y,xleft, ybottom ,xright, ytop)
    If (x>= xmin) and (x<= xmax) and (y>= ymin) and (y<= ymax) then
        - DrawPoint(x, y).

    End if
End Algorithm.
```

Example:

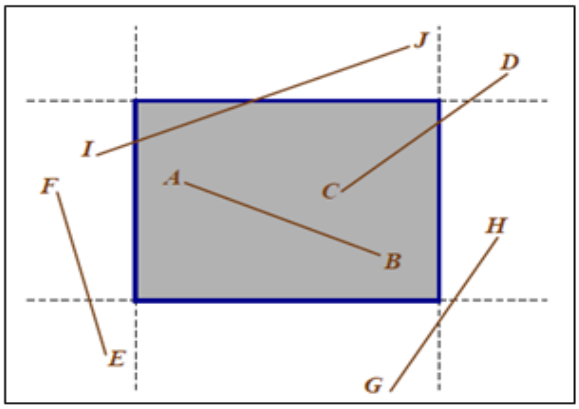


5.5.2 Line Clipping

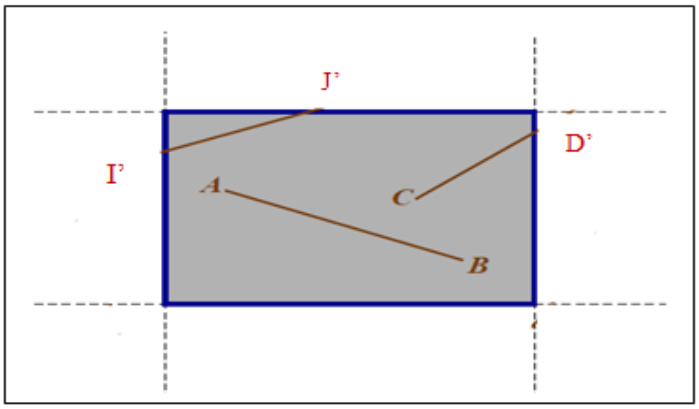
In computer graphics, line clipping is the process of removing lines or portions of lines outside an area of interest. Typically, any line or part thereof which is outside of the viewing area is removed. Lines that do not intersect the clipping window are either completely inside the window or completely outside the window. On the other hand, a line that intersects the clipping window is divided by the intersection point(s) into segments that are either inside or outside the window.

Any particular line segment falls into one of the following clipping categories:

1. **Completely Visible:**
    - **a.** Both endpoints are inside the region (line ***AB***).
    - **b.** No clipping necessary.
2. **Non-Visible:**
    - **a.** Line lies completely outside the clip window (lines ***EF***, ***GH***).
    - **b.** Discard the line segment.
3. **Partially visible(clipping candidate)**
    - **a.** A part of line segment lies within a window (lines ***CD, IJ***).
    - b. Compute intersection points and clip at the intersection points.



Before clipping

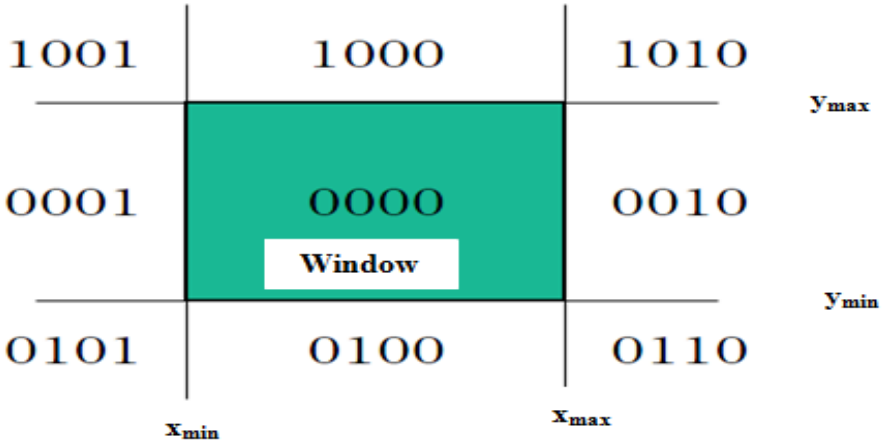


After Clipping

1. Cohen-Sutherland Line Clipping Algorithm

One of the oldest and most popular line-clipping algorithms. In this method, area containing the window is divided into nine regions. All regions have their associated codes. Every line endpoint is assigned a four digit binary code (*region code or out code*). Each bit in the code is set to either a 1(true) or a 0(false).

Each region is assigned a four bit pattern as shown in figure.

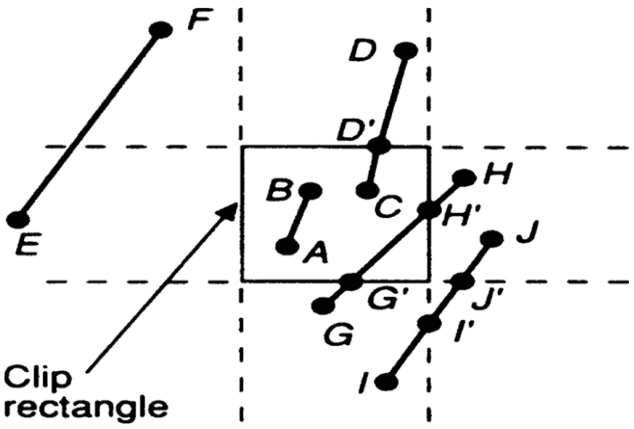


Any point inside the clipping window has a region code 0000.

Any endpoint (x, y) of a line segment, the code can be determined as follows:

- If $x < x_{min}$, first bit is 1, (Point lies to left of window) (0^{th} bit)
  - Otherwise 0.
- If $x > x_{max}$, second bit is 1, (Point lies to right of window) (1^{st} bit)
  - Otherwise 0.
- If $y < y_{min}$, third bit is 1, (Point lies to below window) (2^{nd} bit)
  - Otherwise 0.
- If $y > y_{max}$, fourth bit is 1, (Point lies to above window) (3^{rd} bit)
  - Otherwise 0.

Outcodes – Example:



Line	Code 1	Code 2
A⟹B	0000	0000
C⟹D	0000	1000
E⟹F	0001	1001
G⟹H	0100	0010
I⟹J	0100	0010

Algorithm:

1. *Given a line segment with endpoint* $P_1 = (x_1, y_1)$ *and* $P_2 = (x_2, y_2)$.
2. *Compute the 4-bit codes for each endpoint,* OP_1 *and* OP_2.
3. *If* $OP_1 \vee Op_2 = 0000$
   - *Line lies completely inside the window, so draw the line segment.*
4. *If* $OP_1 \wedge Op_2 \neq 0000$
   - *Line lies completely outside the window, so discard the line segment.*
5. *Otherwise, compute the intersection of the line segment with window boundary, and discard the portion of the segment that falls completely outside the window. Assign a new four-bit code to the intersection and repeat until either 3 or 4 above are satisfied.*

Finding intersection points

- For a line with endpoints (x_1, y_1), (x_2, y_2), intersection points with the clipping boundary can be calculated as follows.
- If intersection is with the vertical boundary,

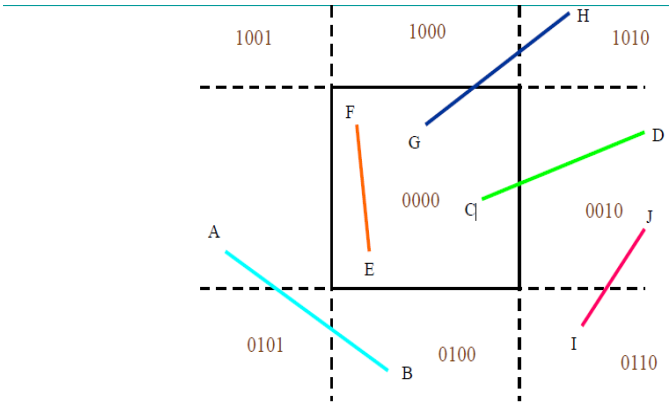
  $x = x_{min}$ or x_{max}.

  and $y = y_1 + m(x - x_1)$,
- If intersection is with the horizontal boundary,

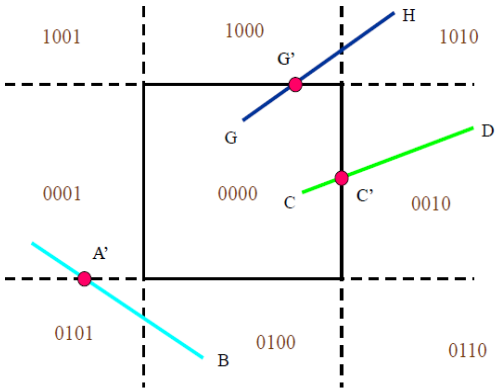
  $y = y_{min}$ or y_{max}.

  $x = x_1 + 1/m * (y - y_1)$,
- And $m = (y_2 - y_1) / (x_2 - x_1)$.

Cohen-Sutherland Line Clip Examples



A 0001	C 0000	E 0000	G 0000	I 0110
B 0100	D 0010	F 0000	H 1010	J 0010
OR 0101	OR 0010	OR 0000	OR 1010	OR 0110
AND 0000	AND 0000	AND 0000	AND 0000	AND 0010
clip	**clip**	**accept**	**clip**	**reject**



remove A'B	**remove C'D**	**remove G'H**
A 0001	C 0000	G 0000
A' 0001	C' 0000	G' 0000
OR 0001	OR 0000	OR 0000
AND 0001	AND 0000	AND 0000
reject	**accept**	**accept**

Solved Example:

- Use Cohen-Sutherland Algorithm to clip the line $P_1(70, 20)$, and $P_2(100, 10)$ against a window with a lower left corner at (50, 10) and upper right corner at(80, 40).

- **Solution:**

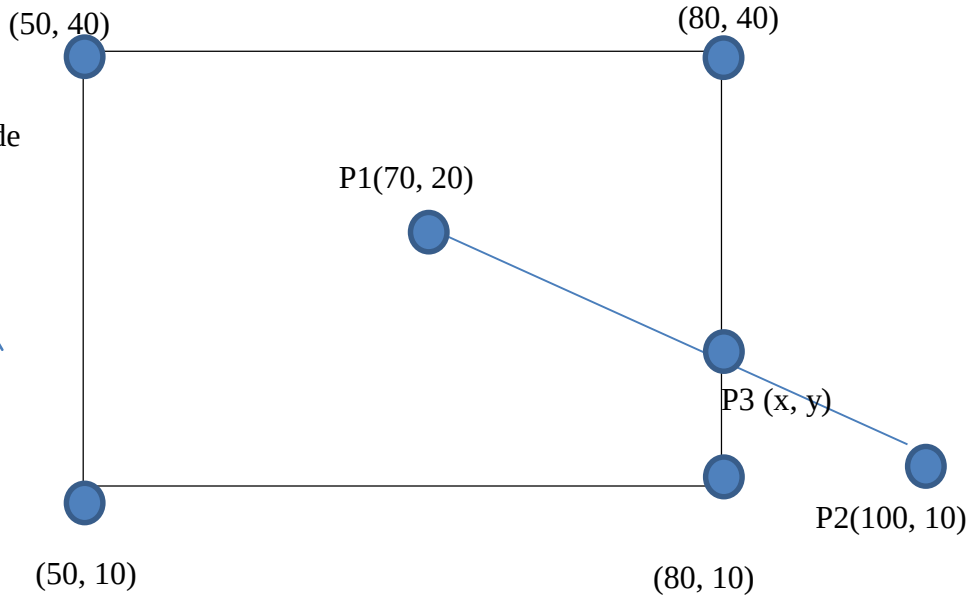
Given,

$P_1(70, 20)$, and $P_2(100, 10)$.

Window's lower left corner is (50, 10).

Window's upper left corner is (80, 40).

Therefore the window must be-



Now, we need to calculate 4 bit outcode for each endpoint P1 and P2.

For P1;

OP1=	70< 50	0 (L)
	70>80	0 (R)
	20<10	0 (B)
	20>40	0 (T)
		0000

For P2;

OP2 =	100<50	0 (L)
	100>80	1 (R)
	10<10	0 (B)
	10>40	0 (T)
		0010

Therefore,

$OP1 \vee OP2 = 0000 \vee 0010 = 0010$

Hence, line does not lie completely inside the window.

Again,

$OP1 \wedge OP2 = 0000 \wedge 0010 = 0000$

Hence, line does not lie completely outside of the window. So, we need to calculate intersection point.

Let P3(x, y) bet the intersection point.

But, from figure $x = 80$ (since vertical intersection)

And $y = y_1 + m(x - x_1)$

= 20 + (-1/3)(80-70)

y = 16.667

S, the part P2P3 is clipped as it is outside of the window.

Therefore,P3(x, y) = P3(80, 16.667).

Now, region code for P3;

OP3 = 0000.

So, OP1∨ OP3= 0000

Hence line segment P1P3 is accepted.

Exercise:

- Clip the line XY with endpoints X(3, 5), Y(9, 3) against clipping boundary MNOP; M(3, 3), N(3, 8), O(8, 8), P(8, 3) using Cohen-Sutherland line clipping algorithm.

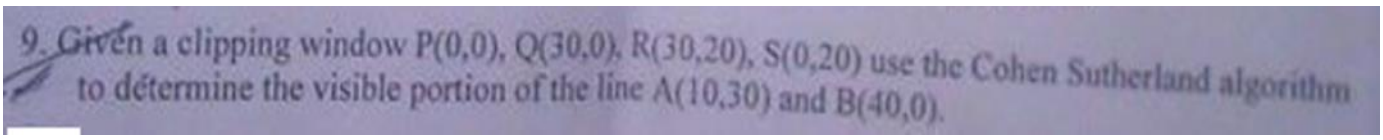
- Use Cohen Sutherland algorithm to clip the line for the following dimensions. Line endpoints (15, 45) and (25,15)

  Window:

  Lower left: (10, 20)

  Upper right: (30, 40)

- Use Cohen-Sutherland Algorithm to clip the line P_1(70, 20), and P_2(100, 10) against a window with a lower left corner at (50, 10) and upper right corner at(80, 40).



- You are provided with the clipping rectangle with co-ordinates A (4,4), B(4,10), C(10,10) and D (10,4). Clip the given lines XY, PQ and MN with coordinates X(-2,5), Y(8,11), P(1,1), Q(11,2), M(4,5) and N(9,10) using Cohen-Sutherland line clipping algorithm.

2. Liang-Barsky Algorithm

- It is an efficient algorithm, which uses parametric equations to clip the line segment.
- The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clip window.
- With these intersections it knows which portion of the line should be drawn. This algorithm is significantly more efficient than Cohen–Sutherland.
- The idea of the Liang–Barsky clipping algorithm is to do as much testing as possible before computing line intersections.
- It performs better than Cohen-Sutherland algorithm as it requires less number of intersection point calculations.

Let P1(x1, y1), and P2(x2, y2) are line endpoints. The parametric equation of a line can be given by,

$$x = x_1 + t\Delta x$$
$$y = y_1 + t\Delta y, \qquad 0 \le t \le 1$$
$$\text{where} \quad \Delta x = x_2 - x_1 \text{ and } \Delta y = y_2 - y_1$$

Then any point P(x, y) is inside of the clipping window iff;

$$x_{wmin} \le x_1 + t\Delta x \le x_{wmax} \text{ and}$$
$$y_{wmin} \le y_1 + t\Delta y \le y_{wmax}$$

Re-writing these four inequalities with two parameters as follows;

$$tp_i \le q_i \qquad i = 1, 2, 3, 4$$

Where parameters p and q are defined as

$$p_1 = -\Delta x, \quad q_1 = x_1 - x_{wmin} \qquad \text{(Left Boundary)}$$
$$p_2 = \Delta x, \quad q_2 = x_{wmax} - x_1 \qquad \text{(Right Boundary)}$$
$$p_3 = -\Delta y, \quad q_3 = y_1 - y_{wmin} \qquad \text{(Bottom Boundary)}$$
$$p_4 = \Delta y, \quad q_4 = y_{wmax} - y_1 \qquad \text{(Top Boundary)}$$

From above, we can conclude that;

1. If (p_i=0), the line is parallel to the i^{th} boundary.
    a. If ($q_i < 0$), line is completely outside the boundary and hence can be eliminated.
    b. If ($q_i \ge 0$), line is inside the boundary, and needs further processing.
2. If ($p_i \ne 0$), line or extended line intersects the clipping boundary, and in this case parameter t is calculated.

Liang-Barsky algorithm calculates two values of Parameter t; t1 and t2 that define that part of the line that lies within the clip rectangle. t1, and t2 are called **line intersection parameters**.

Following observations can be easily made from above definitions of parameters p and q.

If $p_1 = 0$: Line is parallel to left clipping boundary.
If $p_2 = 0$: Line is parallel to right clipping boundary.
If $p_3 = 0$: Line is parallel to bottom clipping boundary.
If $p_4 = 0$: Line is parallel to top clipping boundary.
If $p_i = 0$: Line is parallel to one of the clipping boundaries corresponding to the value of i.
If $q_i < 0$: Line is completely outside the boundary and can be eliminated.
If $q_i \geq 0$: Line is inside the clipping boundary.
If $p_i < 0$: Line proceeds from outside to inside of the clipping boundary.

If $p_i > 0$: Line proceeds from inside to outside of the clipping boundary.

Algorithm

1. Read two endpoints of the line say p_1 (x_1, y_1) and p_2 (x_2, y_2).
2. Read two corners (left-top and right-bottom) of the window, say $(x_{wmin}, y_{wmax}, x_{wmax}, y_{wmin})$
3. Calculate the values of parameters p_i and q_i for i = 1, 2, 3, 4 such that

   $p_1 = -\Delta x$ $q_1 = x_1 - x_{wmin}$
   $p_2 = \Delta x$ $q_2 = x_{wmax} - x_1$
   $p_3 = -\Delta y$ $q_3 = y_1 - y_{wmin}$
   $p_4 = \Delta y$ $q_4 = y_{wmax} - y_1$
4. if $p_i = 0$, then
   { The line is parallel to i^{th} boundary.
   Now, if $q_i < 0$ then
   { line is completely outside the boundary, hence
   discard the line segment and goto stop.
   }

5. Initilize values for t1, and t2 as;

t1= 0, and t2 = 1.

6. Calculate r_i as;

$r_i = qi / p_i$, for i=1,2,3,4.

7. Find all r_i for which $p_i < 0$.

Set t1= max(0, r_i)

8. Find all r_i for which $p_i > 0$.

Set t2= min(1, r_i)

9. If (t1 > t2)

Line lies completely outside the window, eliminate the line segment.

Else

{

If(t1=0 and t2=1)

Line lies completely inside window, display the line segment.

Else

{

Calculate the endpoints of the clipped line as follows;

$xx_1 = x_1 + t_1 \Delta x$

$xx_2 = x_1 + t_2 \Delta x$

$yy_1 = y_1 + t_1 \Delta y$

$yy_2 = y_1 + t_2 \Delta y$

Draw line (xx_1, yy_1, xx_2, yy_2)

}

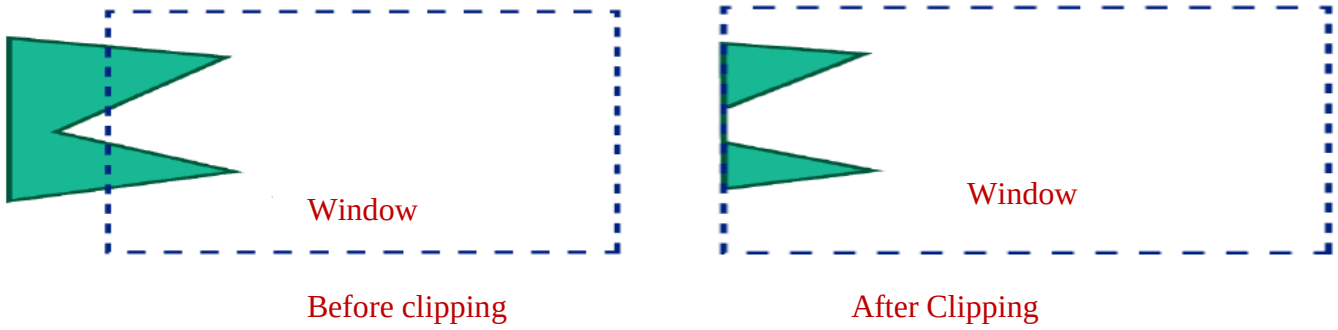
}

10. End.

3. Polygon Clipping

Collection of connected lines is considered as Polygon. A polygon can be defined as a geometric object "consisting of a number of points (called vertices) and an equal number of line segments (called sides or edges)". Polygon clipping is the process of removing those parts of a polygon that lie outside of a clipping window.

- Polygon clipping includes:
  - Discarding existing edges.
  - Dividing existing edges.
  - Adding new edges.
  - Retaining existing edges.
- Thus, Multiple polygons may result from clipping a single polygon.



3.1 Sutherland-Hodgeman Polygon Clipping Algorithm

Sutherland and Hodgeman's polygon-clipping algorithm uses a divide-and-conquer strategy. It clips a given polygon successively against the edges of the given clip window. Each edge of the polygon must be tested against each edge of the clip rectangle. It starts with the initial set of polygon vertices.

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

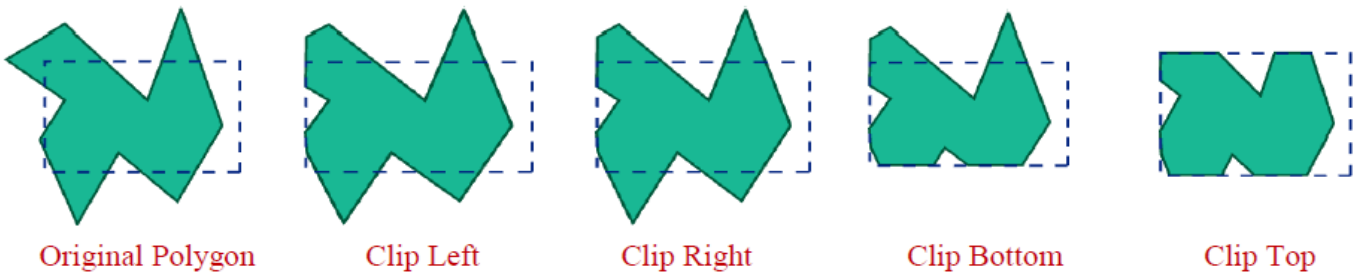
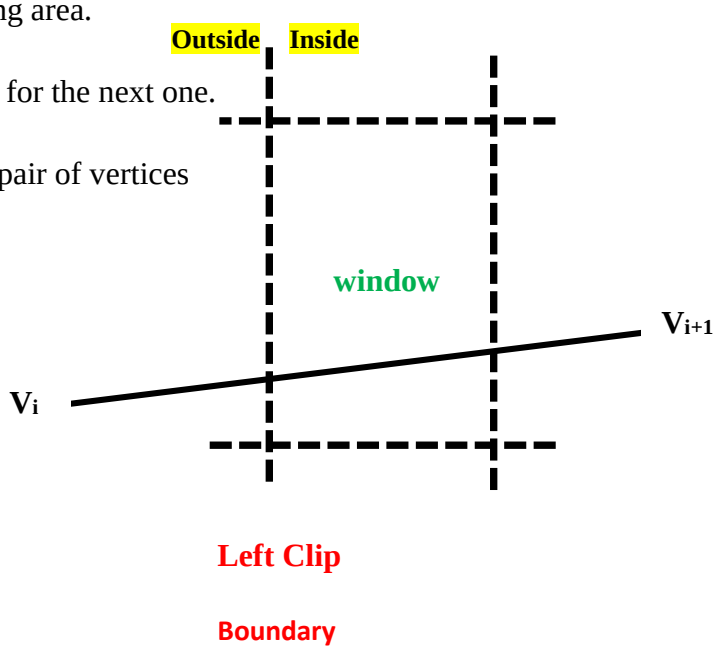


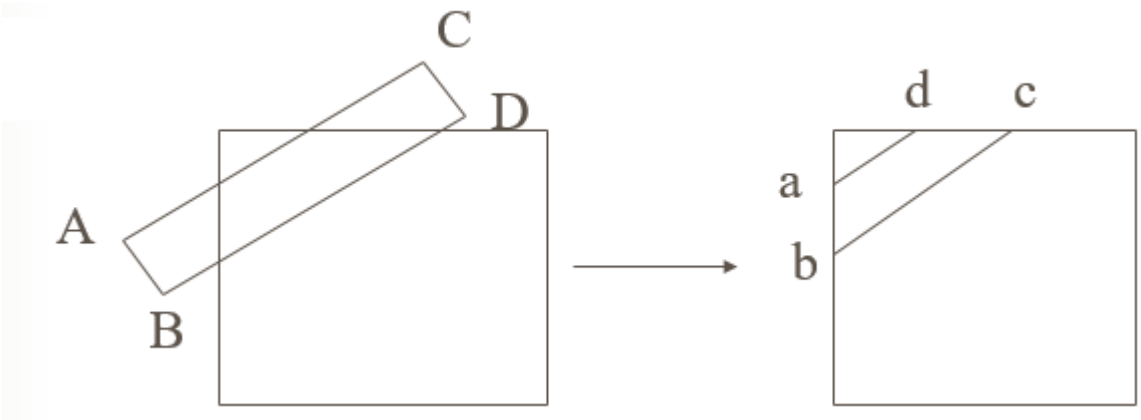
Fig: Clipping a polygon against successive window boundaries.

❖ **Basic approach**:

- Clip a polygon against each clip window edge i.e., left, right, bottom, and top. Clip polygon against all four edges. Each successive clipped polygon sent to next edge. Vertices which are kept after clipping against one window edge are saved for clipping against the remaining edges.
- **Inside-Outside Test: Perform a test for each pair of adjacent vertices to check whether** they are inner or outer side of each edge of the clipping area.
  - ➢ The output of each clipping edge is input for the next one.
  - ➢ Figure shows, inside and outside test for pair of vertices V_i V_{i+1}, against left clip boundary.



❖ **Post Process:** A closed Polygon when clipped provides one or more open polygon or lines, as shown in figure below. So, after clipping the polygon, resulting polygon should be closed. That is, we need to add line segments to form the closed polygon.



It requires that lines ab & cd be added to make it closed polygon.

Algorithm:

- For each clip window boundary perform the following steps:
  1. Construct an input vertex list ($v_0, v_1 \ldots v_n$) where $v_0 = v_n$.
  2. Create empty output vertex list.
  3. For each pair of adjacent vertices v_i and v_{i+1} perform the following *inside-outside* test:
     a. If out-in (v_i is outside the window boundary and v_{i+1} is inside),
        - *add intersection point v_i' and v_{i+1} to the output vertex list.*

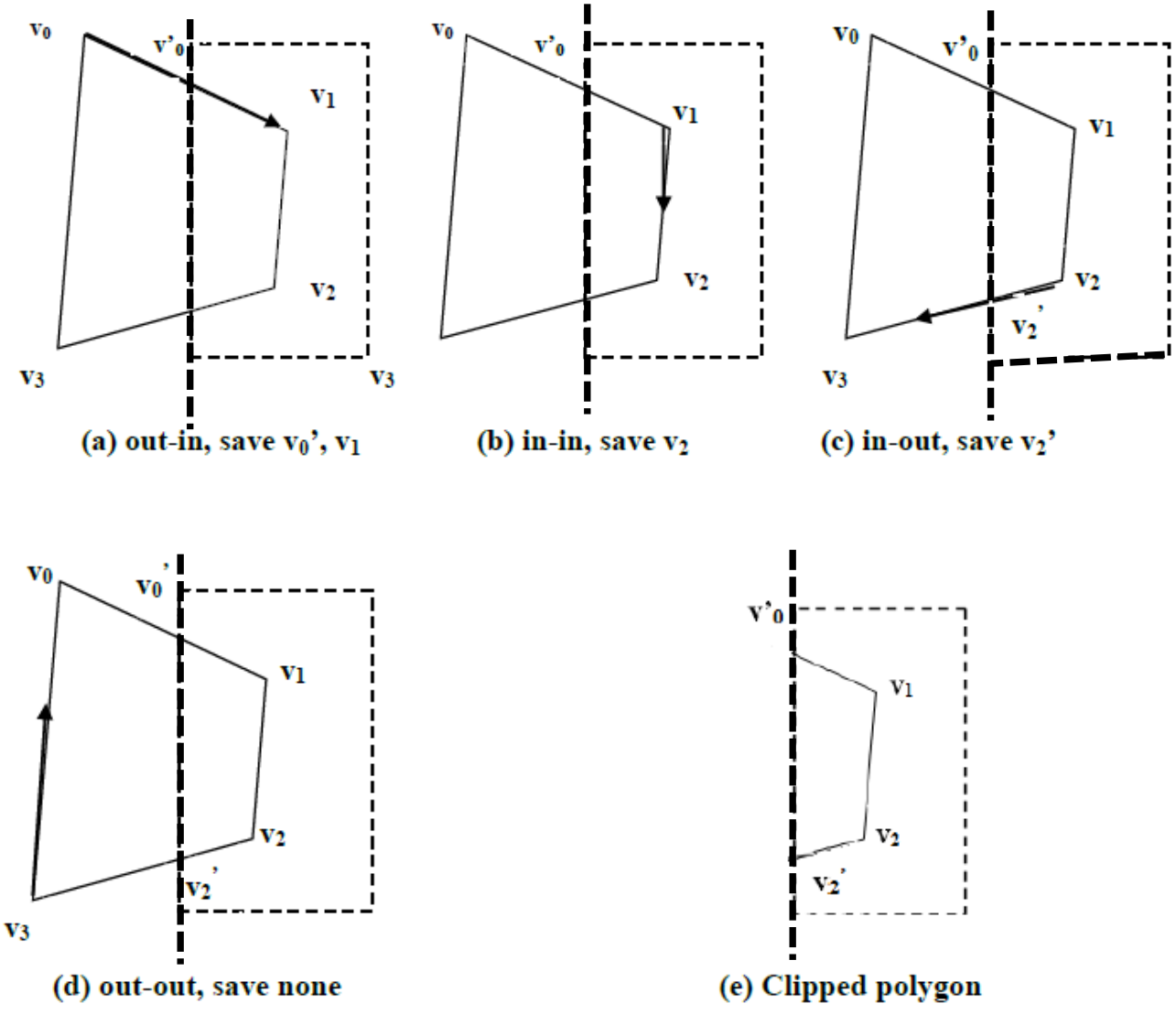
     b. If in-in (both v_i, v_{i+1} are inside the window boundary),
        - *add v_{i+1} to output vertex list.*

     c. If in-out (v_i is inside the window boundary and v_{i+1} is outside),
        - *add intersection point v_i' to output vertex list.*

     d. If out-out (both v_i, v_{i+1} are outside the window boundary),
        - *add nothing to output vertex list.*

4. Apply post process, if required.

Example:



Clipping against left clip boundary:

Input vertex list: {v_0, v_1, v_2, v_3}

Output vertex list: { }.

For vertices v_0, v_1, *out - in* pair so intersection point v_0' and v_1 are added to the output vertex list.

Therefore, output vertex list = {v_0', v_1 }.

For vertices v_1, v_2, *in-in* pair so only v_2 is added to the output vertex list.

Now, output vertex list = {v_0', v_1, v_2 }.

For vertices v_2, v_3, *in-out* pair so only intersection point v_2' is added to the output vertex list.

Now, output vertex list = { v_0', v_1, v_2, v_2'}.

For vertices v_3, v_0, *out-out* pair so nothing is added to the output vertex list.

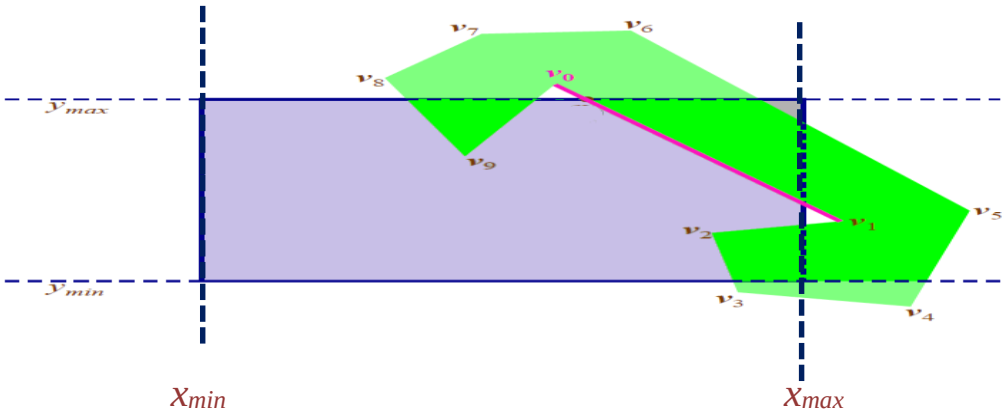
Finally, output vertex list = { v_0', v_1, v_2, v_2'}.

Post process: connect v2', and v1' to form the closed polygon.

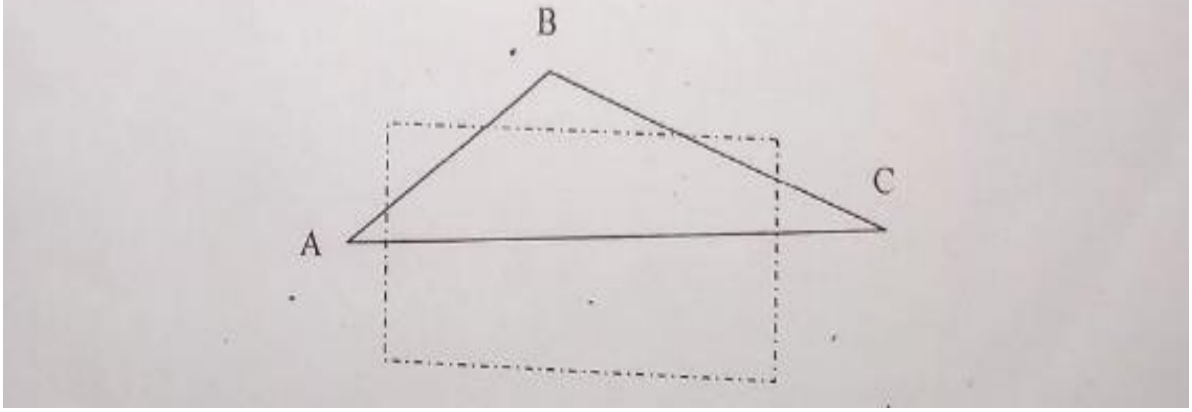
Similar, process is repeated for right, bottom, and top boundary.

Exercise:

- Clip the given polygon using Sutherland-Hodgeman Polygon clipping Algorithm.



Explain polygon clipping in detail. By using the Sutherland-Hodgemen Polygon clipping algorithm clip the following polygon.



4. Curve Clipping

- Curve clipping procedures will involve non-linear equations (so requires more processing than for objects with linear boundaries).
- The bounding rectangle (coordinate extent) for a circle or other curved object is used to test for overlap with a rectangular clip window.
- For a circle, we can use the coordinate extents of individual quadrants and then octants before calculating curve-window intersections.
- For an ellipse, we can test the coordinate extents of individual quadrants.

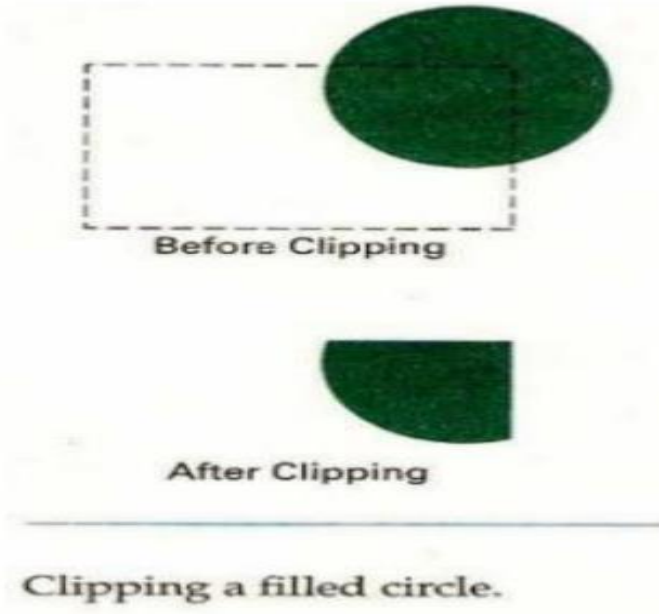
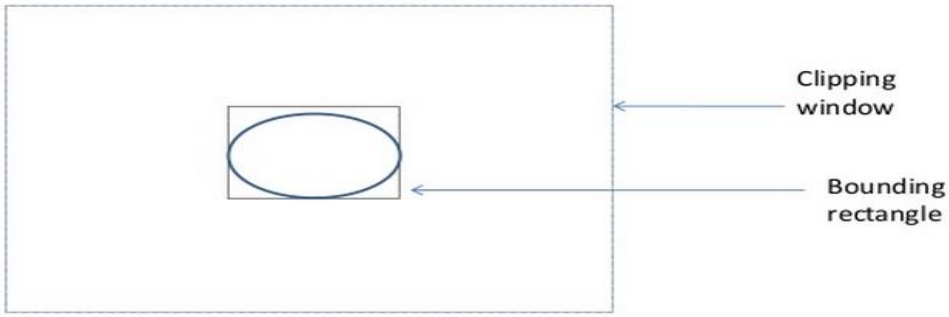


Fig: Showing circle clipping against a window

Algorithm:

- If the bounding rectangle (coordinate extent) for the object is completely inside the window, we save the object.



- If the rectangle is determined to be completely outside window, we discard the object.

- Otherwise
  - Calculate the intersection points of the curve with each clip edge by solving simultaneous line-curve equations.

Chapter 6

3D Concepts

6.1 Three Dimensional (3D) Transformations

Just as 2D-transfromtion can be represented by 3x3 matrices using homogeneous co-ordinate can be represented by 4x4 matrices, provided we use homogenous co-ordinate representation of points in 3D space as well.

6.1.1 Translation

Translation in 3D is similar to translation in the 2D except that there is one more direction parallel to the z-axis.

If, t_x, t_y, and t_z are used to represent the translation vectors. Then the translation of the position P(x, y, z) into the point P' (x', y', z') is done by

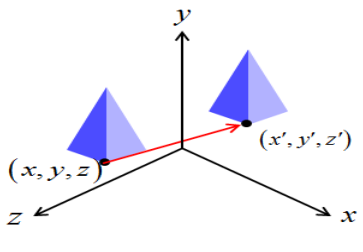
$x' = x + t_x$

$y' = y + t_y$

$z' = z + t_z$

In matrix notation using homogeneous coordinate this is performed by the matrix multiplication,

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



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

Fig: Translation of a point (x, y, z) to(x', y', z')

Inverse translation,

$T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z)$

6.1.2 Rotation

To generate a rotation transformation for an object, we must designate an axis of rotation (about which the object is to be rotated) and the amount of angular rotation.

(i) Rotation About z-axis:

- Z-component does not change.

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

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

  $z' = z$

Matrix representation for rotation around z-axis,

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

(ii) Rotation about x-axis:

- X-component does not change.

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

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

  $x' = x$

Matrix representation for rotation around x-axis,

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

(iii) Rotation about y-axis:

- Y-component does not change.

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

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

  $y' = y$

Matrix representation for rotation around y-axis,

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

(iv) General 3D Rotations:

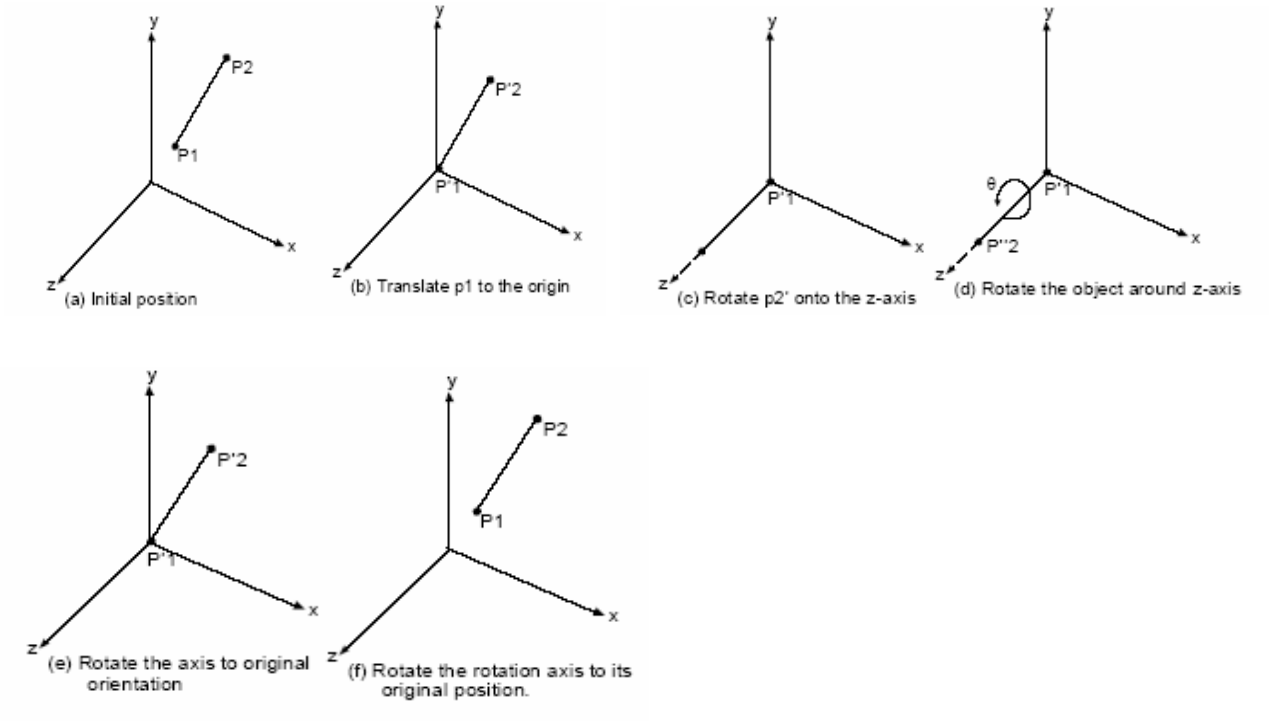
(a) Rotation about an axis parallel to any of the coordinate-axis: *When an object is to be rotated about an axis that is parallel to one of the co-ordinate axis, we need to perform series of transformation.*

1. Translate the object such that the rotation axis coincides with the parallel co-ordinate axis.
2. Perform the specified rotation about the axis.
3. Translate the object so that the rotation axis is moved to its original position.

(b) **Rotation about an axis that is not parallel to any of the coordinate-axis:**

1. Translate the object such that rotation axis passes through co-ordinate origin.
2. Rotate the axis such that axis of rotation coincides with one of the co-ordinate axis.
3. Perform the specified rotation about the coordinate axis.
4. Apply inverse rotation to bring the rotation axis back to its original orientation.

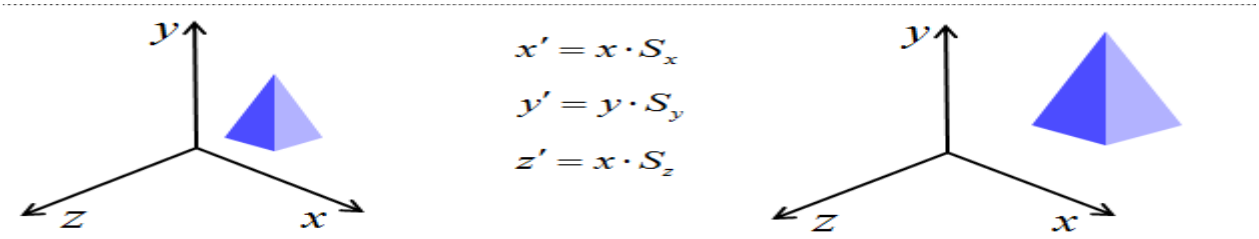
5. Apply inverse translation to bring the rotation axis back to its original position.



6.1.3 3D Scaling

- Scaling means changing the size of object.

(i) About origin:



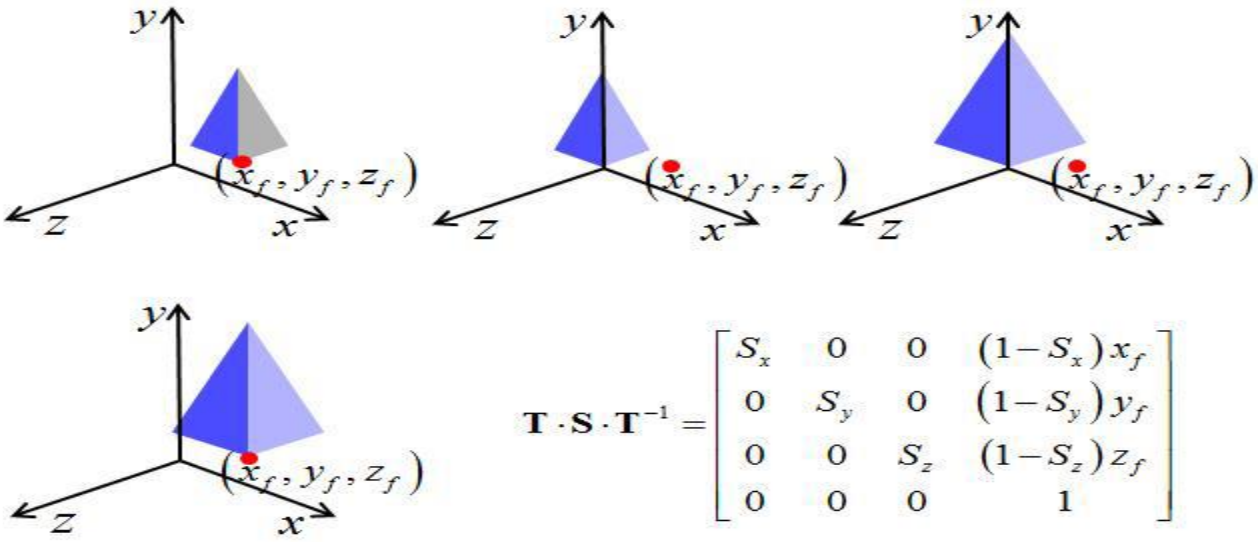
Enlarging object also moves it from origin

$$\mathbf{P}' = \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{S} \cdot \mathbf{P}$$

Fig: Scaling about origin and scaling matrix

(ii)Scaling about an arbitrary point (x_f, y_f, z_f)

- Translate the fixed point to the origin.
- Scale the object relative to the coordinate origin.
- Translate the fixed point back to its original position.



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

Fig: Scaling about an arbitrary point (x_f, y_f, z_f) and Scaling matrix

6.1.4 Reflection

In 3D reflection the reflection takes place about a plane whereas 2D reflection it used take place about an axis.

Transformation matrix for reflection through X-Y plane is

$$T_{xy} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix for reflection through Y-Z plane is

$$T_{yz} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix for reflection through Z-X plane is

$$T_{yz} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.1.5 Shear

❖ Shearing transformations can be used to modify object shapes.

Z-axis Shear

This transformation alters x- and y-coordinate values by an amount that is proportional to the z value, while leaving the z coordinate unchanged, i.e.

$x' = x + s_{hx}.z$

$y' = y + s_{hy}.z$

$z' = z.$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & S_{hx} & 0 \\ 0 & 1 & S_{hy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

, Parameters s_{hx} and s_{hy} can be assigned any real values.

Similarly, we can find X-axis shear and Y-axis shear.

$$SH_z = \begin{bmatrix} 1 & 0 & S_{hx} & 0 \\ 0 & 1 & S_{hy} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad SH_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ S_{hx} & 1 & 0 & 0 \\ S_{hx} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad SH_y = \begin{bmatrix} 1 & S_{hx} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & S_{hx} & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Exercise

Given solid PQR, P(2, -5, 0), Q(4, -2, 3) and R(6, -7, -1) is translated with T(9, 0, -2) and then scaled about origin with scale factor (2, 4, 5). Find the final co-ordinates.

A 3D object with vertices A (-2, -4, 4), B(3, -6, -8), C(-6, 1, 0) and D (3, -6, 2) is required to be scaled with scale factor S (2, 4, 6) about origin. Find the final coordinates.

6.2 Three Dimensional Viewing (3D Viewing)

It is the process of mapping 3D object descriptions into 2D device coordinates. It is the process of selecting different views of a 3D object, and mapping those selected views on to 2D output devices. Part of a 3D scene that is selected for displaying is called a window. And part of the screen coordinate on to which a window is mapped is called a viewport.

The steps for computer generation of a view of 3D scene are analogous to the process of taking photograph by a camera.

For a snapshot, we need to position the camera at a particular point in space and then need to decide camera orientation. Finally when we snap the shutter, the seen is cropped to the size of window of the camera and the

light from the visible surfaces is projected into the camera film. Figure below show the conceptual model of the 3D-viewing process.

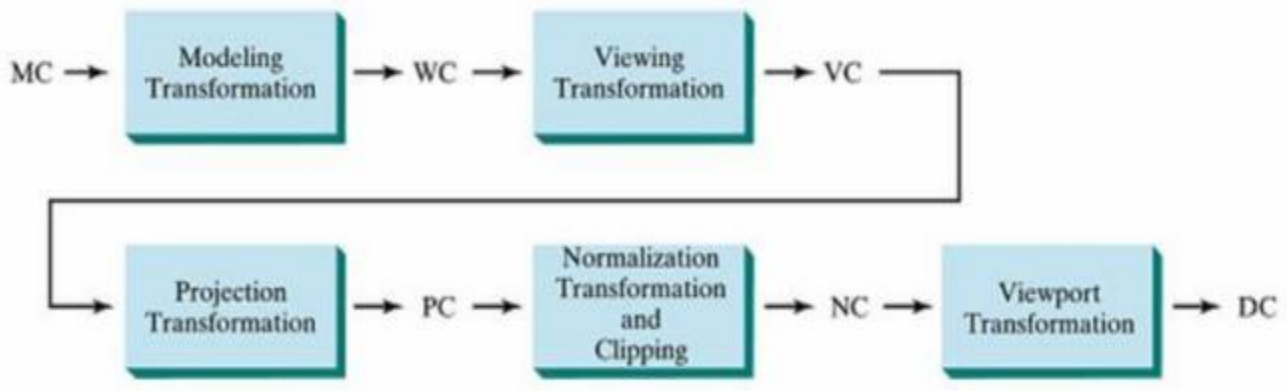


Fig: *General three-dimensional transformation pipeline from modeling coordinates to final device Coordinates*

Problem: how can we map 3D objects onto 2D screen coordinates i.e. monitor, mobile devices, printout etc?

Solution: Projection

6.2.1 Projections

- In general, the process of transformation of a n-dimensional points to m-dimensional points (m<n) is called projection.
- In computer graphics, projection is the process of conversion of 3D objects to a 2D screen coordinates or device coordinates. It means the transformation of a three-dimensional (3D) area into a two-dimensional (2D) area. The plane in the area into which we transform (project) objects is called the view plane or projection plane.
- Thus, projection means a shadow of an object.
- 2D projected images are formed by the intersection of projection lines with a plane (projection plane or view plane).
- Ideally, an object is projected by projecting each of its endpoints and joining these projected points by straight line in a 2D plane.

6.2.2 Basic Terminologies related to projection

- ***Center of projection (COP):*** Projector's converging point. It is a point from where projection is taken. It can be either light source or eye position.
- ***Projection plane or view plane (VP):*** The plane on which the projection of the object is formed.
- ***Projectors***: Lines emerging from COP and hitting the projection plane and passing through a point in the object to be projected.

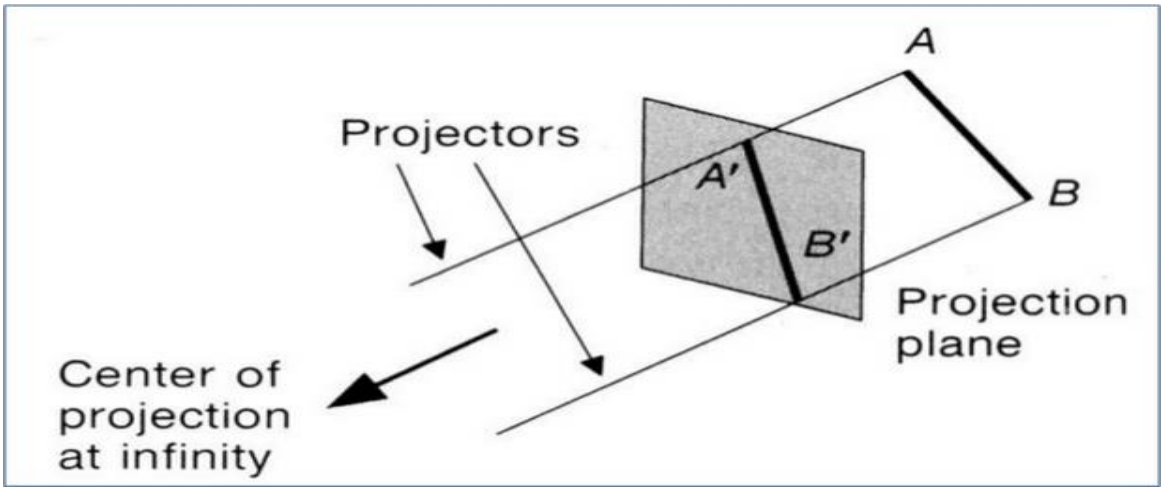


Fig. (a): Parallel Projection

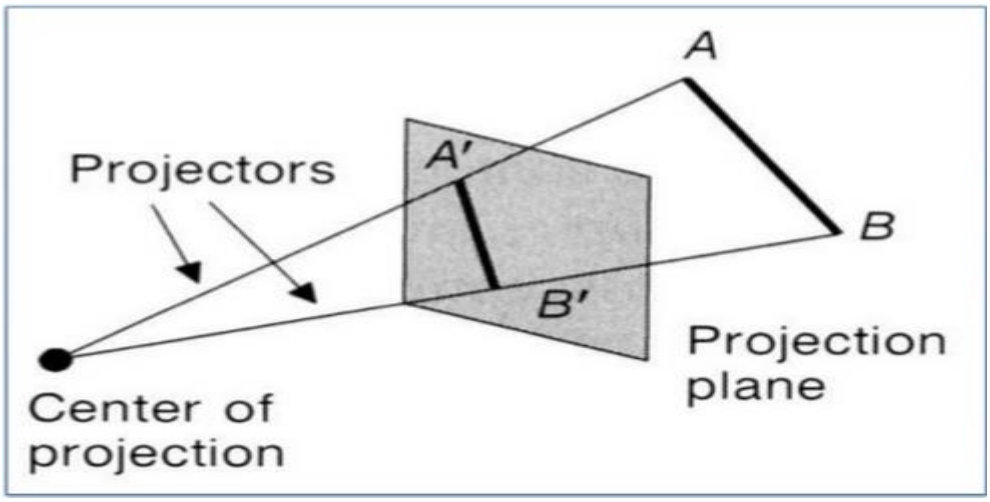


Fig. (b): Perspective Projection

6.2.3 Categories of Projection

- Basically, there are two categories of projections
  - **Parallel projection**
  - **Perspective projection**

Parallel Projection

- In parallel projection, the coordinate positions are transformed to the view plane along parallel lines i.e., all projectors are parallel to each other.
- Parallel lines are still parallel after projection.
- This type of projection is often referred by the direction of projection(DOP).
- The center of projection is at infinity, shown in fig(a). That is, distance from COP to PP is infinite.
- Usually represented as a direction of projection.

- A parallel projection preserves the relative proportions of the object but it does not give us realistic representation.

In parallel projection, z - coordinate is discarded and parallel lines from each vertex on the object are extended until they intersect the view plane. The point of intersection is the projection of the vertex. We connect the projected vertices by line segments which correspond to connections on the original object.

- It is used in engineering and architectural drawings.

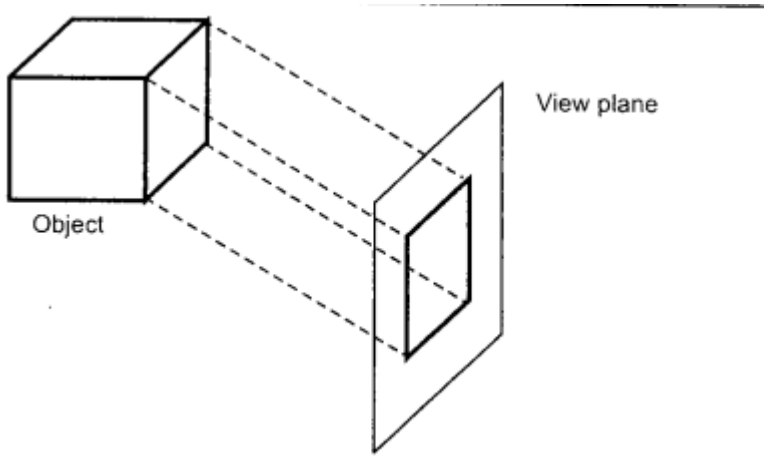
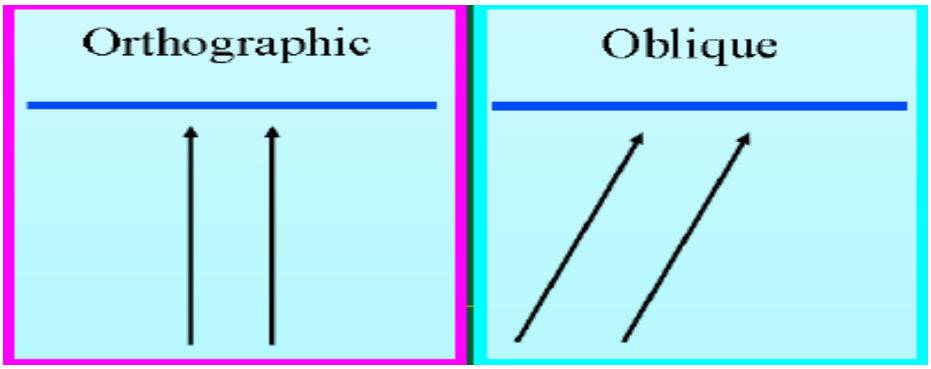


Figure: Parallel projection of an object

- The most common types of parallel projections are:
  - **Orthographic Projection:** Projectors (projection vectors) are perpendicular to the projection plane. That is the angle between projectors and the view plane is equal to 90º.
  - **Oblique Projection:** Projectors (projection vectors) are not perpendicular to the projection plane. It preserves 3D nature of an object. That is the angle between projectors and the view plane is not equal to 90º.



Perspective Projection

- Projectors are not parallel to each other.
- The center of projection is at a finite distance. That is, distance from COP to PP is finite
- In perspective projection, object positions are transformed to the view plane along lines that converge to a point called as a center of projection.

- The projected view of an object is determined by calculating the intersection of the projection lines with the view plane.
- Produces realistic views but does not preserve relative proportions.
- Used in realistic displaying because it resembles to that our photographic systems (camera) and human eye.
  - Size of object varies inversely with distance from the center of projection. Projection of a distant object are smaller than the projection of objects of the same size that are closer to the projection plane.

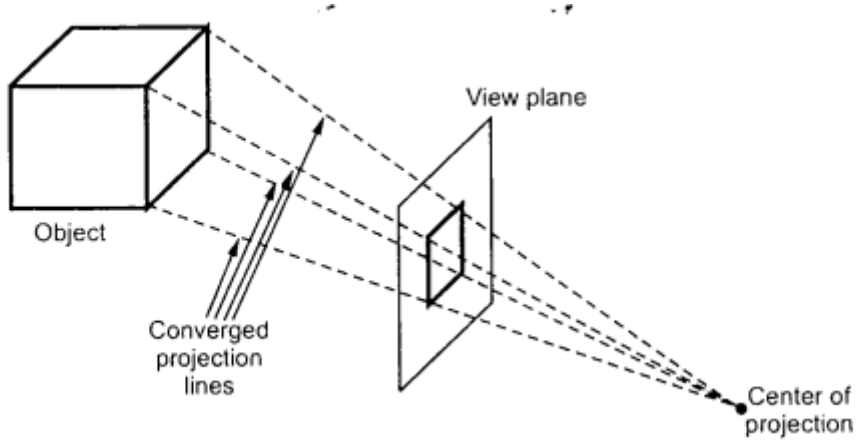


Figure: Perspective projection of an object.

6.3 3D object representation

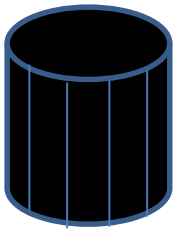
In 3D computer graphics, 3D modeling (or three-dimensional modeling) is the process of developing a mathematical representation of any three-dimensional surface of an object via specialized software.

Graphical scenes can contain many different kinds of objects like trees, flowers, rocks, water...etc. To produce realistic display of scenes, we need to use representations that accurately model object characteristics.

Simple Euclidean objects like polyhedrons and ellipsoids can be represented by polygon and quadric surfaces. Spline surface are useful for designing aircraft wings, gears and other engineering objects. Procedural methods and particle systems allow us to give accurate representation of clouds, clumps of grass, and other natural objects. Octree encodings are used to represent internal features of objects such as medical CT images.

3D object representation is divided into two categories. Representation schemes for solid objects are often divided into two broad categories:

- **Boundary Representations (B-reps):** Boundary representations (B-reps) describe a three-dimensional object as a set of surfaces that separate the object interior from the environment. Objects are represented as a collection of surfaces. **Example:** *Polygon surfaces, spline patches.*



❖ **Space-partitioning Representations:** Space-partitioning representations are used to describe interior properties, by partitioning the spatial region containing an object into a set of small, non-overlapping, contiguous solids (usually cubes). A common space-partitioning description for a three-dimensional object is an octree representation. **Example: Quadtree,** *Octree representations.*

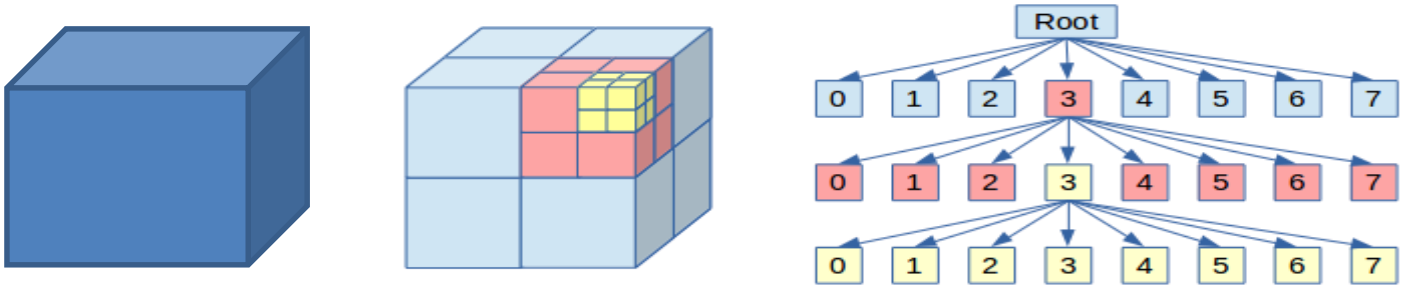


Figure: Recursive subdivision of a cube into octants and the corresponding octree.

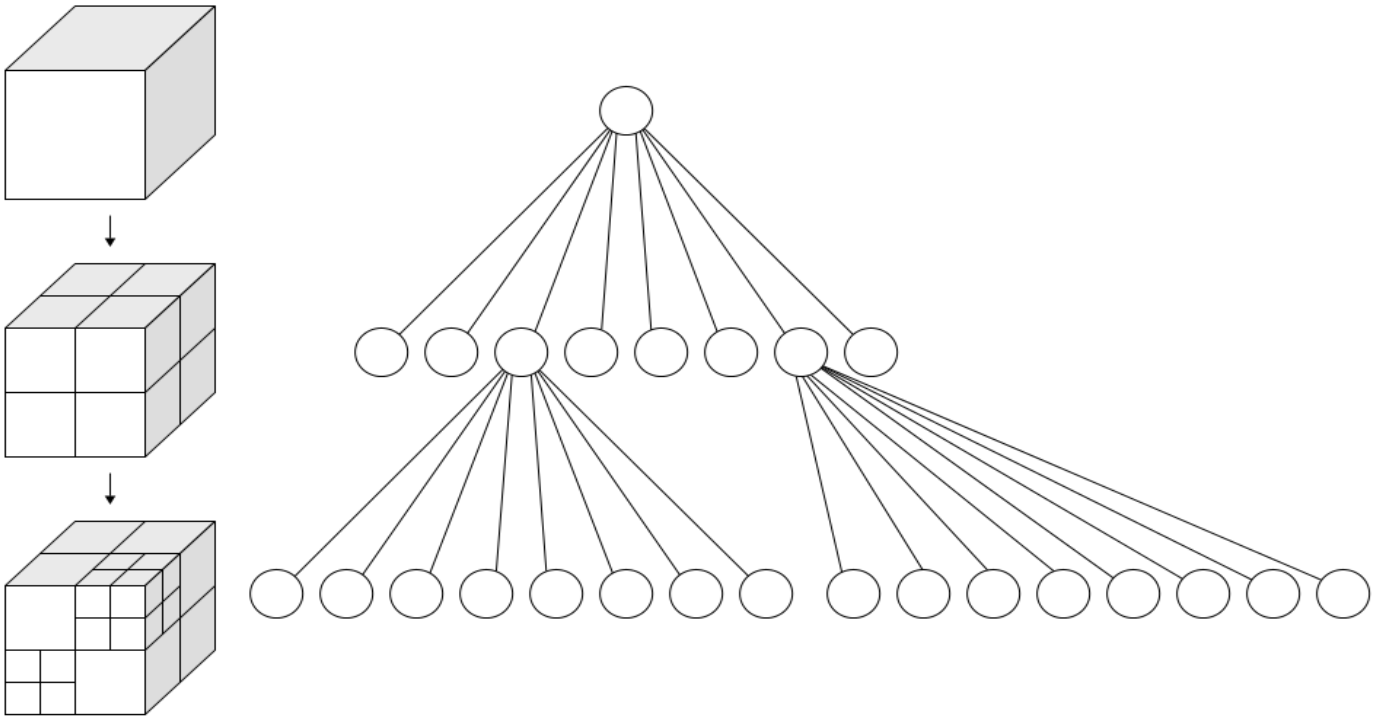


Figure: *Left: Recursive subdivision of a cube into octants. Right: The corresponding octree.*

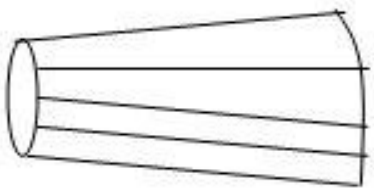
6.3.1 Boundary Representation

Each 3D object is supposed to be formed by its surfaces and is represented by collection of polygon surfaces. Polygon Surfaces are the most widely used boundary representation. Other methods such as Wireframe

Representation, Blobby Objects, Quadric Surfaces, Spline Representation, Bezier Curve and Surfaces are also used.

1. Polygon Surfaces

It is most common representation for 3D graphics object. In this representation, a 3D object is represented by a set of surfaces that enclose the object interior. Many graphics system use this method. Object descriptions are stored as a set of surface polygons. Therefore, all surfaces can be described with linear equations. This simplifies and speeds up the surface rendering (creating high quality realistic images or pictures) and display of object. For this reason, polygon descriptions are often referred to as "standard graphics objects".



A 3D object represented by polygons

Generally polygon surfaces are specified using;

- ❖ Polygon Table
- ❖ Polygon Meshes
- ❖ Plane Equations

1.1 Polygon Tables

In this method, the surface is specified by the set of vertex coordinates and associated attributes. Polygons tables can be used to specify polygon surfaces. Polygon table representation of polygon surfaces contains two parts:

- ❖ **Geometric table**

a. It contains vertex co-ordinates and other parameters to identify the spatial orientation of polygon surface.

b. Geometric data consists of three tables: a vertex table, an edge table, and a surface table.

i. **Vertex table:** It is used to store co-ordinate values for each vertex of the object. Each vertex stores x, y, and z coordinate information which is represented in the table as v1: x1, y1, z1.

ii. **Edge Table:** The Edge table is used to store the edge information of polygon. The edge table contains pointers back into the vertex table to identify the vertices for each

polygon edge. If E1 is an edge between vertex v1 and v2 which is represented in the table as E1: v1, v2.

iii. **A polygon Surface table:** Polygon surface table stores the number of surfaces present in the polygon. A polygon surface table contains pointers back into the edge table to identify the edges for each polygon surfaces. If surface S1 is covered by edges E1, E2 and E3 which can be represented in the polygon surface table as S1: E1, E2, and E3.

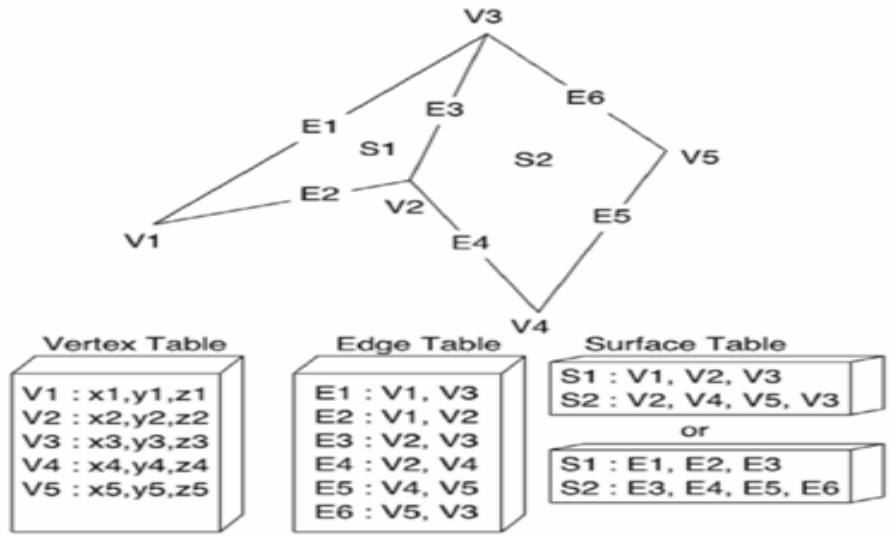


Fig: **Geometric data table representation for two adjacent polygon surfaces, formed with 6 edges and 5 vertices.**

❖ Attribute table

- It gives attribute information for an object (degree of transparency, surface reflectivity, color etc.)

1.2 Polygon Meshes

3D surfaces and solids can be approximated by a set of polygon faces and line elements. Such surfaces are called polygonal meshes. A polygon mesh is a collection of edges, vertices and faces connected such that each edge is shared by at most two polygons. An edge connects two vertices and a polygon is a closed sequence of edges. **An edge can be shared by two polygons and a vertex is shared by at least two edges.** The faces usually consist of triangles (triangle strip), quadrilaterals (quadrilateral mesh), or convex polygons.

One type of polygon mesh is the triangle strip. This produces n-2 connected triangles, where n is the number of vertices, as shown in figure (a).

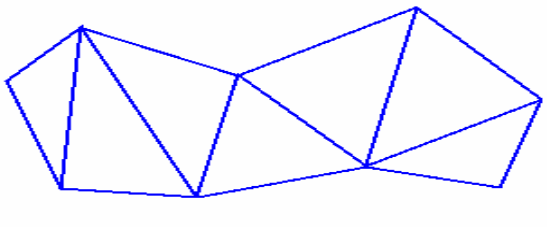


Fig (a): A triangle strip formed by triangles

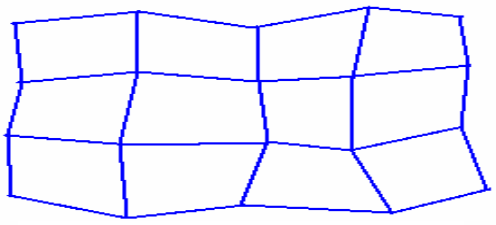


Fig (b): A quadrilateral mesh formed by quadrilaterals

Another type of polygon mesh is the quadrilateral mesh, which produces a mesh of (n-1) by (m-1) quadrilaterals, given the co-ordinates for an n*m array of vertices, as shown in figure (b). It shows 20 vertices forming a mesh of 12 quadrilaterals.

Representing polygon meshes

1. Explicit representation

2. Pointers to a vertex list

3. Pointers to an edge list

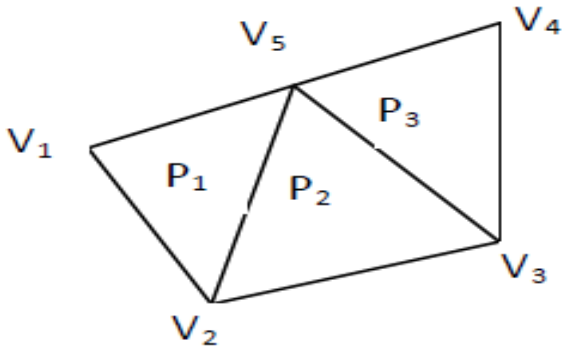


Fig: A triangle strip formed by triangles.

Explicit representation

- Simplest representation, each face is represented by a list of vertex co-ordinates.

  P=((x1, y1 ,z1), (x2, y2 ,z2),....., (xn, yn, zn))

- The vertices are stored in the order in which they are traversed.
- There are edges between successive vertices in the list and between the last and first vertices.
- For a single polygon it is efficient but for polygon mesh it is not space efficient since coordinates of shared vertices may duplicate.
  - For above mesh;
    - P1=((x1, y1 ,z1), (x2, y2 ,z2), (x5, y5 ,z5))
    - P2=((x2, y2 ,z2), (x3, y3 ,z3), (x5, y5 ,z5))
    - P3=((x3, y3 ,z3), (x4, y4 ,z4), (x5, y5 ,z5))
    -

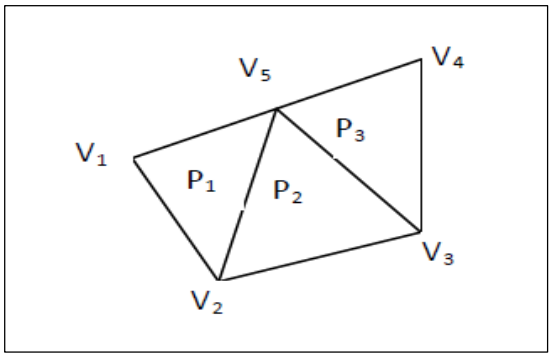
Pointers to a vertex list

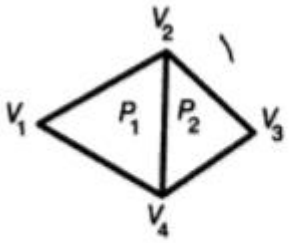
Polygons defined with *pointers to a vertex list*, the method used by SPHIGS, have each vertex in the polygon mesh stored just once, in the vertex list $V = ((x_1, y_1, z_1), \ldots, (x_n, y_n, z_n))$. A polygon is defined by a list of indices (or pointers) into the vertex list. A polygon made up of vertices 3, 5, 7, and 10 in the vertex list would thus be represented as $P = (3, 5, 7, 10)$.

- Define polygon with pointers to a vertex list. So each vertex is stored just once, in vertex list. A polygon is defined by list of indices (pointers) into the vertex list
- It represents each face as a vertex list, such as.

  V= {v0, v1, …, vn}

  = ((x1, y1 ,z1), (x2, y2 ,z2), ...,(xn, yn ,zn))
  - For given mesh;
    - P1={v1, v2, v5}
    - P2={v2, v3, v5}
    - P3={.v3, v4, v5}





$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \ldots, (x_4, y_4, z_4))$

$P_1 = (1, 2, 4)$

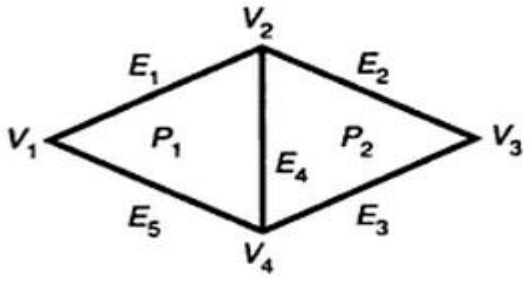
$P_2 = (4, 2, 3)$

Fig. Polygon mesh defined with indexes into a vertex list.

Pointers to an edge list

- In this method, we again have a vertex list V, represent a polygon as a list of pointers not to the vertex list but rather to an edge list, in which each edge occurs just once.
- Each edge in edge list points to the two vertices in the vertex list, and also to the one or two polygons, to which the edge belongs.
- Hence we describe polygon as a collection of ;

  V= (V_1, V_2,…,Vn), P= (E_1, E_2...E_n), and an edge as E= (v1, v2, P_1, P_2).
- When an edge belongs to only one polygon face, either P1 or P2 is null.



$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \ldots, (x_4, y_4, z_4))$
$E_1 = (V_1, V_2, P_1, \lambda)$
$E_2 = (V_2, V_3, P_2, \lambda)$
$E_3 = (V_3, V_4, P_2, \lambda)$
$E_4 = (V_4, V_2, P_1, P_2)$
$E_5 = (V_4, V_1, P_1, \lambda)$
$P_1 = (E_1, E_4, E_5)$
$P_2 = (E_2, E_3, E_4)$

Fig. Polygon mesh defined with edge lists for each polygon (λ represents null).

1.3 Plane Equations

Plane equation method is another method for representation of the polygon surface for 3D object. The information about the spatial orientation of object is described by its individual surface, which is obtained by the vertex co-ordinates and the equation of each surface. The equation for a plane surface can be expressed in the form:

$Ax + By + Cz + D = 0$

Where (x, y, z) is any point on the plane, and A, B, C, D are constants describing the spatial properties of the plane. The values of A, B, C, D can be obtained by solving a set of three plane equations using coordinate values of three non-collinear points on the plane.

Where x, y, z is any point on the plane, and the coefficients A, B, C, and D are constants describing the spatial properties of the plane. We can obtain the values of A, B, C, and D by solving a set of three plane equations using the coordinate values for three non collinear points in the plane.

Let us assume that (x_1, y_1, z_1), (x_2, y_3, z_2) and (x_3, y_3, z_3) are three such points on the plane then,

$Ax_1 + By_1 + Cz_1 + D = 0$

$Ax2 + By2 + Cz2 + D = 0$

$Ax3 + By3 + Cz3 + D = 0$

Let us solve the following simultaneous equations for ratios A/D, B/D, and C/D. We get the values of A, B, C, and D.

$(A/D)x_1 + (B/D)y_1 + (C/D)z_1 = -1,$

$(A/D)x_2 + (B/D)y_2 + (C/D)z_2 = -1,$

$(A/D)x_3 + (B/D)y_3 + (C/D)z_3 = -1,$

$A/D(-x_1) + B/D(-y_1) + C/D(-z_1) = 1,$

$A/D(-x_2) + B/D(-y2) + C/D(-z_2) = 1,$

$A/D(-x_3) + B/D(-y3) + C/D(-z_3) = 1,$

The solution of these equations can be obtained in determinant form using Cramer's rule as:-

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \quad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix} \quad C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \quad D = -\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

2. Wireframe Representation

A wire-frame model is a visual presentation of a 3-dimensional (3D) or physical object used in 3D computer graphics. It is created by specifying each edge of the physical object where two mathematically continuous smooth surfaces meet, or by connecting an object's constituent vertices using straight lines or curves. The object is projected into screen space by drawing lines at the location of each edge. The term wire frame comes from designers using metal wire to represent the three-dimensional shape of solid objects. 3D wireframe allows to construct and manipulate solids and solid surfaces.

A wireframe model represents the shape of a solid object with its characteristic lines and points. There are two types of wireframe modelling: Pro's and Con's. In Pro's user gives a simple input to create a shape. It is useful in developing systems. While in Con's wireframe model, it does not include information about inside and outside boundary surfaces.

Today, wireframe models are used to define complex solid objects. The designer makes a wireframe model of a solid object, and then the CAD operator reconstructs the object, including detailed analysis.

❖ **This technique has some advantages:**

- Generally the 3-dimensional solid objects are complex, but wireframe models can be viewed in 1 dimension.
- The solid object can be modified further; the designer can ignore the geometry inside a surface.
- Wireframe models require less memory space and CPU capacity.
- Using a wire-frame model allows visualization of the underlying design structure of a 3D model.

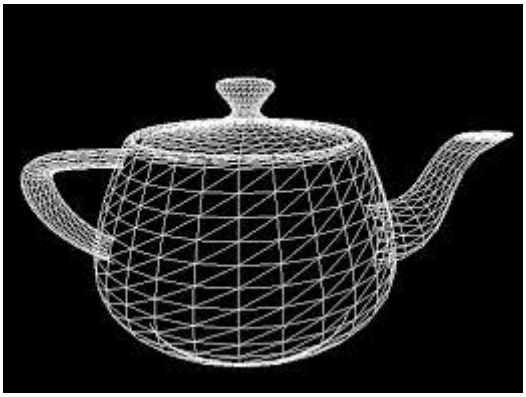


Fig: Wireframe representation of a 3D object.

❖ **Simple example of wireframe model**

An object is specified by two tables: (1) Vertex Table, and, (2) Edge Table. The vertex table consists of three-dimensional coordinate values for each vertex with reference to the origin. Edge table specifies the start and end vertices for each edge.

Vertex	X	Y	Z
1	1	1	1
2	1	-1	1
3	-1	-1	1
4	-1	1	1
5	1	1	-1
6	1	-1	-1

Edge	Start Vertex	End Vertex
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8

3. Quadric surfaces

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

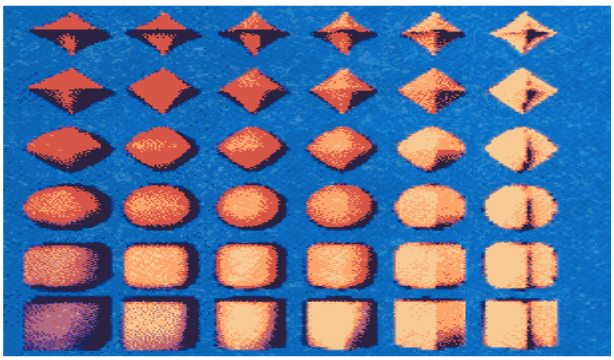


Fig: Quadric Surfaces

Quadric surfaces include:

1. Sphere: For the set of surface points (x, y, z) the spherical surface is represented as: $x^2+y^2+z^2 = r^2$, with radius r and centered at co-ordinate origin.
2. Ellipsoid: $\frac{x^2}{a^2}+\frac{y^2}{b^2}+\frac{z^2}{c^2}=1$, where (x, y, z) is the surface point and a, b, c are the radii on X, Y and Z directions respectively.
3. Elliptic parboiled: $\frac{x^2}{a^2}+\frac{y^2}{b^2}=z$
4. Hyperbolic parboiled : $\frac{x^2}{a^2}-\frac{y^2}{b^2}=z$
5. Elliptic cone : $\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=0$
6. Hyperboloid of one sheet: $\frac{x^2}{a^2}+\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$
7. Hyperboloid of two sheet: $\frac{x^2}{a^2}-\frac{y^2}{b^2}-\frac{z^2}{c^2}=1$

4. Blobby Objects

Non-rigid objects that don't maintain a fixed shape but change their surface characteristics in certain motions or when in proximity to other objects are referred to as blobby objects. Blobby objects are used to generate irregular surfaces.

E.g. molecular structures, water droplets, melting objects, muscle shaped in human body, cloth, rubber, liquids, water droplets, etc.

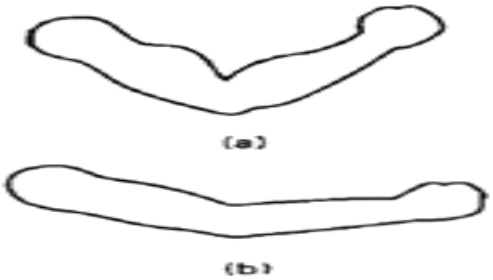


Fig: Blobby muscle shapes in a human arm

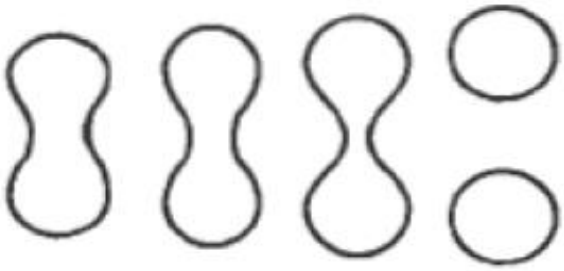


Fig: Molecular bonding (stretching and contracting into spheres)

Several models have been developed to handle these kinds of objects. One technique is to use a combination of Gaussian density functions (Gaussian bumps).

A surface function is defined as:

$$f(x, y, z) = \sum_k b_k \, e^{-a_k r_k^2} - T = 0$$

Where $r_k = \sqrt{x_k^2 + y_k^2 + z_k^2}$, T = Threshold,

a and b are used to adjust amount of blobbiness.

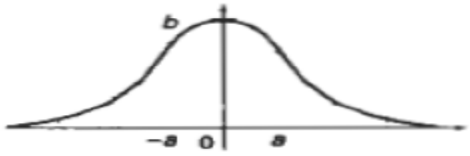


Fig: A three-dimensional Gaussian bump centered at position 0, with height band standard deviation a.

Advantages

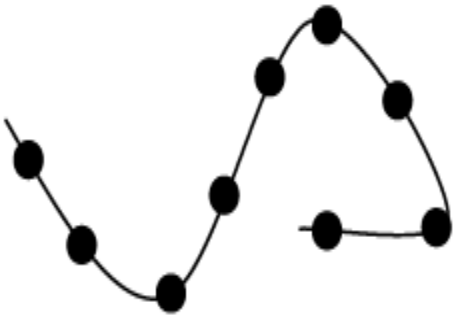
- Can represent organic, blobby or liquid line structures.
- Suitable for modeling natural phenomena like water, human body.
- Surface properties can be easily derived from mathematical equations.

Disadvantages

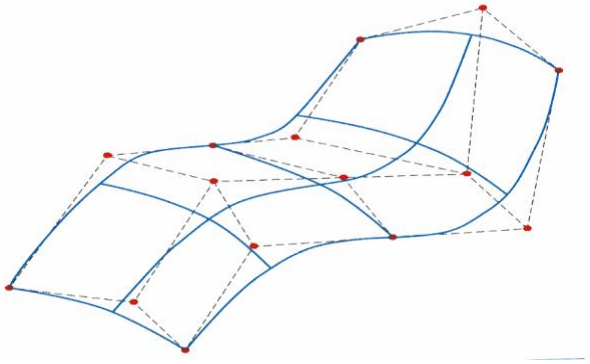
- Requires expensive computation.
- Requires special rendering engine.
- Not supported by most graphics hardware.

5. Spline Representations

Splines are the smooth curves passing through a set of given points. Mathematically, a spline curve refers to any composite curve formed with polynomial functions satisfying specified boundary conditions at the section endpoints. Spline curves are used to model 3D object surface shape smoothly. A spline curve is specified by using set of coordinate positions called **control points**. **These control points determine the general shape of the curve.** In computer-aided geometric design a control point is a member of a set of points used to determine the shape of a spline curve or, more generally, a surface or higher-dimensional object.



Fig(a): A spline curve through nine control points



Fig(b): A spline surface through 15 control points

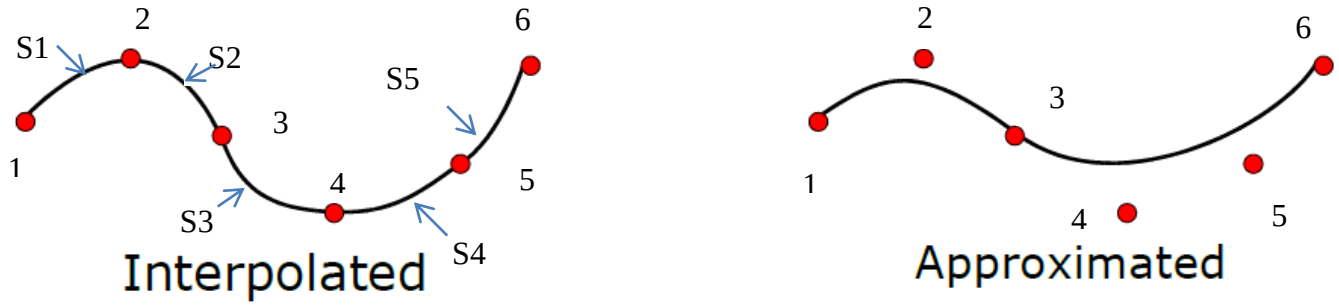
In computer graphics, a spline surface can be described with two set of orthogonal spline curves. Spline is used in graphics application to design and digitalize drawings for storage in computer and to specify animation path. Typical CAD application for spline includes the design of automobile bodies, aircraft and spacecraft surface etc. Depending upon how the curve fits the given set of data points or control points, there are two broad categories of spline curves:

1. Interpolated Splines

Given the set of control points, if a curve passes through each control point then the resulting curve is called interpolated spline. Interpolation splines are commonly used to digitize drawing, specify animation paths, and graphs.

2. Approximated Splines

If the curve is fitted from the given control points such that it follows the path of control point without necessarily passing through the set of point, then it is said to approximate the set of control point.



Five polynomials curve sections S1, S2, S3, S4, S5 interpolated between control points 1-2, 2-3, 3-4, 4-5, 5-6

5.1 Spline Specifications

There are three equivalent methods for specifying a particular spline representation:

❖ **Boundary Conditions**

- We can state the set of boundary conditions that are imposed on the spline.
- Suppose we have parametric cubic polynomial representation specified by the following set of equations:

$$x = x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \qquad 0 \le u \le 1.$$
$$y = y(u) = a_y u^3 + b_y u^2 + c_y u + d_y$$
$$z = z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

- Boundary conditions for this curve might be set, for example, on the endpoint coordinates x(0) and x(1) and on the parametric first derivatives at the endpoints x'(0) and x'(1). These four boundary conditions are sufficient to determine the values of the four coefficients a_x, b_x, c_x, and d_x.
- Similar approach can be used to determine values for y and z coordinate.

❖ **Characterizing Matrix**

- We can state the matrix that characterizes the spline.
- From the boundary condition, we can obtain the characterizing matrix for spline. Then the matrix representation for the x-coordinate can be written as;

$$x(u) = [u^3\ u^2\ u\ 1] \begin{bmatrix} a_x \\ b_x \\ c_x \\ d_x \end{bmatrix} = \boldsymbol{U} \cdot \boldsymbol{C}$$

  Where U is the row matrix of powers of parameter u, and C is the coefficient column matrix.
- Similar approach for y and z coordinate.

❖ **Blending Functions or Basis Functions**

- We can state the set of blending functions (or basis functions) that determine how specified geometric constraints (boundary conditions) on the curve are combined to calculate positions along the curve path.
- Polynomial representation for coordinate x in terms of the geometric constraint parameters:

$$x(u) = \sum_{0}^{3} x_i . B_i(u)$$

  Where x_i is the x-coordinate of the control point and $B_i(u)$ is i^{th} blending function which can be obtained by using Lagrange interpolation method.
- Similar approach for y and z coordinate, that is;

$$y(u) = \sum_{0}^{3} y_i . B_i(u)$$

$$z(u) = \sum_{0}^{3} z_i . B_i(u)$$

5.2 Cubic spline

It is most often used to set up path for object motions or to provide a representation for an existing object or drawing. To design surface of 3D object any spline curve can be represented by piece-wise cubic spline. Cubic polynomial offers a reasonable compromise between flexibility and speed of computation. Cubic spline requires less calculation with comparison to higher order polynomials and requires less memory. Compared to lower order polynomial cubic spline are more flexible for modeling arbitrary curve shape.

Given a set of control points, cubic splines can be obtained with a piecewise cubic polynomial curve (constructing a cubic polynomial curve section between each pair of control points) that passes through every control point. Suppose we have (n+1) control points specified with coordinates-

$$P_k(x_k, y_k, z_k), \qquad 0 \le k \le n.$$

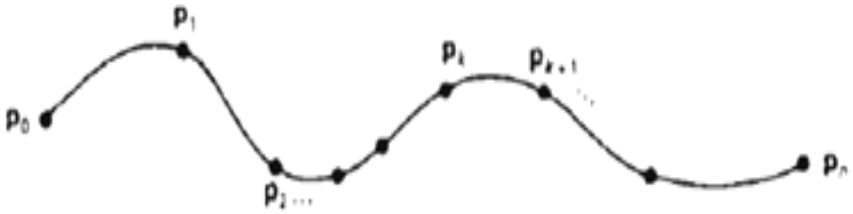


Fig: A piecewise continuous cubic-spline interpolation of n+1 control points.

We can describe the parametric cubic polynomial that is to be fitted between each pair of control points with the following set of parametric equations.as follows:

$$x = x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$
$$y = y(u) = a_y u^3 + b_y u^2 + c_y u + d_y \qquad (0 \le u \le 1)$$
$$z = z(u) = a_z u^3 + b_z u^2 + c_z u + d_z$$

For each of these three equations, we need to determine the values of the four coefficients a, b, c, and d in the polynomial representation for each of the n curve sections between the n + 1 control points. We do this by setting enough boundary conditions at the "joints" between curve sections so that we can obtain numerical values for all the coefficients.

6. Bezier Curves and Surfaces

This is spline approximation method, developed by the Engineer Pierre Bezier for use in the design of automobile body. Beziers spline has a number of properties that make them highly useful and convenient for curve and surface design. They are easy to implement. For this reason, Bezier spline is widely available in various CAD systems.

6.1 Bezier Curves

- A Bezier curve is an approximated parametric curve frequently used in computer graphics and related fields.
- A Bezier curve can be fitted to any number of control points.
- The Bezier curve can be specified with boundary condition, with characterizing matrix or blending functions. But for general Bezier curves, blending function specification is most convenient.
- A set of characteristic polynomial approximating functions, called Bezier blending functions are used to blend the control points to produce a Bezier curve segment.

- The degree of a Bezier curve segment is determined by the number of control points to be fitted with that curve segment.
- **Mathematical Definition:**
  - Given a set of (n+1) control points, P_0(x0, y0, z0), P_1(x1, y1, z1), P_2(x2, y2, z2),…, P_n(xn, yn, zn), a parametric Bezier curve(Bernstein-Bezier curve) segment can be defined by

$$C(u) = \sum_{i=0}^{n} P_i . BEZ_{i,n}(u) \qquad \ldots(1)$$

Here;

u is a parameter such that $0 \leq u \leq 1$.

$BEZ_{i,n}(u)$ is a Bezier Blending function or Bernstein Polynomial and is defined by-

$$BEZ_{i,n}(u) = C(n,\ i)u^i(1-u)^{n-i} \qquad \ldots(2)$$

where $C(n,i)$ are the binomial coefficients given by;

$$C(n,i) = \frac{n!}{(n-i)!i!} \qquad \ldots(3)$$

Equation (1) represents a set of three parametric equations, so for individual curve coordinates;

$$x(u) = \sum_{i=0}^{n} x_i . BEZ_{i,n}(u)$$

$$y(u) = \sum_{i=0}^{n} y_i . BEZ_{i,n}(u)$$

$$z(u) = \sum_{i=0}^{n} z_i . BEZ_{i,n}(u)$$

As a rule, a Bezier curve is a polynomial of degree one less than the number of control points used: Three points generate a parabola; four points a cubic curve, and so forth.

- Bezier curves are widely used in computer graphics to model smooth curves. As the curve is completely contained in the convex hull of its control points, the points can be graphically displayed and used to manipulate the curve.

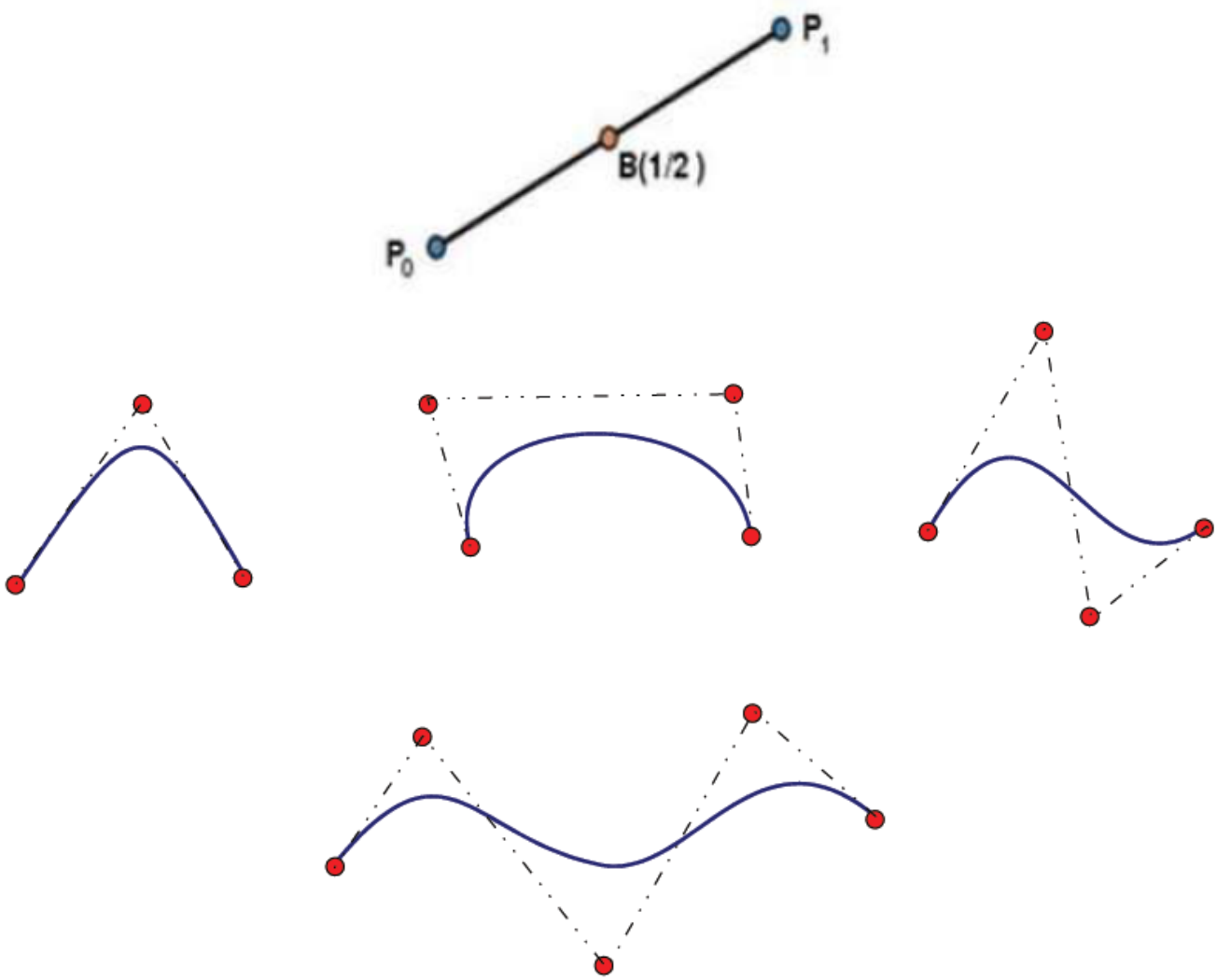
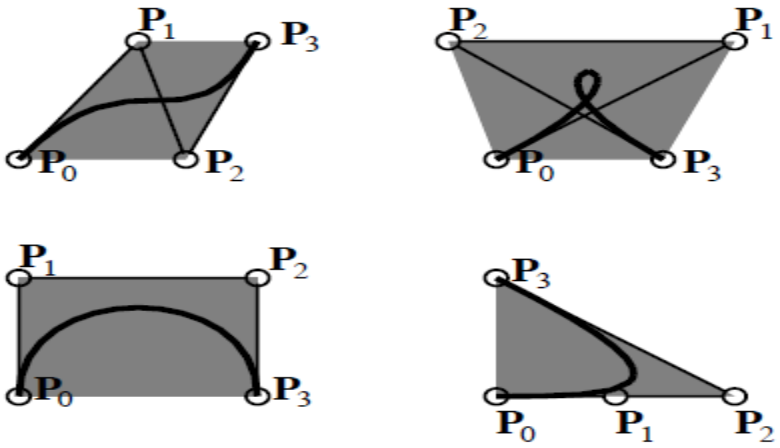


Fig: Examples of two-dimensional Bezier curves generated from two, three, four, and five control points. Dashed lines connect the control-point positions.

6.2 Properties of Bezier Curves

- The Bezier curve always passes through the first and last control points.
  - That is $C(u=0)=P_0$ and $C(u=1)=P_n$, since u=0(for first point) and u=1(for last control point).

- The degree of polynomial defining the curve segment is one less than the number of defining polygon points. Therefore, for 4 control points the degree of polynomial is three, for three control points the degree of polynomial is 2 and so on.
- The curve generally follows the shape of the defining polygon.
- The curve is always contained within the convex hull (The convex polygon boundary that encloses a set of control points is called the convex hall) of the control points.



- The curve can be translated and rotated by applying transformation on the control points.
- Closed curves can be generated by making the last control point the same as the first control point.
- The sum of all Bezier blending functions, if the curve lies within the convex hull, is always 1 and are always positive i.e.,

$$\sum_{i=0}^{n} B_{i,n}(u) = 1$$

Exercise:

1. Model the Bezier curve whose control points are P0(4, 2), P1(8,8), and P2(16, 4) at u={0.5,0.7,0.9}.
2. 2. Construct the Bezier Curve of order 3 and with 4 polygon vertices A(1,1), B(2,3), C(4,3), and D(6,4) at u={0,1/4,1/2,3/4}.

6.3 Bezier Surface

A Bezier Surface is formed as the Cartesian product of the blending functions of two orthogonal Bezier curves. Two sets of orthogonal Bezier curves can be used to design an object surface by specifying an input mesh of control points. That is,

$$P(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j}.BEZ_{i,m}(u).BEZ_{j,n}\ (v)$$

Where, $P_{i,j}$ specify the location of the (m + 1) by (n + 1) control points.

The control points are connected by dashed lines, and the solid lines show curves of constant u and constant v. Each curve of constant u is plotted by varying v over the interval from 0 to 1, with u fixed at one of the values in this unit interval. Curves of constant v are plotted similarly.

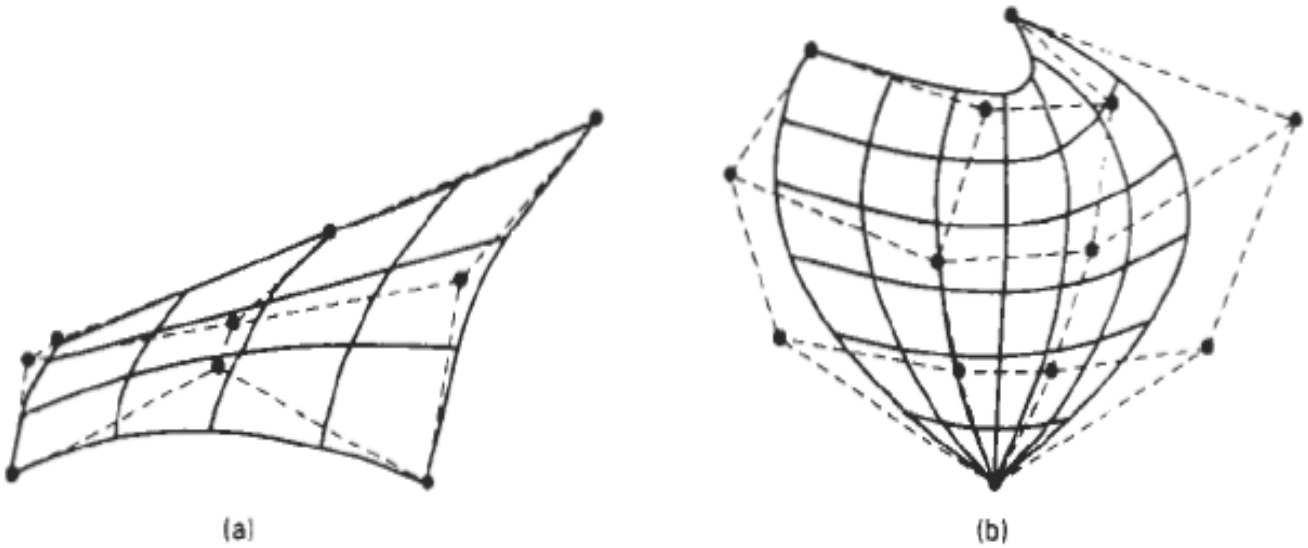


Fig: Bezier surfaces constructed tor (a) m = 3, n = 3, and (b) m = 4, n = 4.
Dashed lines connect the control points.

6.3.2 Space Partitioning Method

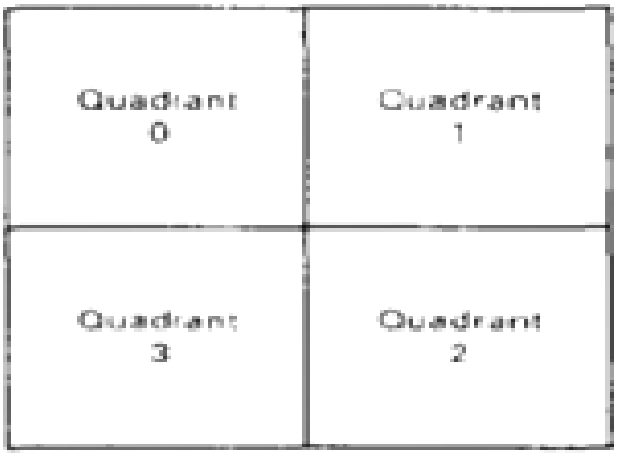
Quadtree and Octree Representation

Quadtree and Octree are the most widely used space partitioning methods for 3D solid object representation.

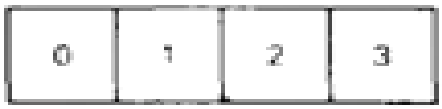
Quadtree:

This is the space-partitioning method for 3D solid object representation. This is hierarchical tree structures (quadtree) used to represent solid object in some graphical system. Each node corresponds to a region of 3D space.

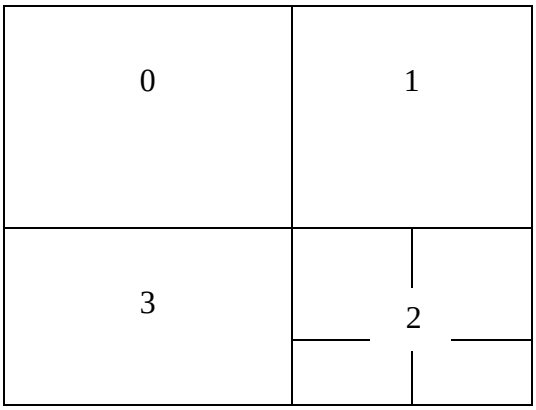
Quadtrees are generated by successively dividing a two-dimensional region (usually a square) into quadrants. Each node in the quadtree has four data elements, one for each of the quadrants in the region. If all pixels within a quadrant have the same color (a homogeneous quadrant) the corresponding data element in the node stores that color. In addition, a flag is set in the data element to indicate that the quadrant is homogeneous. Otherwise, the quadrant is said to be heterogeneous, and that quadrant is itself divided into quadrants as shown in figure.



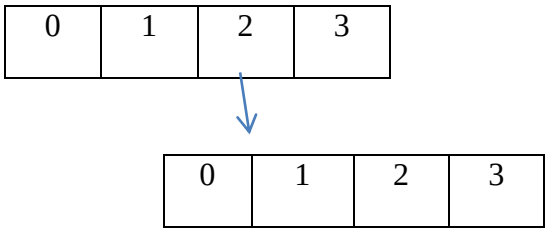
Region of a Two Dimensional Space



Data Elements in the Representative Quadtree Node



Region of 2D space



Quadtree Representation

Medical imaging and other applications that require displays of object cross section commonly use this method. E.g. CT-scan. Quadtree provides some memory savings.

Octree Representation:

This is hierarchical tree structures (octree) used to represent solid object in some graphical system. Each node corresponds to a region of 3D space.

The octree encoding procedure for a three-dimensional space is an extension of an encoding scheme for two-dimensional space, called **quadtree encoding**. It provides a convenient representation for storing information about object interiors. An octree encoding scheme divides region of 3D space into octants and stores 8 data elements in each node of the tree. Individual elements are called volume element or voxels. When all voxels in an octant are of same type, this type value is stored in corresponding data elements. Any heterogeneous octants are subdivided into octants again and the corresponding data element in the node points to the next node in the octree. Procedures for generating octrees are similar to those for quadtrees: Voxels in each octant are tested, and octant subdivisions continue until the region of space contains only homogeneous octants. Each node in the octree can now have from zero to eight immediate descendants.

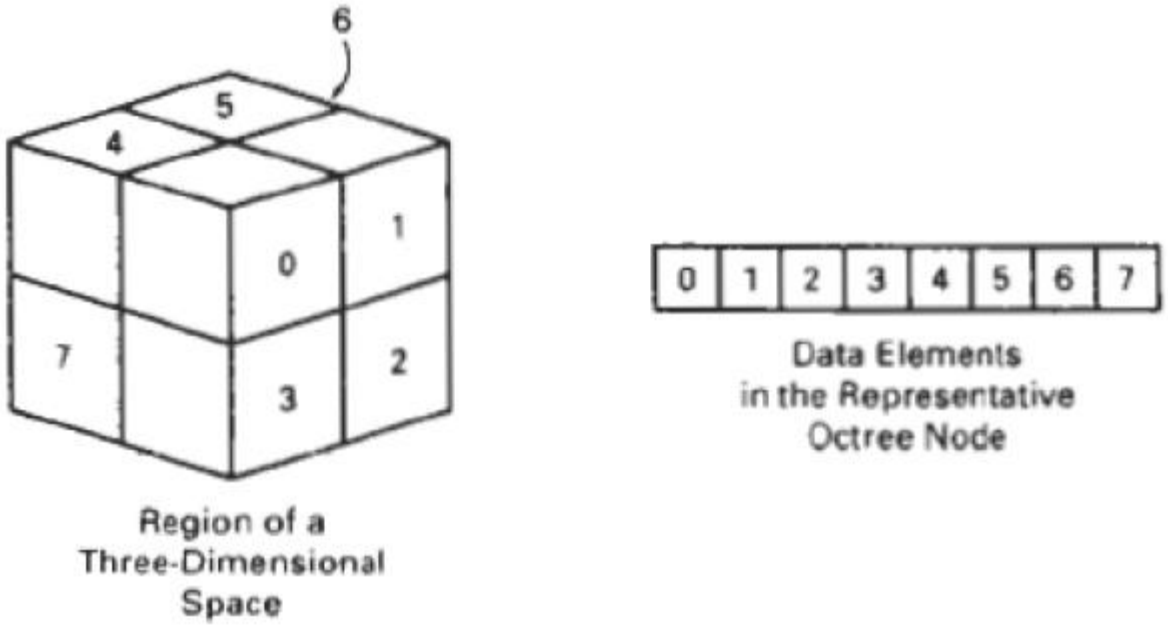


Fig: Region of a three-dimensional space divided into numbered octants and the associated octree node with eight data elements

6.4 Solid Modeling

- **Solid modeling** (or **modelling**) is a consistent set of principles for mathematical and computer modeling of three-dimensional solids.
- A mathematical technique for representing solid objects.
- An object with following specification is called solid
  - **Vertices (nodes), edges, surfaces, weight, and volume.**
- **Methods for solid modeling:**
  - **Boundary Representation**
  - **Octrees/Quadtrees**
  - **Sweep Representation**
  - **Constructive solid Geometry**

6.4.1 Constructive Solid Geometry (CSG)

A CSG model is based on the topological notion that a physical object can be divided into a set of primitives (basic elements or shapes) that can be combined in a certain order following a set of rules (Boolean operations) to form the object. Each primitive is bounded by a set of surfaces; usually closed and orientable.

- It is based on the idea of providing a set of predefined object types such as cubes, cone, sphere etc.
- A solid model is created by retrieving primitive solids and performing Boolean operations.
- Three types of Boolean operations:
  - **Union (join):** the operation combines two volumes included in the different solids into a single solid.
  - **Subtract (cut):** the operation subtracts the volume of one solid from the other solid object.

- **Intersection**: the operation keeps only the volume common to both solids.

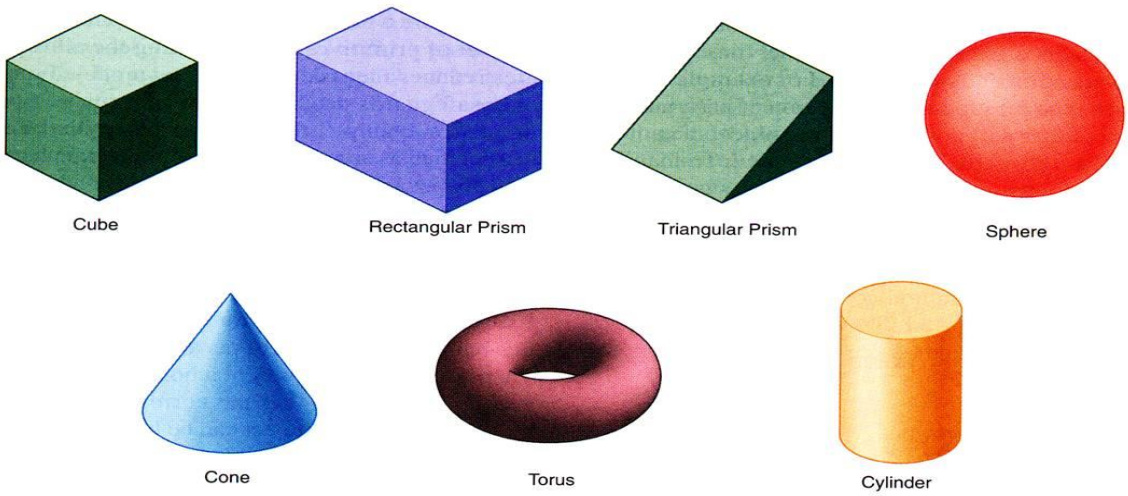
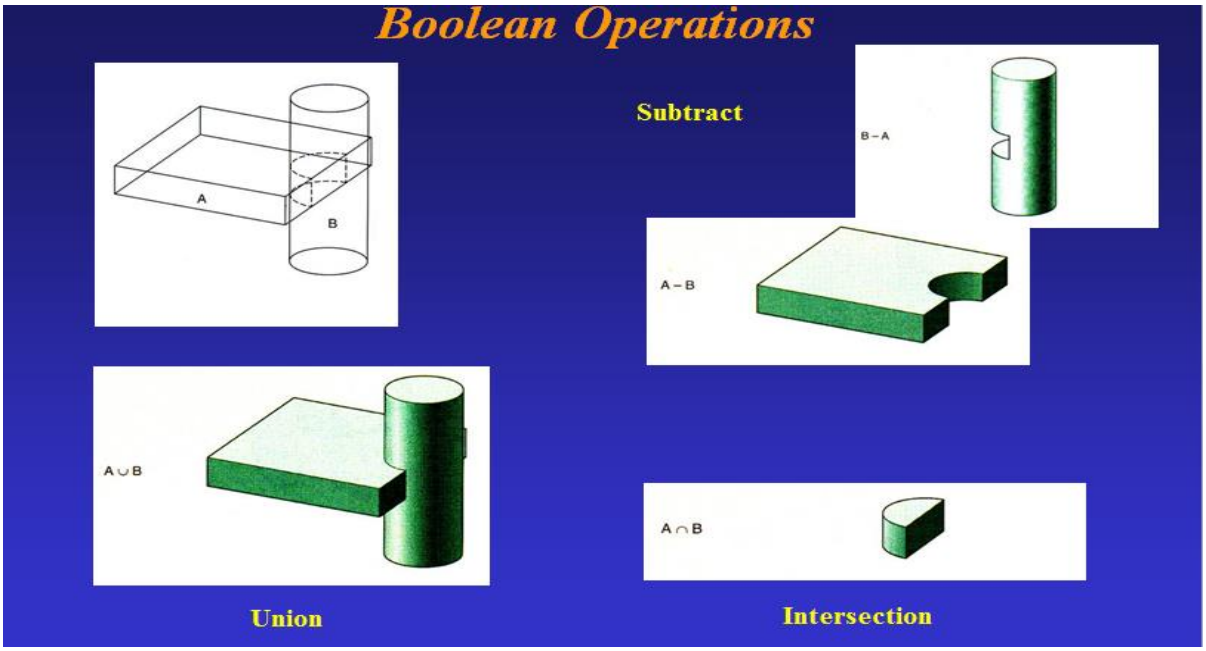


Fig: the basic primitive solid



6.4.2 Sweep Representation

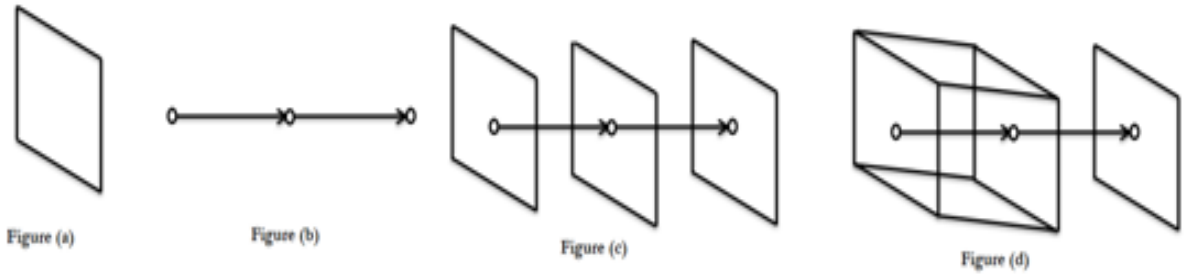
- Sweep representations create solid by moving a 2D shape (triangle, rectangle, polygon etc.) according to predefined rule (translating, rotating).
- Sweep representations are used to construct 3D object from 2D shape that have some kind of symmetry.
  - For example, a prism can be generated using a translational sweep and rotational sweeps can be used to create curved surfaces like an ellipsoid or a torus.
- there are two types of sweep representation:
  - Translational Sweep
  - Rotational Sweep

Translational sweep

- A 2D shape is translated by a predefined translation vector.
  - Example: following figure shows a translational sweep of a rectangle.

Steps

- Define a shape as a polygon vertex table as shown in figure (a).
- Define a sweep path as a sequence of translation vectors figure (b).
- Translate the shape; continue building a vertex table figure (c).
- Define a surface table figure (d).

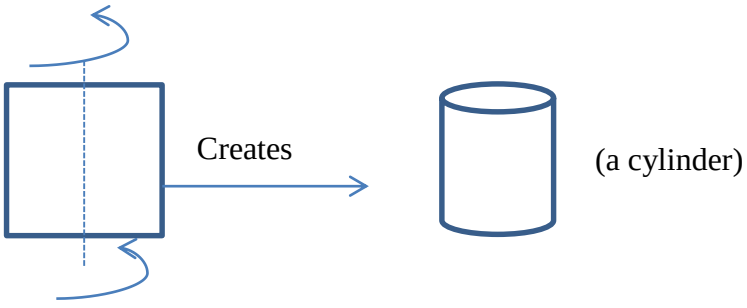


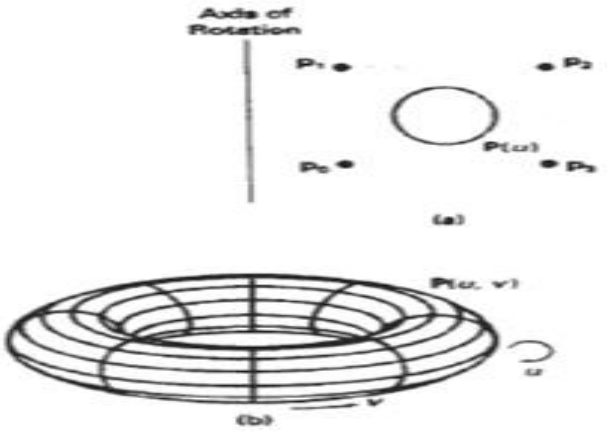
Another Example:

Constructing a solid with a translational sweep. Translating the control points of the periodic spline curve in (a) generates the solid shown in (b), whose surface can be described with point function $P(u,v)$.

Rotational sweep

- A 2D shape is rotated around a predefined rotational axis.
  - Example: following figure shows a rotational sweep.





Constructing a solid with a rotational sweep. Rotating the control points of the periodic spline curve in (a) about the given rotation axis generates the solid shown in (b), whose surface can be described with point function P(u,v).

Steps:

- Define a shape as a polygon vertex table as shown in figure (a).
- Define a sweep path as a sequence of rotations.
- Rotate the shape; continue building a vertex table as shown in figure (b).
- Define a surface table as shown in figure (c).

Figure (a)

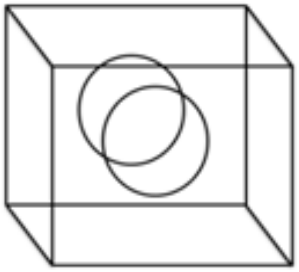


Figure (b)

Figure (c)

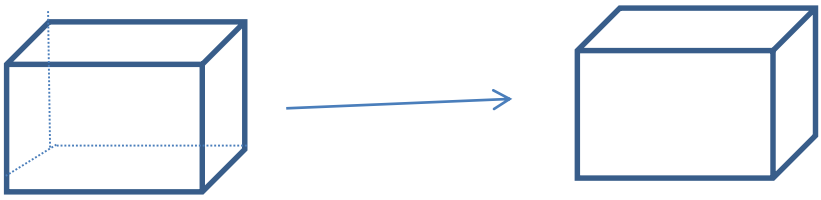
Chapter 7

Visible Surface Detection Methods

7.1 Introduction

When we view a picture containing non-transparent objects and surfaces, then we cannot see those objects from view which are 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.
- *Visible Surface Detection Method/Hidden Surface Elimination* is the process of identifying those parts of a scene that are visible from a chosen viewing position.
- Alternatively, it can be defined as a procedure that removes any hidden surfaces or lines which are not to be displayed in a 3D scene is called as hidden surface/line elimination or visible surface detection algorithm.
- 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.
- Visible surface detection or Hidden surface removal is major concern for realistic graphics displaying. Displaying only visible lines and surfaces in the rendering of a picture enhances clarity, efficiency, and impact.



A cube Hidden Surfaces Removed

- Visible surface detection process can be boosted by applying the **coherence** properties (similarity in faces, depths, scan-lines, object, edges etc).
- Several algorithms have been devised for efficient identification of visible objects.
- Visible surface detection methods are broadly classified into two categories, according to whether they deal with objects directly or with their projected images.

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.

Object Space Methods

- An object-space method compares objects and parts of objects to each other within the scene definition to determine which surfaces are visible.
- **It involves comparing the polygons in the object to other polygon in the object and to polygons in every other object**.
- This method is implemented on world coordinate system.
- Simple but inefficient (requires longer processing time).
- ***BASIC PROCEDURE:***

  For each object in the scene do

  Begin

  - Determine those parts of the object whose view is unobstructed by other parts of it or any other object with respect to the viewing specification.
  - Draw those parts in the object color.

  End.

Image Space Methods

- Visibility is determined point by point at each pixel position on the projection plane.
- These methods are implemented on screen/device coordinate system.
- ***BASIC PROCEDURE:***

  For each pixel (x, y) in the frame buffer

  - Determine the polygon/object/part of polygon closest to that pixel by taking orthographic projection.
  - Draw/color the pixel in the object color.
- Faster method but requires larger memory.

❖ **What's the difference between Image Space and Object Space**

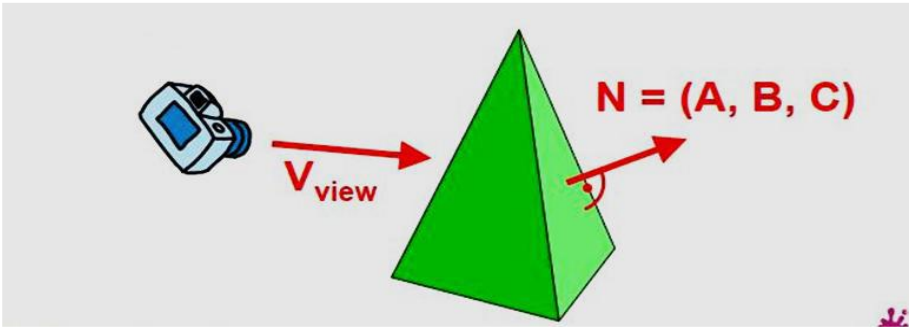
- Image space algorithms are much more efficient than object space algorithms
- Object space algorithms are much more functional than image space algorithms
- Color calculation in object space algorithms is done only one time and is retained by it but in image space algorithm the calculation once done is over written later.

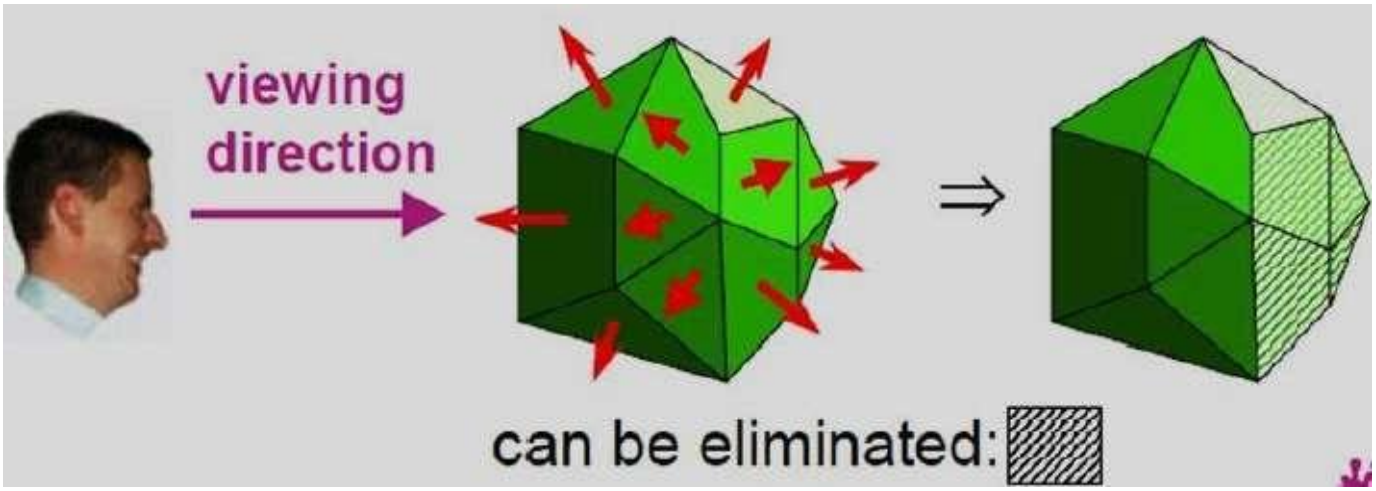
7.2 Different Visible Surface Detection Algorithms

7.2.1 Back-Face Detection (Plane equation Method)

- A fast and simple object-space method for identifying the back faces of an object.
- In this method no faces on the back of the object are displayed.
- In general, about half of the faces of objects are back faces hence this algorithm will remove about half of the total polygon faces in the mage.

- ***Basic Procedure:***
  - Let N be a normal vector on a polygon surface, which has Cartesian components *(A, B, C).* And let V_{view} is a vector in the viewing direction from the eye (or "camera") position.
  - Then this polygon surface is a back face if $\mathbf{V_{view}.N > 0.}$





Pros and Cons:

- It is simple and easy implement.
- No pre-sorting of polygon surfaces is needed.
- Cannot address partial visibility.
- Not suitable for complex scenes.

7.2.2 Depth Buffer Method (Z-Buffer Method)

- A commonly used image-space approach for detecting visible surfaces.
- Also called **z-buffer** method since depth usually measured along z-axis.
- This approach compares surface depths at each pixel position on the projection plane. The depth values for a pixel are compared and the closest surface determines the color to be displayed in the frame buffer.
- Each surface of a scene is processed separately, one point at a time across the surface. And each (x, y, z) position on a polygon surface corresponds to the orthographic projection point (x, y) on the view plane.
- 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.
- The **frame buffer** is used to store the intensity value at each position (x, y).
- ***This method requires two buffers:***
  - A **z-buffer** or **depth buffer:** Stores depth values i.e., z-values for each pixel position (x, y).
  - **Frame buffer (Refresh buffer):** Stores the surface-intensity values or color values for each pixel position (x, y).
- As surfaces are processed, the image buffer is used to store the color values of each pixel position and the z-buffer is used to store the depth values for each (x, y) position.

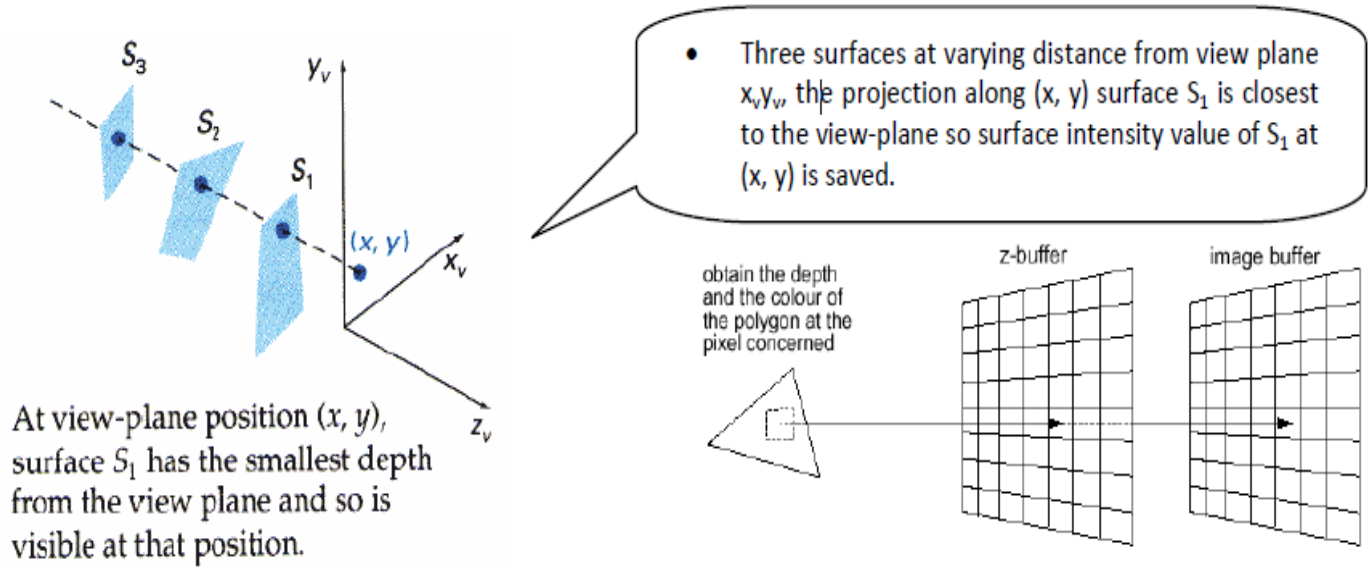
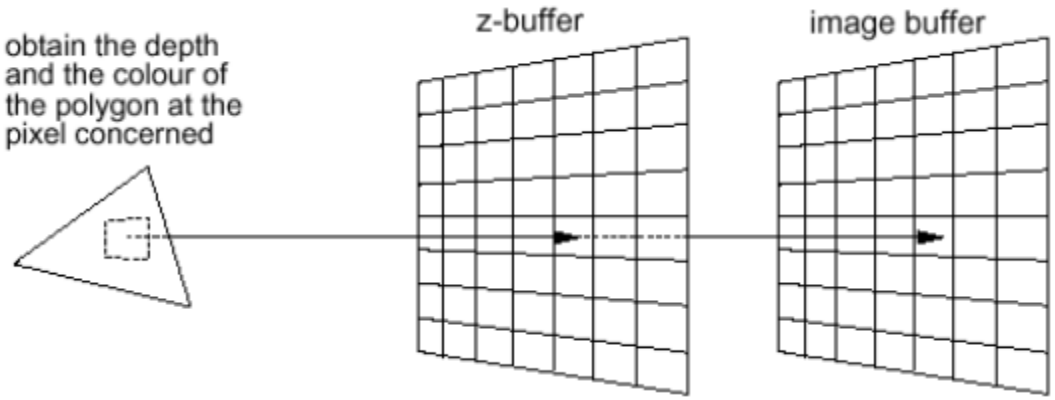


Fig: At view-plane position *(x, y)*, surface S_1, has the smallest depth from the view plane and so is visible at that position.



Algorithm:

Step-1: For all buffer positions (x, y), initialize the buffer values;

- Depth_buffer (x, y) =∞ or maximum z-coordinate.
- Frame_buffer (x, y) = background color.

Step-2: Process each polygon surface P (One at a time)

- For each pixel (x, y), calculate depth(x, y) = d.
  - If d < depth_buffer (x, y) then
    - Compute surface color at position (x, y) i.e., $I_P(x, y)$.
    - Set Depth_buffer (x, y) = d,
    - Frame_buffer (x, y) = $I_P(x, y)$, where $I_P(x, y)$ is the intensity value for the surface at pixel position (x, y).

After all polygon surfaces have been processed, the depth buffer contains depth values for the visible surfaces and each pixel of the frame buffer represents the color of a visible surface at that pixel.

Depth Calculation:

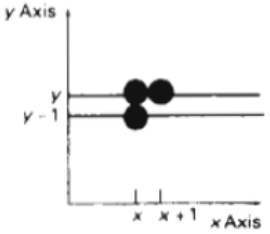
The depth values of the surface position (x, y) are calculated by plane equation of surface.

$$Z = \frac{-Ax - By - D}{C}$$

Let Depth Z' at position (x+1, y)

$$Z' = \frac{-A(x+1) - By - D}{C}$$

$$\Rightarrow Z' = Z - A/C \qquad (1)$$



$-A/C$ is constant for each surface so succeeding depth value across a scan line are obtained form preceding values by simple calculation.

Pros and Cons:

- ❖ It is simple and easy implement, no specific hardware is needed.

- No pre-sorting of polygons is needed.
- No object-object comparison is required. Can be applied to non-polygonal objects.
- Good for animation.
- Additional memory buffer (z-buffer) is required.
- It only deals with opaque surfaces.

7.2.3 A-buffer Method (Accumulation buffer Method)

- A-buffer method is an extension of depth-buffer method.
- The A-buffer method represents an area-averaged, accumulation-buffer method.
- A drawback of the depth-buffer method is that it can only find one visible surface at each pixel position.
- In other words, it deals only with opaque surfaces and cannot accumulate intensity values for more than one surface, as is necessary if transparent or translucent surfaces are to be displayed.

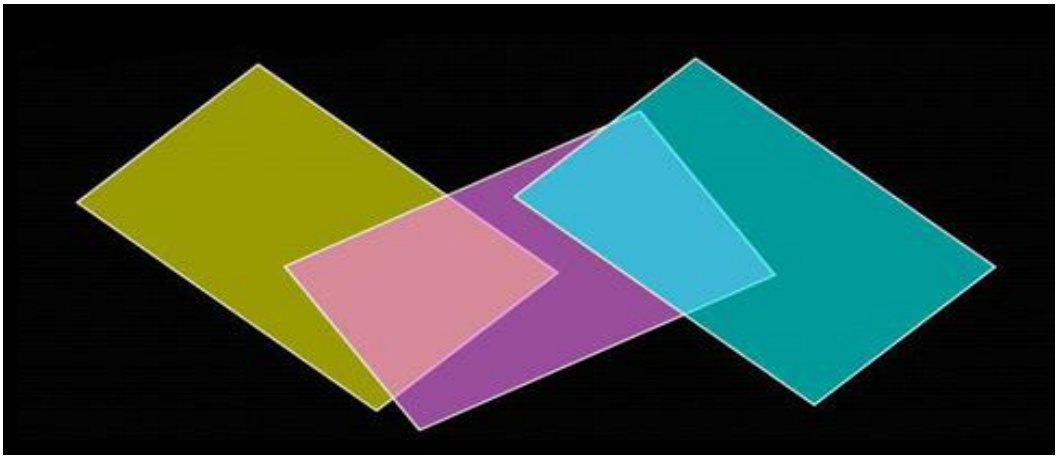


Fig: Viewing two background opaque surfaces through a foreground transparent surface

- The A-buffer method expands the depth buffer so that each position in the buffer can reference a linked list of surfaces.
- It maintains a data structure of background surfaces that are behind the foreground transparent surface. This special data structure is called *accumulation buffer*.
- Each position in the A-buffer has two fields: **Depth field** and **Intensity Field or Surface Data Field**.
  - **Depth field:** Stores a positive or negative depth value.
  - **Intensity Field:** stores surface-intensity information or a pointer value. It includes:
    - RGB intensity components
    - Opacity Parameter
    - Depth
    - Percent of area coverage
    - Surface identifier

- If depth value positive, then that surface is opaque and intensity field stores surface intensity at that position. If depth >= 0, It indicates that there is a single surface overlapping the corresponding pixel area.
- If it is negative, this indicates multiple-surface contribution to the pixel intensity. The intensity field stores pointer to a linked list of background surfaces.

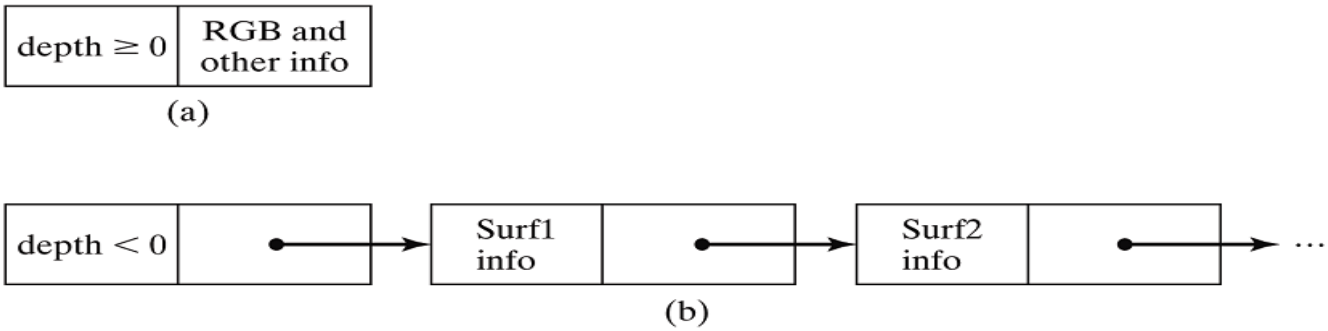


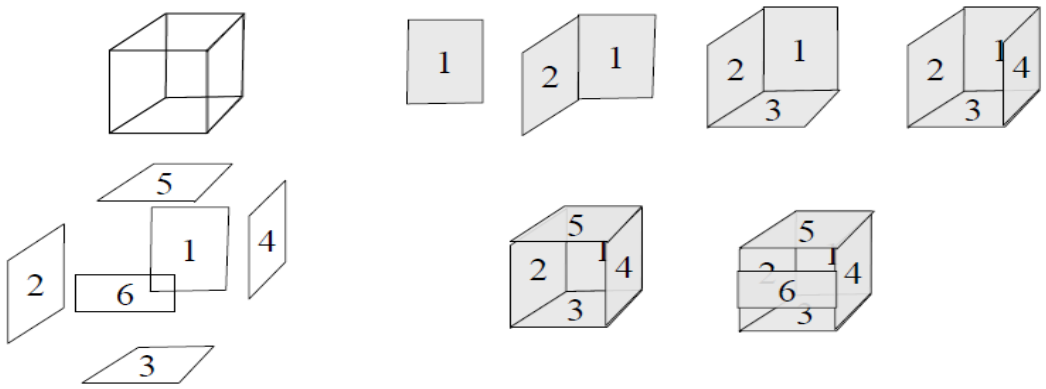
Fig: Organization of an A-buffer pixel position: (a) single-surface (b) multiple surfaces overlap

7.2.4 Depth sorting Algorithm or Painter's Algorithm

- Another widely used object space method.
- This method for solving the hidden-surface problem is often referred to as the ***Painter's algorithm or list priority algorithm.***
- This algorithm is also called "Painter's Algorithm" as it simulates how a painter typically produces his/her painting by starting with the background and then progressively adding new (nearer) objects to the canvas.
- This method requires sorting operation of surfaces in both the image and object space.
- **Basic Procedure:**
  - Sort all surfaces of polygon in order of decreasing depth.
  - The intensity values for farthest surface are then entered into the refresh buffer. That is farthest polygon is displayed first, then the second farthest polygon, so on, and finally, the closest polygon surface.
  - After all surfaces have been processed, the refresh buffer stores the intensity values for all visible surfaces.

- When there are only a few objects in the scene, this method can be very fast. However, as the number of objects increases, the sorting process can become very complex and time consuming.

First draw the
distant objects than the closer objects. Pixels of each object overwrites the previous objects.



7.2.5 Scan Line Algorithm

- Image-space method for identifying visible surfaces. It deals with multiple polygon surfaces.
- The scan line method solves the hidden surface problem one scan line at a time. Processing of the scan line start from the top to the bottom of the display.
- This method calculates the depth values for only the overlapping surfaces which are tested by the scan line.
- Each scan line is processed; all polygon surfaces intersecting that line are examined to determine which are visible.
- Across each scan line, depth calculations are made for each overlapping surface to determine which surface is nearest to the view plane.
- When the visible surface has been determined, the intensity value for that position is entered into the refresh buffer.
- It requires an edge table, polygon table, active edge list and flag.
- **Edge table contains:** Coordinate end points for each scan line, pointers into the polygon table to identify the surfaces bounded by each line.
- **Polygon table contains**: Coefficients of plane equations for each surfaces, pointers into the edge table, and intensity information of the surfaces.
- **Active edge list contains**: Edges that cross the current scan line, shorted in order of increasing x.
- **Flag** is defined for each surface that is set ‘ON’ or ‘OFF’ to indicate whether a scan line is inside or outside of the surface. At the left most boundary surface flag is ‘ON’ and at right most boundary flag is ‘OFF’.

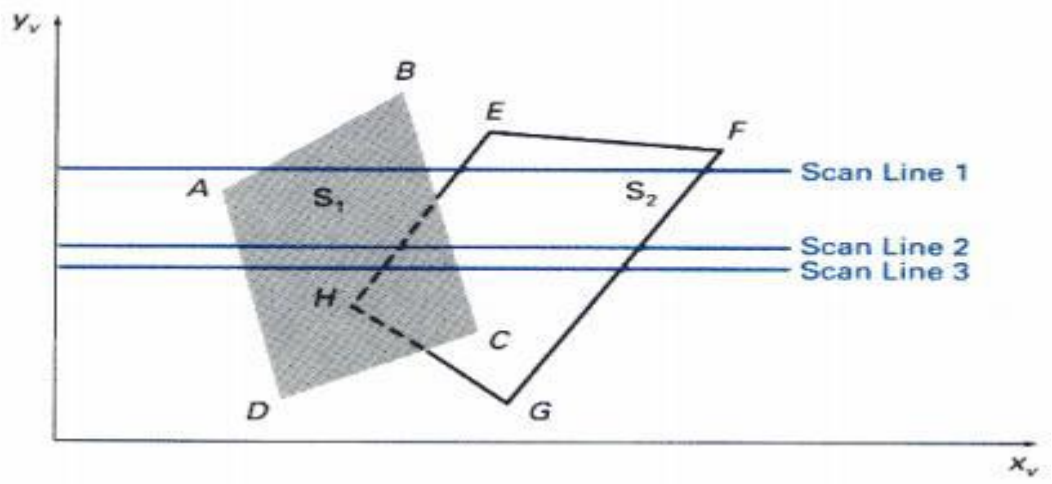


Fig: ***Scan lines crossing the projection of two surfaces, S1, and S2 in the view plane. Dashed lines indicate the boundaries of hidden surfaces.***

Algorithm:

1) Establish and initialize data structure.
    i) Edge table with line endpoints.
    ii) Polygon table with surface information and pointer to the edge table.
    iii) Initially empty active edge list.
        i.e., AEL = { }.
    iv) A flag, initially flag is "off" for each surface.
2) Repeat for each scan line.
    a) Update active edge list AEL.
        For each pixel (x, y) on scan line
        (1) Update flag for each surface.
        (2) When flag is ON for only one surface, enter intensity of that surface to refresh buffer.
        (3) When two or more flags are ON, calculate depth and store the intensity of the surface which is nearest to the view plane.

Working Procedure:

Above figure illustrate the scan line method for locating visible portion of surfaces for pixel position along the line. The active list for scan line one contains information form the edge table for edges AB, BC, EH and FG.

For scan line 1, between edges AB and BC, flag (S1) = ON and flag (S2) = OFF. Therefore no depth calculations are necessary and intensity information for surface S1 is entered from the polygon table into the refresh buffer. Similarly, between edge EH and FG flag (S1) = OFF and flag (S2) = ON.

For scan line 2 and 3, the active edge list contains edges AD, EH, BC and FG. Along the scan line 2, from edge AD to edge EH, flag (S1) =ON and flag (S2) = OFF.

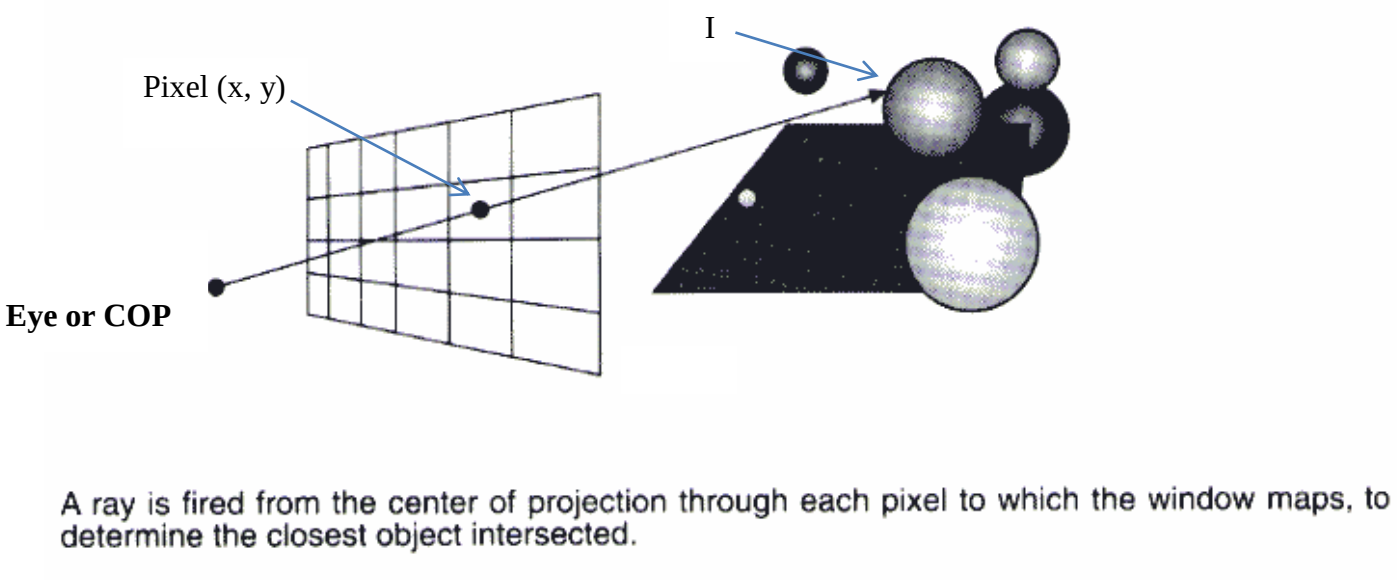
Between edges EH and BC, Flag (S1) = ON and flag (S2) = ON. Therefore depth calculation must be made using the plane coefficient for the two surfaces. For this example the depth of surface S1 is assumed to be less than that of S2, So intensity for surface S1 are loaded into the refresh buffer until boundary BC is encountered. Then the flag for the surface S1 goes off. And intensity for surface S2 is stored until edge FG is passed.

Pros and Cons:

- Any number of polygon surfaces can be processed with this method. Depth calculations are performed only when there are overlapping polygons.
- Deals with transparent, translucent, and opaque surfaces.
- Can be applied to non-polygonal objects.
- Complex.

7.2.6 Ray Tracing Algorithm

- Ray tracing also known as ray casting is efficient image space method for visibility detection in the objects.
- In Ray-tracing algorithm the basic idea is to trace light rays from the viewpoint through pixels until it reaches a surface. Since each ray can intersect several objects, find the intersection point which is closest to the viewpoint. And set the pixel to the color of the surface at the point where the light ray strikes it.
- Ray-tracing algorithm provides the flexibility to handle both flat and curved surfaces. The algorithm is based on the principles of light and optics.
- If resolution of screen is (x_1 * y_1) then there are x_1y_1 pixels and so x_1y_1 light rays are traced.



A ray is fired from the center of projection through each pixel to which the window maps, to determine the closest object intersected.

- **Algorithm:**

  For each pixel(x, y)

  - Trace a ray from eye point or viewpoint through a pixel (x, y) into scene.
  - Find the intersection point 'I' with the surface, which is closest to the viewpoint.
  - Set pixel color to the color of surface on which this point 'I' lies.

❖ **Pros and Cons:**

- Suitable for complex curved surfaces.
- Computationally expensive.
- Can be easily combined with lightning algorithms to generate shadow and reflection.

7.2.7 Binary Space Partition Tree Method (BSP Tree Method)

❖ A binary space partitioning (BSP) tree is an efficient method for determining object visibility by painting surfaces onto the screen from back to front as in the painter's algorithm.

❖ A BSP tree is a recursive sub-division of space that treats each line segment (or polygon, in 3D) as a cutting plane which is used to categorize all remaining objects in the space as either being in "front" or in "back" of that plane.

❖ The BSP tree is particularly useful when the view reference point changes, but object in a scene are at fixed position (static group of 3D objects).

❖ Basic Idea:

- *Sort the polygons for display in back-to-front order.*

BSP Algorithm:

❖ Construct a BSP Tree.

- Select the partitioning plane (one of the polygon in the picture).
- Partition the space into two sets of objects; surfaces that are inside/front and outside/back w.r.t. the partitioning plane.
- In case of intersection, i.e., if object is intersected by partitioning plane then divide object into two objects.
- Identify surfaces in each of the half spaces.
- Each half space is then recursively subdivided using one of the polygon in the half space as partitioning plane. This process is continues till there is only one polygon in each half space.
- Then the subdivided space is represented by a binary tree with the original polygon as the root.
- Objects are represented as terminal nodes with **front objects as left branches and back objects as right branches.**

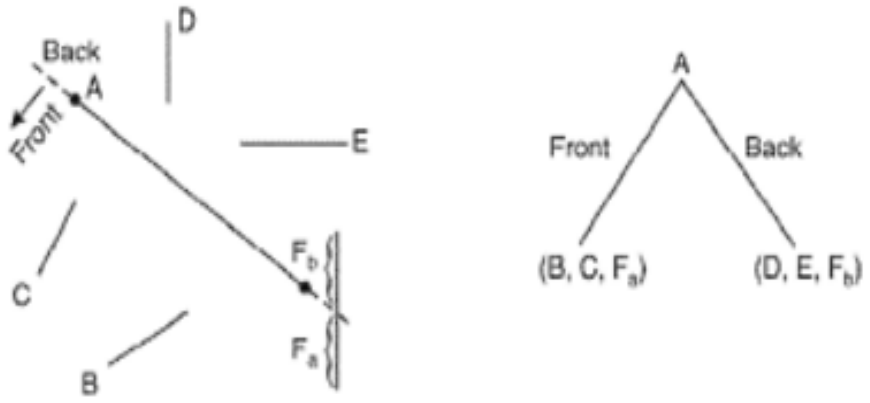
❖ **Display the BSP tree.**

- If the view point is in front of the root polygon, then BSP tree is traversed in the order of back branch, root and then front branch i.e., reverse of inorder traversal.
- If the view point is in back of the root polygon, then BSP tree is traversed in the order of front branch, root and then back branch i.e., normal inorder traversal.

Example:

1. Construction of BSP tree

Consider polygon A as a partitioning plane. Then on front side of A, the polygons will be B, C whereas on back side of A, the polygons will be d and E.



But polygon F is getting intersected by plane A into two polygon F_a and F_b. Then each of this subdivided polygon is placed in appropriate half space. **Please note that here we are considering polygon A as dividing plane, so the root of BSP tree will be A only**. The BSP tree at this level will be as shown in Fig.

Each half space is then recursively subdivided using one of the polygon in the half space as separating plane. This process of space subdivision continues till there is only one polygon in each half space, because this is a binary tree. Then the subdivided space is represented by a binary tree with the original polygon as the root.

Let us proceed with our example, we have divided the space into two parts, front and back. Let's concentrate on front part only. In front part we are having B, C and F_a polygons. So now we have to select one polygon out of these as a dividing plane. Let us select polygon B see the Fig.

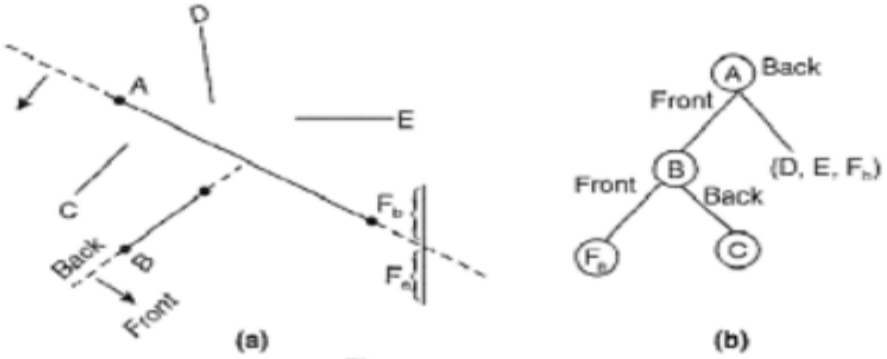
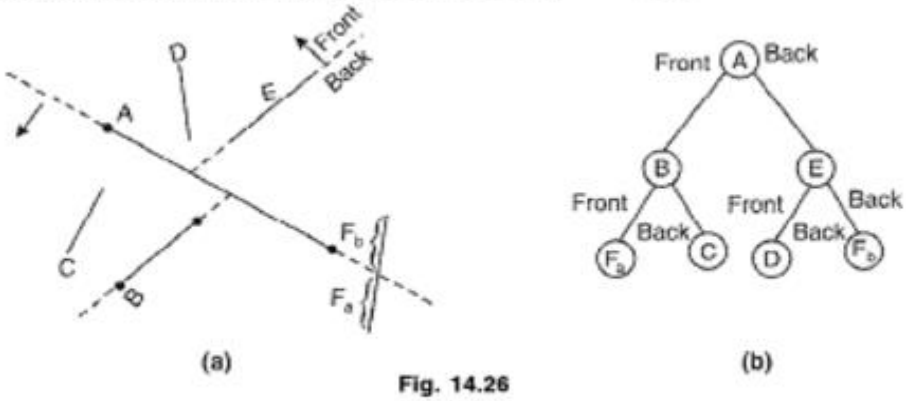


Fig.

After selecting polygon B as a dividing plane, it places polygons C and F_a in back and front parts respectively. The BSP tree is also shown in Fig. (b). Now further we cannot divide F_a

or C polygons, as there is nothing to divide. So, we have to select another path, i.e. Back path of root, which is having D, E and F_b. Again divide this into two parts by selecting one of the polygon as a dividing plane. Let us select polygon E. See the Fig. (a).



(a) (b)

Fig. 14.26

After selecting polygon E as a dividing plane, it places polygons D and F_b in front and back parts respectively. Now the BSP tree will become same as shown in Fig. (b). Further we cannot divide. So this is a final BSP tree.

2. Display a BSP tree

If the view point is in front of the root polygon, then BSP tree is traversed in the order of back branch, root, front branch i.e. reverse of inorder traversal.

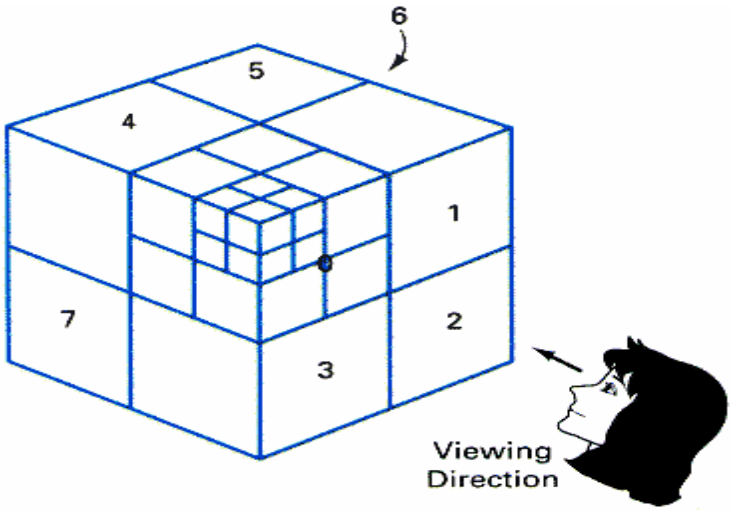
For our example, if the viewpoint is in front of the root polygon then the sequence of polygons for display will be F_b, E, D, A, C, B, F_a.

If the view point is in back of the root polygon then BSP tree is traversed in the order of front branch, root, back branch i.e, normal inorder traversal.

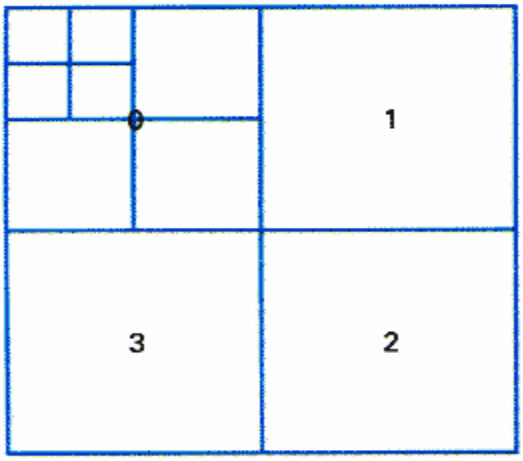
For our example, if the view point is in back of the root polygon, as shown then the sequence of polygons for display will be F_a, B, C, A, D, E, F_b.

7.2.8 Octree Method

- In octree method, octree nodes are projected onto the viewing surface in a front-to-back order as shown in figure below.



Octants in Space



Quadrants for the View Plane

- Any surfaces toward the rear of the front octants (0, 1, 2, 3) or in the back octants (4, 5, 6, 7) may be hidden by the front surfaces.
- With the numbering method (0, 1, 2, 3, 4, 5, 6, 7), nodes representing octants 0, 1, 2, 3 for the entire region are visited before the nodes representing octants 4, 5, 6, 7.
- Similarly the nodes for the front four suboctants of octant 0 are visited before the nodes for the four back sub octants.
- When a color is encountered in an octree node, the corresponding pixel in the frame buffer is painted only if no previous color has been loaded into the same pixel position.
- **Algorithm:**
  - If the front octant is homogeneously filled with some color, we do not process the back octant.
  - If the front is empty, it is necessary only to process the rear octant.
  - If the front octant has heterogeneous regions, it has to be subdivided and the sub-octants are handled recursively.

Chapter 8

Illumination Models and Surface Rendering Methods

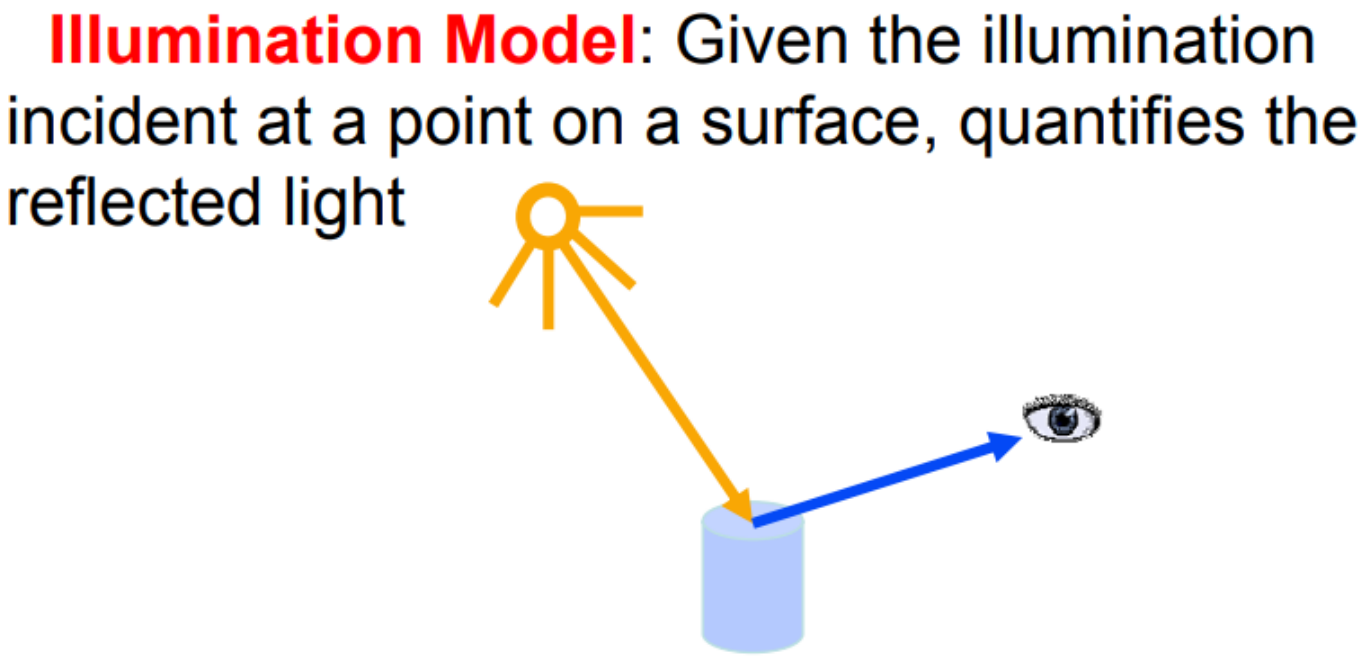
Motivation: In order to produce realistic images we must simulate the appearance of surfaces under various lighting conditions.

8.1 Introduction

- Realistic displays of a scene are obtained by perspective projections and applying natural lighting effects to the visible surfaces of object.
- For realistic displaying of 3d scene it is necessary to calculate appropriate color or intensity for that scene.
- The realism of a raster scan image of a 3D scene depends upon the successful stimulation of shading effects.
- Once visible surface has been identified by hidden surface algorithm, a shading model is used to compute the intensities and color to display for the surface.

8.1.1 Illumination model or a lighting model or shading model

- It is a model for the interaction of light with a surface.
- It is the model for calculating light intensity at a single surface point. Sometimes also referred to as a shading model.
- An illumination model is used to calculate the intensity of the light that is reflected at a given point on a surface.



8.1.2 Surface Rendering/Surface Shading

- Rendering/Shading is the process of creating a high quality realistic images or pictures.
- In 3-D graphic design, rendering is the process of add shading (how the color and brightness of a surface varies with lighting), color in order to create life-like images on a screen.
- Surface rendering is the process of calculating intensity values for all pixel positions for the various surfaces in a scene.
- A rendering method uses intensity calculations from the illumination model to determine the light intensity at all pixels in the image.
- **It is the process of applying illumination model to obtain the pixel intensities for all the projected surface positions in a scene.**
- Surface rendering can be performed by applying the illumination model to every visible surface point, or the rendering can be accomplished by interpolating intensities across the surface.

❖ **Importance of illumination models and rendering:**

- **to produce high quality realistic images**

8.1.3 Light Sources

❖ Object that radiates energy are called light sources, such as sun, lamp, bulb, fluorescent tube etc.

Point Light Source

The rays emitted from a point light radially diverge from the source. Approximation for sources that are small compared to the size of objects in the scene. Radiates equal intensity in all directions. For example sun.

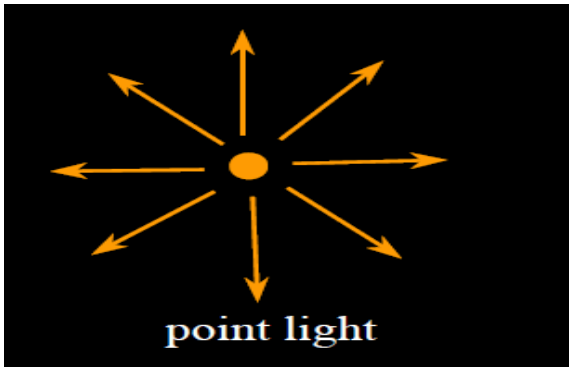


Fig: ***Diverging ray paths from a point light Source.***

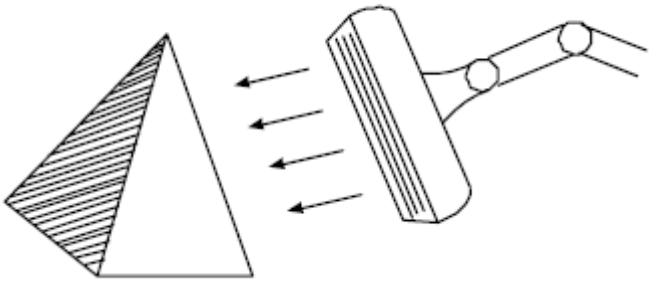


Fig: ***An object illuminated with a distributed light source.***

Distributed Light Source

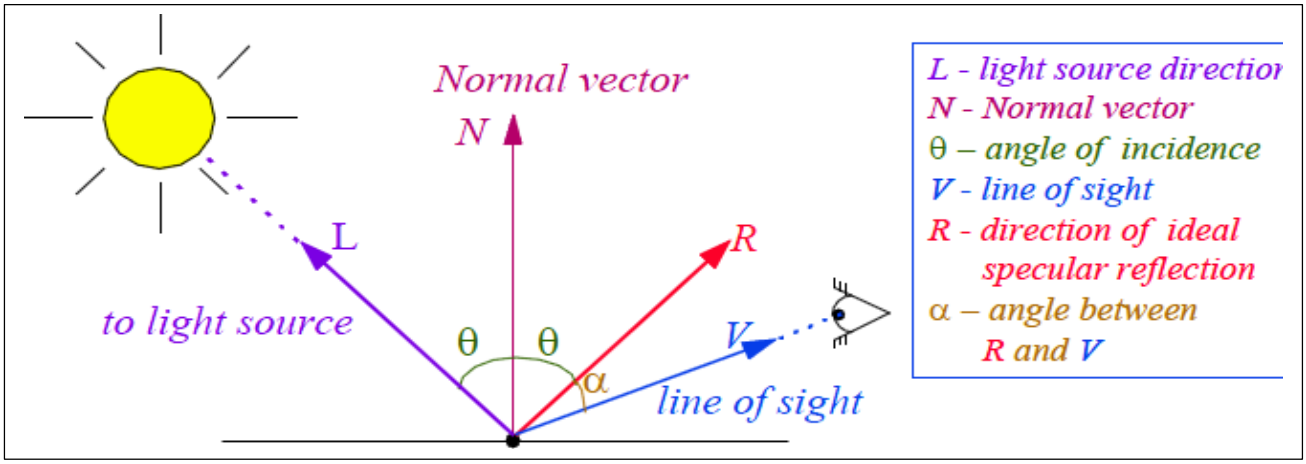
A nearby source, such as the long fluorescent light. All of the rays from a directional/distributed light source have the same direction, and no point of origin. All light rays are parallel.

Interaction of Light Source with Surfaces

❖ When light is incident on opaque surface part of it is reflected and part of it is absorbed.

- The amount of incident light reflected by a surface depends on the type of material. Shining material reflects more incident light and dull surface absorbs more of the incident light.
- For transparent surfaces, some of the incident light will be reflected and some will be transmitted through the material.

Some useful concepts:



8.2 Illumination Models

- Illumination models are used to calculate light intensities that we should see at a given point on the surface of an object.
- Intensity calculations are based on the optical properties of surfaces such as:
  - Reflectivity, opaque/transparent/translucent, shiny/dull, the background lighting conditions and the light source specifications.

Some illumination models are described in the following sections:

8.2.1 Ambient Light

- Surface that is not exposed directly to a light source still will be visible if nearby objects are illuminated. This light is called *ambient light*.
- This is a simple way to model the combination of light reflections from various surfaces to produce a *uniform illumination called the ambient light, or background light*.
- The amount of ambient light incident on each object is a constant for all surfaces and over all directions.
- If a surface is exposed only to ambient light, then the intensity of the diffuse reflection at any point on the surface is;

  $I = k_a I_a$. Where I_a is the intensity of the ambient light, and $\mathbf{k_a}$ is the ambient reflection coefficient.

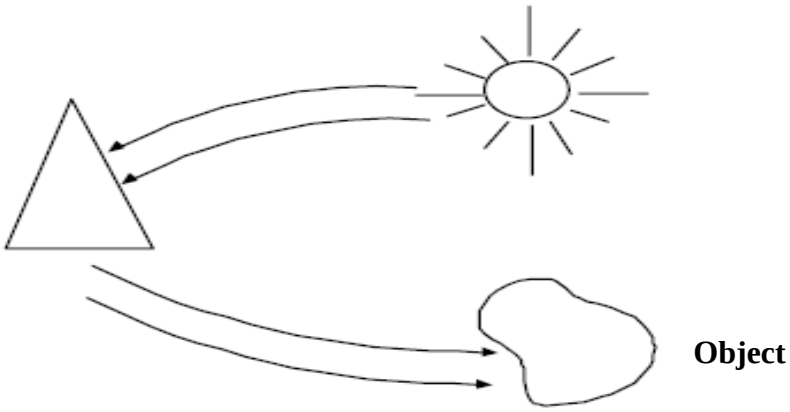


Fig: Object illuminated due to ambient light.

8.2.2 Diffuse reflection

- It is a reflection due to even scattering of light by uniform, rough surfaces.
- Rough surface tends to scatter the reflected light in all direction. The scattered light is called diffuse reflection. So surface appears equally bright from all viewing directions.

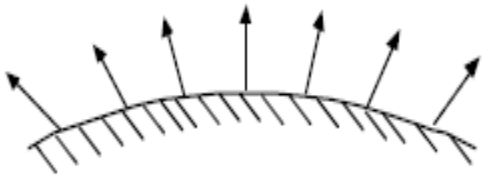


Fig: ***Diffuse reflections from a surface.***

- Diffuse reflections are constant over each surface in a scene, independent of the viewing direction. Surfaces appear equally bright from all viewing angles since they reflect light with equal intensity in all directions.
- Color of an object is determined by the color of the diffuse reflection of the incident light. If any object surface is red then there is a diffuse reflection for red component of light and all other components are absorbed by the surface.
- The intensity of diffuse reflection due to ambient light is;

$$I_{adiff} = k_a I_a \qquad \text{……1}$$

- **If surface is exposed to a point source, then intensity of diffuse reflection can be calculated by using Lambert's Cosine Law.**
  - **Lambert's Cosine Law:** The radiant energy from any small surface dA in any direction relative to surface normal is proportional to cos θ. That is, brightness depends only on the angle θ between the light direction L and the surface normal N.

∴ Light intensity α cos θ.

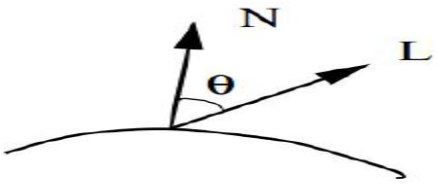


Fig: Angle of incidence θ between the unit light-source direction vector L and the unit surface normal N

- If $\mathbf{I_l}$ is the intensity of the point light source and $\mathbf{k_d}$ is the diffuse reflection coefficient, then the diffuse reflection for single point-source can be written as;

$$\mathbf{I_{pdiff} = k_d I_l \cos\theta}$$

$$\mathbf{I_{pdiff} = k_d I_l (N \cdot L)} \qquad \text{……2}$$

∴ Total diffuse reflection (I_{diff}) = Diff (due to ambient light)+Diff. due to pt.source.

$$I_{diff} = I_{adiff} + I_{pdiff}$$

$$I_{diff} = k_a I_a + k_d I_l (N \cdot L)$$

8.2.3 Specular reflection and Phong model

- In shiny surface we see high light or bright spot from certain viewing directions called specular reflection. Also referred to as an irregular reflection.
- Light source creates high lights or bright spots called ***specular reflection***. However this effect is seen more on shiny surface then dull surfaces. Example: Persons forehead.

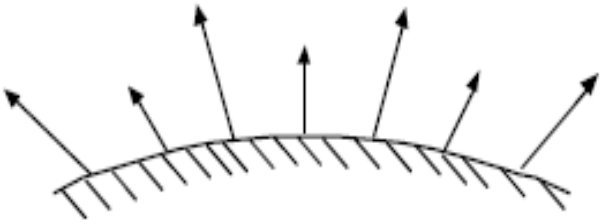
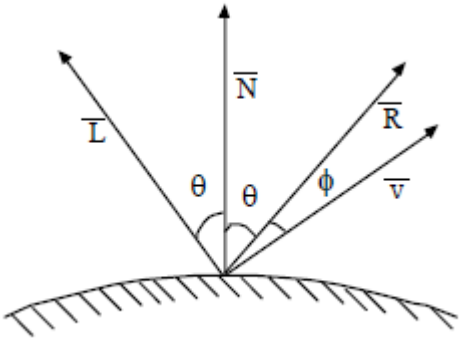


Fig: *Specular reflection.*

- Light is strongly reflected in one particular direction. This is due to total or nearly total reflection of light.
- For an ideal reflector, such as a mirror, the angle of incidence equals the angle of specular reflection (perfect mirror) = φ = 0.



- The empirical formula for calculating the specular reflection is given by **Phong Model**.
  - **Phong Model:** This is an empirical model, which is not based on physics, but physical observation. It sets the intensity of the specular reflection proportional to $\cos^{\eta s}\varphi$, where η_s is specular reflection parameter and is determined by the type of surface. For very shiny surface η_s is set 100 and for dull surface η_s is set 1. The intensity of specular reflection can be modeled using *specular reflection coefficient* W(θ).

    $$\therefore I_{spec} = W(\theta)\, I_l \cos^{\eta s}\varphi.$$

Where I_l is the intensity of the light source, θ angle of incidence. General variation of W(θ)is over the range 0^o-90^o. At $\theta = 90^o$, W(θ) = 1.

8.2.4 Intensity Attenuation

- As radiant energy from a point light source travels through space, its amplitude is attenuated by the factor $\mathbf{l/d^2}$, where d is the distance that the light has traveled.
- This means that *a surface close to the light source (small **d**) receives higher incident intensity from the source than a distant surface (large **d**).*
- ***Therefore to produce realistic lighting effects, illumination model should take intensity attenuation into account. Otherwise we are likely to illuminate all surfaces with same intensity.***
- For a point light source attenuation factor is $1/d^2$.
- And for a distributed light source attenuation factor is given by **inverse quadratic attenuation function**, $\mathbf{f(d) = 1/a_0 + a_1 d + a_2 d^2}$.

8.2.5 Transparency and Shadows

Transparency

A transparent surface, in general, produces both reflected and transmitted light. The relative contribution of the transmitted light depends on the degree of transparency of the surface and whether any light sources or illuminated surfaces are behind the transparent surface.

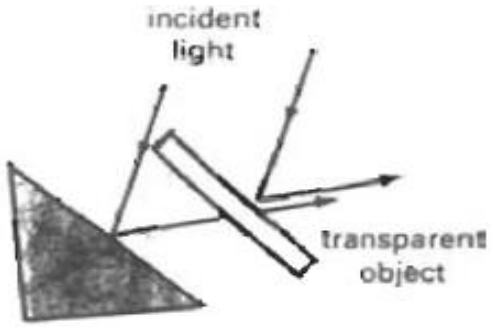


Fig: Light emission from a transparent surface is in general a combination of reflected and transmitted light.

When light is incident upon a transparent surface, part of it is reflected and part is refracted.

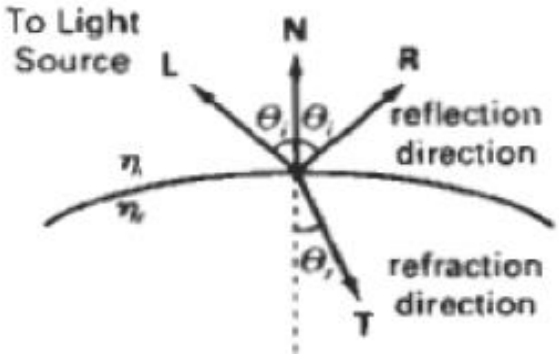


Fig: Reflection direction R and refraction direction T for a ray of light incident upon a surface with index of refraction $\acute{\eta}_r$

According to Snell's law,

$$\sin\theta_r = \frac{\eta_i}{\eta_r}\sin\theta_i$$

Where, θ_i is an angle of incidence, θ_r is an angle of refraction, $\acute{\eta}$ is refractive index.

Shadows

Shadow can help to create realism. Without it, a cup, e.g., on a table may look as if the cup is floating in the air above the table. ***By applying hidden-surface methods with pretending that the position of a light source is the viewing position, we can find which surface sections cannot be "seen" from the light source => shadow areas. We usually display shadow areas with ambient-light intensity only***.

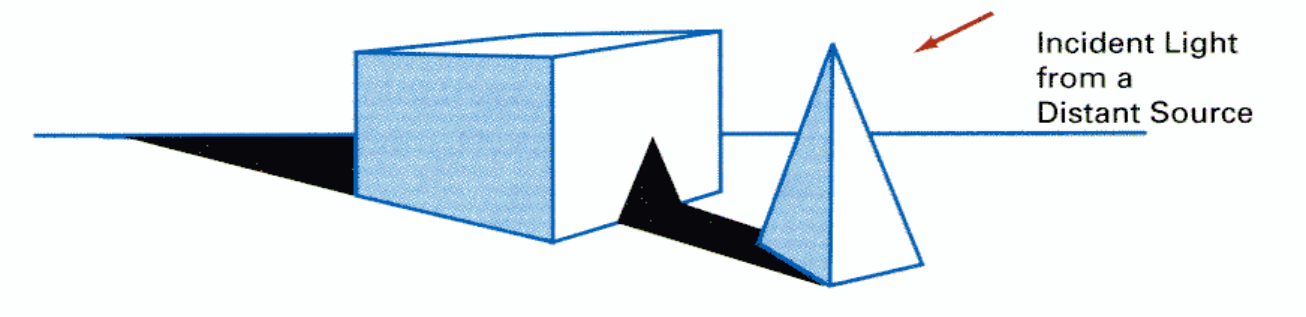


Fig: Shadow

8.3 Surface Rendering

- Surface-rendering procedures are also called surface-shading methods.
- **It is the process of applying illumination model to obtain the pixel intensities for all the projected surface positions in a scene.**
- Each surface can be rendered using:
  1. **Rendering entire surface with a single intensity**: very fast but does not produce realistic surfaces.
  2. **Interpolation Scheme:** intensity values are interpolated to render the surfaces. Widely used approach, produces more realistic object surfaces than first method. **Still suffers from Mach Band Effect**.
  3. **By applying the illumination model to every visible surface point**: best option, widely used approach, produces best quality surfaces, but requires large computations, so comparatively slow.
- **Three widely used approaches:**
  - **Constant Intensity shading Method (Flat Shading)**
  - **Gouraud Shading method (Intensity Interpolation)**
  - **Phong Shading Method (Normal Vector Interpolation).**

8.3.1 Constant Intensity shading Method (Flat Shading)

- The fast and simplest model for shading/rendering a polygon is constant intensity shading also called faceted Shading or flat shading.
- In this approach, the illumination model is applied only once for each polygon to determine a single intensity value.
- The entire polygon is then displayed with the single intensity value.

- It does not provide realistic displaying.
- **It provides an accurate rendering for an object if all of the following assumptions are valid:**
  - Polygon surface must be one face of a polyhedron and is not a section of a curved-surface.
  - The light source is sufficiently far so that $\underset{N}{\rightarrow}.\underset{L}{\rightarrow}$ is constant across the polygon face.
  - The viewing position is sufficiently far from the surface so that $\underset{V}{\rightarrow}.\underset{R}{\rightarrow}$ is constant over the surface.

Algorithm:

- Divide polygon surface into polygon meshes.
- Determine surface unit normal vectors for each polygon.
- Calculate intensity value for a point for each surface (usually at the center).
- Apply this intensity value to all the points of that surface.

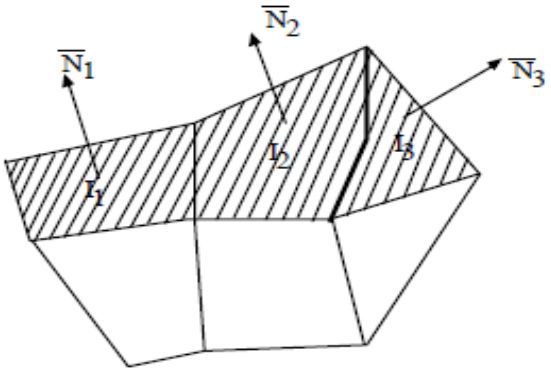


Fig: Flat Shading

8.3.2 Gouraud Shading

- It is an intensity interpolating shading or color interpolating shading introduced by Henri Gouraud.
- The polygon surface is displayed by linearly interpolating intensity values across the surface.
- **Idea is to calculate intensity values at polygon vertices. Then, linearly interpolate these intensities across polygon surfaces of an object**.

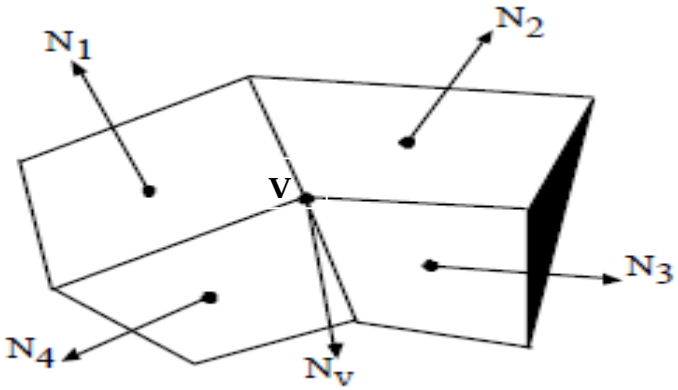


Fig: Gouraud Shading

Algorithm:

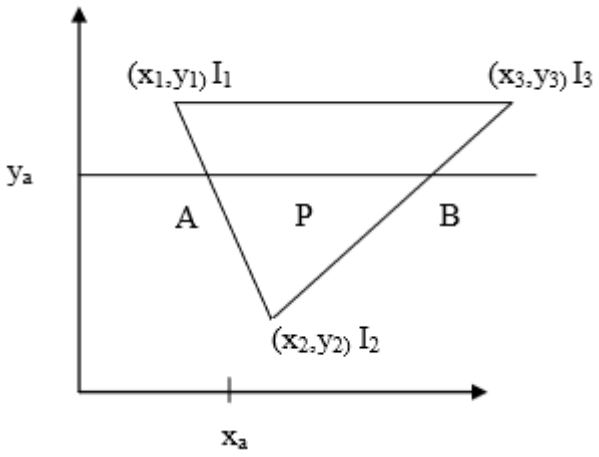
- Determine the average unit normal vector at each polygon vertex.
  - At each polygon vertex, we obtain a normal vector by averaging the surface normals of all polygons sharing that vertex.
  - Therefore, average unit normal vector at vertex V, is given by-

$$Nv = \frac{\sum_{k=1}^{n} N_k}{|\sum_{k=1}^{n} N_k|}$$, where n is the number of adjoining surfaces.

For above figure;

$$Nv = \frac{\sum_{k=1}^{4} N_k}{|\sum_{k=1}^{4} N_k|} = \frac{N_1+N_2+N_3+N_4}{|N_1+N_2+N_3+N_4|}$$

- Apply an illumination model to each vertex to calculate the vertex intensity.
- Linearly interpolate the vertex intensities over the surface of the polygon.
  - Interpolation of intensities can be calculated as follows:



Here in the figure, intensity of vertices 1, 2, 3 are I_1 , I_2, I_3 which are obtained by averaging normals of each surface sharing vertices & applying an illumination model. For each scan line, intensity at intersection of the line with Polygon edge is linearly interpolated from intensities at edge end point.

So Intensity at intersection point A, I_a is obtained by the linearly interpolating intensities of I_1 and I_2 as

$$I_a = \frac{y_a - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_a}{y_1 - y_2} I_2$$

Similarly, the intensity at point B is obtained by linearly interpolating intensities at I_2 and I_3 as

$$I_b = \frac{y_a - y_2}{y_3 - y_2} I_3 + \frac{y_3 - y_a}{y_3 - y_2} I_2$$

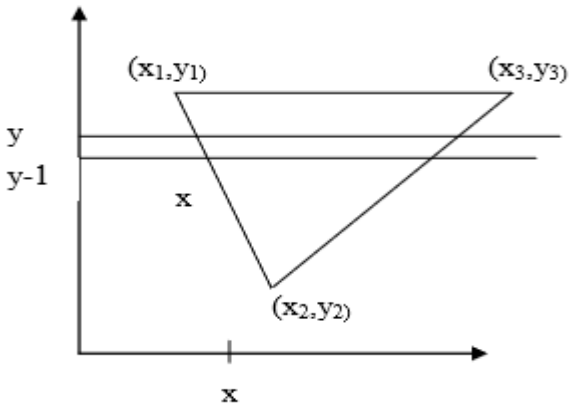
The intensity of a point P in the polygon surface along scan-line is obtained by linearly interpolating intensities at I_a and I_b as,

$$I_p = \frac{x_p - x_a}{x_b - x_a} I_b + \frac{x_b - x_p}{x_b - x_a} I_a$$

Then incremental calculations are used to obtain Successive edge intensity values between scan-lines as :

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

Then we can obtain the intensity along this edge for next scan line at y -1 position as



$$I' = \frac{y - 1 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - (y - 1)}{y_1 - y_2} I_2$$

$$= I + \frac{I_2 - I_1}{y_1 - y_2}$$

Similar calculations are made to obtain intensity successive horizontal pixel.

Advantages:

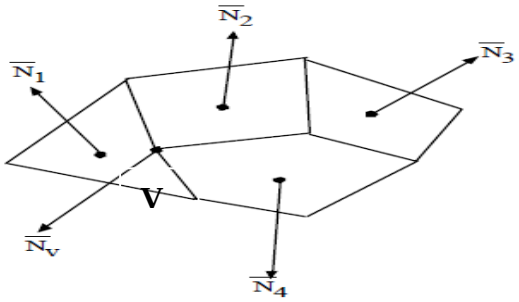
- It provides more realistic graphics than constant intensity shading.
- It eliminates intensity discontinuities that occur in flat shading.

Disadvantages:

- It can cause bright or dark intensity streaks to appear on the surface called **Mach banding**.
- Involves additional computation.

8.3.3 Phong Shading

- Best known shading algorithm, developed by *Phong Bui Tuong*, is called Phong shading or normal vector interpolation shading.
- A more accurate method for rendering a polygon surface.
- Idea here is to interpolate normal vectors instead of the light intensity and then apply the illumination model to each surface point.
- **Basic Idea:**
  - Phong shading calculates the average unit normal vector at each of the polygon vertices and then interpolates the vertex normal over the surface of the polygon.



Algorithm:

1. **Determine the average unit normal vector at each polygon vertex.**
   - At each polygon vertex, we obtain a normal vector by averaging the surface normals of all polygons sharing that vertex.
   - Therefore, Unit normal vector at vertex V, is given by-

$$Nv = \frac{\sum_{k=1}^{n} N_k}{|\sum_{k=1}^{n} N_k|}$$, where n is the number of adjoining surfaces.

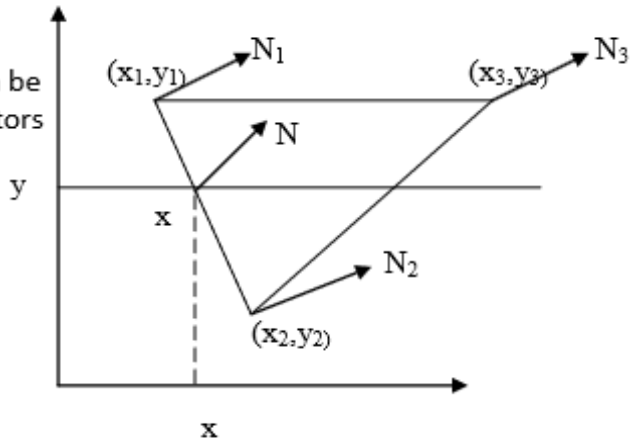
For above figure;

$$Nv = \frac{\sum_{k=1}^{3} N_k}{|\sum_{k=1}^{3} N_k|} = \frac{N_1+N_2+N_3}{|N_1+N_2+N_3|}$$

2. **Linearly interpolate the vertex normals over the surface of the polygon.**

In figure, N_1, N_2, N_3 are the normal unit vectors at each vertex of polygon surface. For scan-line that intersect an edge, the normal vector N can be obtained by vertically interpolating normal vectors of the vertex on that edge as.

$$N = \frac{y-y_2}{y_1-y_2}N_1 + \frac{y_1-y}{y_1-y_2}N_2$$



3. **Apply an illumination model at each pixel positon along each scan line to calculate pixel intensity at that point.**

Advantages:

- ❖ It provides more realistic highlights on a surface.
- ❖ It reduces the Mach-Band effect.
- ❖ It gives more accurate results.

Disadvantages:

- ❖ It requires more calculations.

- ❖ It increases the cost of shading.

Fast Phong Shading

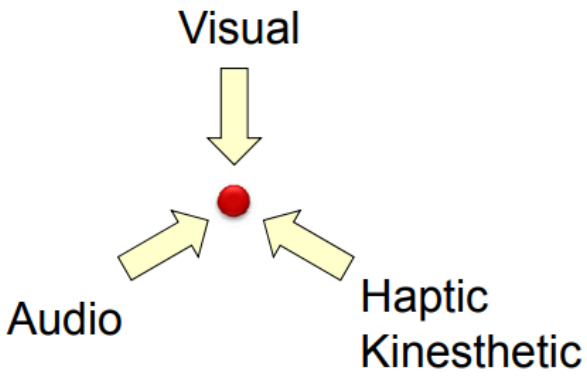
Surface rendering with Phong shading can be speeded up by using approximations in the illumination model calculations of normal vectors. Fast Phong shading approximates the intensity calculations using a Taylor series expansion and Triangular surface patches.

Chapter 9

Introduction to Virtual Reality

9.1 Introduction

- Virtual reality is an artificial reality that makes users to feel in a virtual environment by using the computer hardware and software.
- Virtual reality (VR) refers to the computer-generated simulation in which a person can interact within an artificial three-dimensional environment.
- VR immerse the user in a computer-generated environment that simulates reality through the use of interactive devices, which send and receive information and are worn as goggles, headsets, gloves, or body suits.
- The purpose of VR is to allow a person to experience and manipulate the environment as if it were the real world.
- **Virtual reality can be divided into**:
    - The simulation of a real environment for training and education.
    - The development of an imagined environment for a game or interactive story.



The best virtual realities are able to immerse the user completely. Virtual reality should not be confused with simple 3-D environments like those found in computer games, where you get to experience and manipulate the environment through an avatar, rather than personally becoming part of the virtual world.

Virtual reality can be divided into:

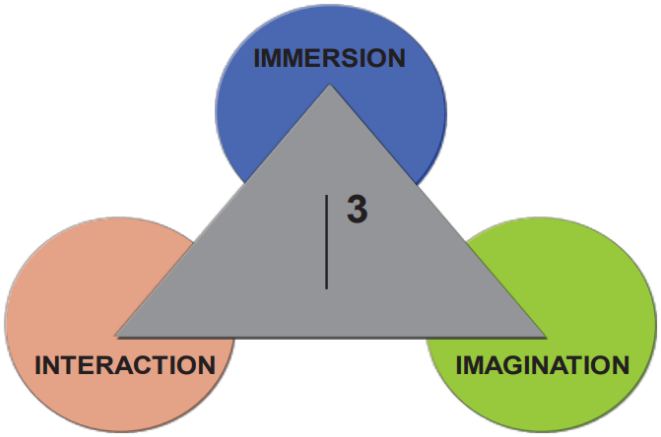
- The simulation of a real environment for training and education.
- The development of an imagined environment for a game or interactive story.

The system responsible for running the virtual environment must be able to track the user's motions, especially the eye and head movements, so that it can react and change the images on the display or initiate any related events.

In order to immerse the user fully, there are two major components:

- **Depth of Information**:
    - Refers to the quality and amount of data the user is fed by the virtual environment itself. This could be achieved through the display resolution, graphics quality and complexity of the environment, sound quality, & haptic feedback (sensation of touch).
- **Breadth of Information**:
    - Refers to how many senses are being stimulated by the virtual environment. The most basic of these should be audio and visual, while the most advanced systems should include stimulation of all five senses in order to enhance immersion.

9.1.1 Virtual Reality Triangle



The Three I's of Virtual Reality:

The Three I's of Virtual Reality:

- ***Immersion***
    - The feeling of presence, being there.
    - The amount and quality of stimuli and sensations.
- ***Interaction***
    - Not just passive watching.
    - Interacting with the virtual world.
    - Real time: actions can immediately modify the state.
- ***Imagination***
    - Forming mental images, sensations, and concepts in a moment when they are not perceived through sight.

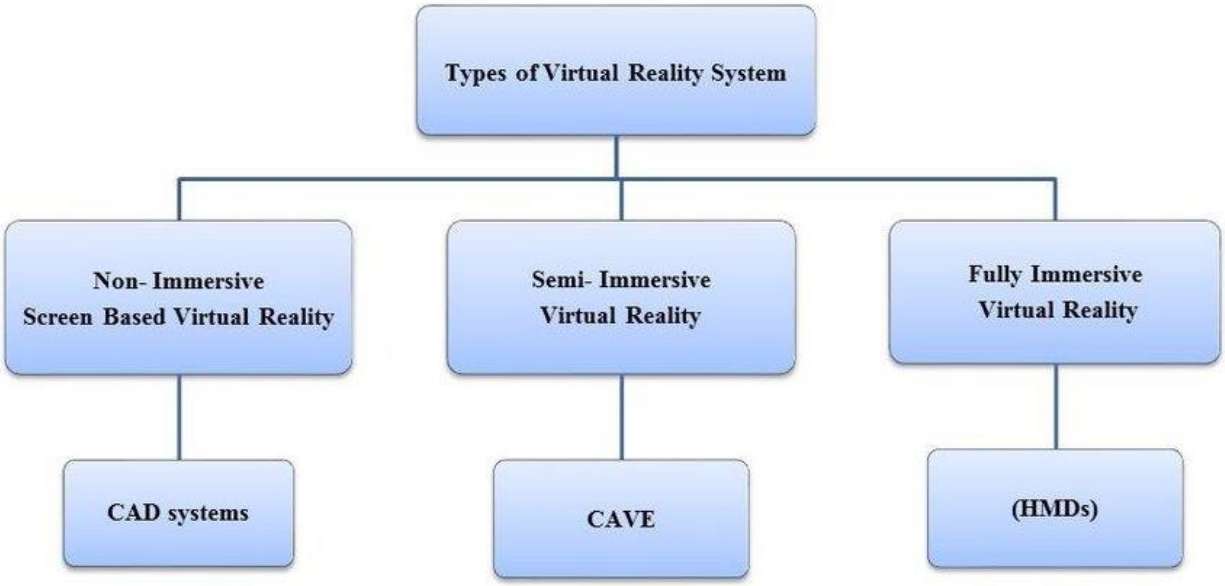
9.1.2 Properties of VR

- Synthetically generated environment
    - Computers, 3D, real-time
- Sensory feedback
    - I/O devices
- Interaction, moving
    - In time
    - In space
    - In scale
- Immersion
    - Being there

9.1.3 Types of VR

VR differ in their levels of immersion:

- Non-Immersive
- Semi-Immersive
- Fully-Immersive

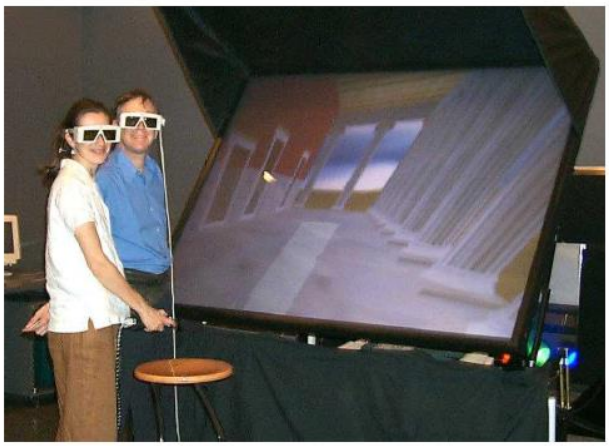


1. Non-Immersive VR

- Non-immersive VR system is also called Desktop VR system, fish tank or window on world system.
- Non-immersive VR system is least immersive and least expensive of the VR systems, as it requires the least sophisticated components.
- It allows user to interact with a 3D environment through a stereo display monitor and glasses.

Flat/curved screen VR

Basic stereoscopic 3D

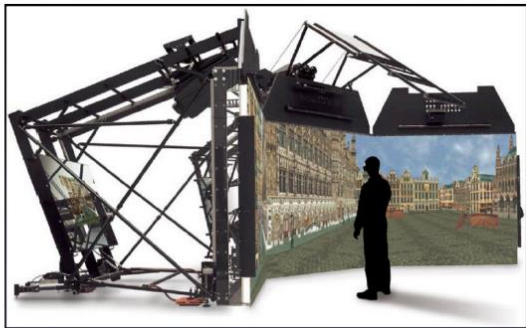


2. Semi-Immersive VR

- Semi-immersive VR system provides high level of immersion, while keeping the simplicity of the desktop VR system or utilizing some physical model.
- Example of such system includes: CAVE (Cave Automatic Virtual Environment) and an application is the driving simulator.

Surround-projection: The CAVE system, VR domes

- Shared experience: 1 tracked user, others share the view
- Physical scale interaction and movement



3. Fully-Immersive VR

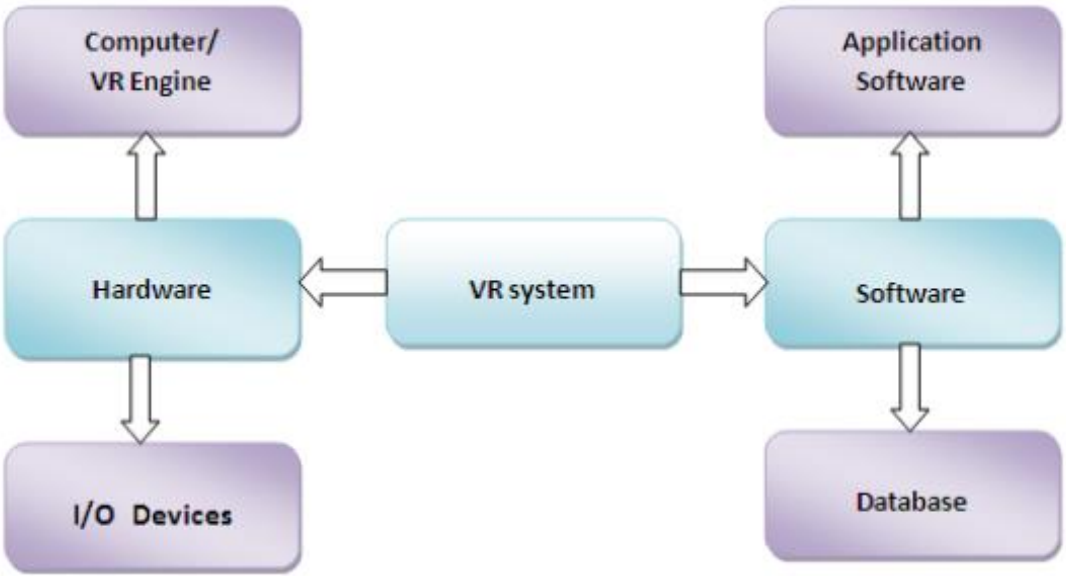
- Immersive VR system is most expensive and gives the highest level of immersion.
- It's component includes HMD(Head Mounted Display), tracking devices, data gloves and others, which encompasses the user with computer generated 3D animation that gives the user the feeling of the part of the virtual environment.

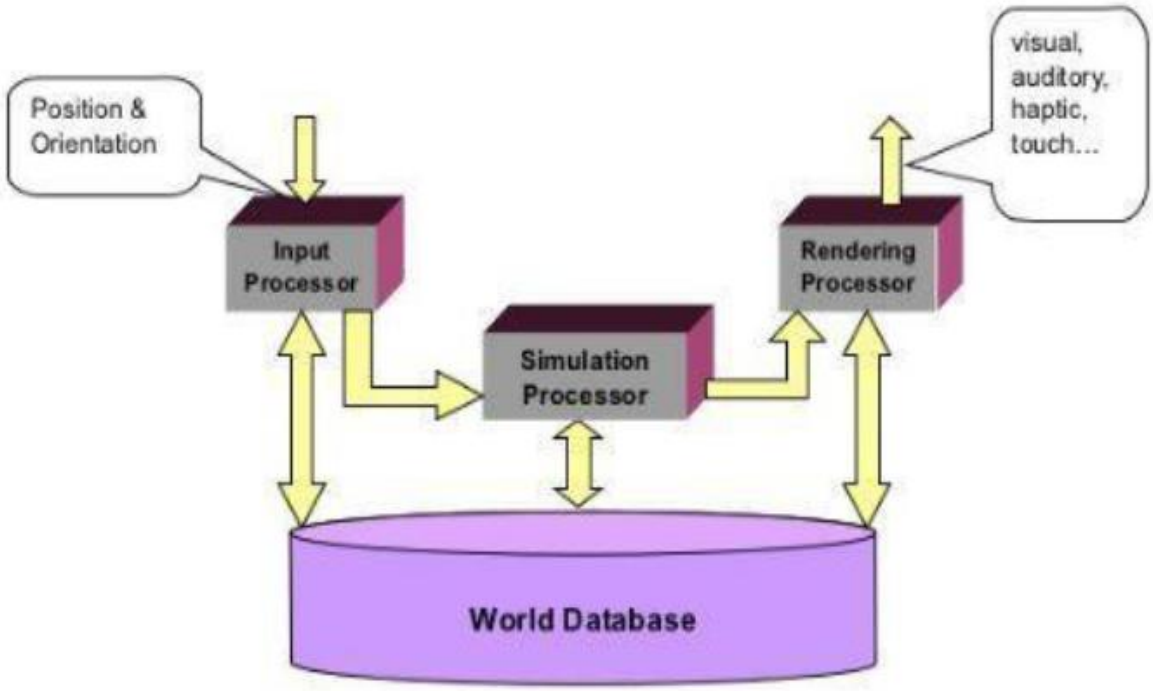
HMDs **(Head Mounted Display's)**

- Personal/egocentric experience
- Individualized single-/multi-user experience
- Inconvenient/unnatural motion (navigation) metaphors

9.1.4 The VR System Architecture/Components of Virtual Reality Systems

A VR system is made up of 2 major subsystems; the **hardware and software**. The **hardware** can be further divided into computer or VR engine and I/O devices, while the **software** can be divided into application software and database as illustrated below.





Input Processor/Input Devices

- Controls the devices used to input information to the computer.
- The objective of input processor is to get the coordinate data and provide to the rest of the system with minimum time.
- The input devices are the means by which the user interacts with the virtual world. They send signals to the system about the action of the user, so as to provide appropriate reactions back to the user through the output devices in real time.
- They can be classified into ***tracking device, point input device, bio-controllers and voice device***.
- Tracking devices sometimes referred to as position sensors, are used in tracking the position of the user, and they include, electromagnetic, ultrasonic, optical, mechanical and gyroscopic sensors, data gloves, neural and bio or muscular controllers.
- Examples of point-input devices include 6DOF (6 degrees of freedom) mouse and force or space ball. Their technology is an adaptation of the normal mouse with extended functions and capability for 3D.
- Voice communication is a common way of interaction among humans. So it feels natural to incorporate it into a VR system. Voice recognition or processing software can be used in accomplishing this.
- Example of input processor are:
    - Mouse, Keyboard, 3D position trackers, a voice recognition system etc.

Simulation Processor/VR Engine

- Simulation processor is the core component of VR system.
- It takes the user input along with any task programmed into the world and determine the action that will take place in the virtual world.
- In VR systems, the VR engine or computer system has to be selected according to the requirement of the application. Graphic display and image generation are some of the most important factors and time consuming task in a VR system.
- The choice of the VR engine depends on the application field, user, I/O devices, level of immersion and the graphic output required, since it is responsible for calculating and generating graphical models, object rendering, lighting, mapping, texturing, simulation and display in real-time.
- The VR engine also handles the interaction with users and serves as an interface with the I/O devices.
- The VR engine is required to recalculate the virtual environment approximately every 33ms and produce real time simulation of more than 24fps, furthermore, the associated graphic engine should be capable of producing stereoscopic vision.

Rendering Processor/Output Devices

- Rendering processor creates the sensations that are output to the user.
- The separate rendering processor are used for visual, auditory, haptic and other sensory systems.
- The output devices get feedback from the VR engine and pass it on to the users through the corresponding output devices to stimulate the senses.
- The possible classifications of output devices based on the senses are: graphics (visual), audio (aural), haptic (contact or force), smell and taste. Of these, the first 3 are frequently used in VR systems, while smell and taste are still uncommon.
- Two possible common options for the graphics are the stereo display monitor, and the HMD which provides a higher level of immersion. In the HMD, the two independent views produced are interpreted by the brain to provide a 3D view of the virtual world.
- Audio or sound is an important channel in VR; its importance is only surpassed by that of visual. 3D sound can be used in producing different sounds from different location to make the VR application more realistic.
- Haptic is used to allow the user feel virtual objects. This can be achieved through electronic signals or mechanical devices.

World Database

- The world database store the objects that exist in the world and scripts that describes the actions of those objects.

9.2.5 3D Position Tracker

- Positional tracking detects the precise position of the head-mounted displays, controllers, other objects or body parts in VR system.
- Position tracking and mapping of the human body and environments is a basic requirement that permeates virtual environment (VE) system.
- **Types**:
    - Head and eye tracking for visual displays.
    - Hand and arm tracking for haptic interfaces.
    - Body tracking for locomotion and visual displays.
    - Body surface mapping for facial expression recognizers, virtual clothiers, and medical telerobots.
    - Environment mapping to build a digitized geometrical model for simulation.

9.2.6 Visual Computation in Virtual Reality

- A typical virtual reality (VR) system uses computer-graphic imagery displayed to a user through a head-mounted display (HMD) to create a perception in the user of a surrounding three-dimensional virtual world.
- It does this by tracking the position and orientation of the user's head and rapidly generating stereoscopic images in coordination with the user's voluntary head movements as the user looks around and moves around in the virtual world.
- The hardware for a typical VR system consists of an HMD for visual input, a tracker for determining position and orientation of the user's head and hand, a graphics computer for generating the correct images based on the tracker data, and a hand-held input device for initiating actions in the virtual world. The visual environment surrounding the user is called the virtual world. The world contains objects, which are collections of graphics primitives such as polygons.
- Each object has its own position and orientation within the world, and may also have other attributes.
- The human being wearing the HMD is called the user, and also has a location and orientation within the virtual world.

9.1.7 VR Applications

- Training, simulators
    - Flight, cars, military, surgery, etc.
- CAD / prototypes / visualization
    - Useful when designing many kinds of products
    - Architecture, oil exploration, Boeing 777.
    - Marketing
- Entertainment, casinos, games.
- Architecture, Arts, Business, Design and Planning, Education and Training, Entertainment, Manufacturing, Medical and Scientific Visualization.

9.2 Augmented Reality (AR)

Augmented reality is the integration of digital information with the user's environment in real time. Unlike virtual reality, which creates a totally artificial environment, augmented reality uses the existing environment and overlays new information on top of it.

Snapchat filters, instagram filters, Pokemon Go are all Examples of AR. It is these AR apps which allow a customer to place virtual furniture in their house before buying. Projection based AR applications allow human interaction by sending light onto a real surface and then sensing human touch, learning has become way more intresting and easy with AR, markerless AR provides data based on our location, it is also a powerful tool for marketing as it allows users to try products before buying.

Augmented Reality (AR) is a variation of VR that allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. Therefore, AR supplements reality, rather than completely replacing it

Key characteristics:

- Combines real and virtual world
- Interactive in real time
- Registered in 3-D (real and virtual objects are in a 3D relation to each other)

Augmented reality (AR) is a type of interactive, reality-based display environment that takes the capabilities of computer generated display, sound, text and effects to enhance the user's real-world experience. Augmented reality combines real and computer-based scenes and images to deliver a unified but enhanced view of the world. Augmented reality has many different implementation models and applications, but its primary objective is to provide a rich audiovisual experience.

AR works by employing computerized simulation and techniques such as image and speech recognition, animation, head-mounted and hand-held devices and powered display environments to add a virtual display on top of real images and surroundings.

Chapter 10

Introduction to OpenGL

10.1 Introduction

- OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics.
- Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering.
- OpenGL is the software interface to graphics hardware.
- In other words, OpenGL graphic rendering commands issued by your applications could be directed to the graphic hardware and accelerated.
- OpenGL program:
    - only responds to events
    - Do nothing until event occurs
        - **Example Events:**
            - mouse clicks
            - keyboard stroke
            - window resize
        - **Programmer:**
            - defines events
            - actions to be taken
- There are mainly 3 sets of libraries in OpenGL programs:
    - **Core OpenGL (GL):**
        - Consists of hundreds of commands, which begin with a prefix "gl" (e.g., glColor, glVertex, glTranslate, glRotate). The Core OpenGL models an object via a set of geometric primitives such as point, line and polygon.
    - **OpenGL Utility Library (GLU):**
        - Built on-top of the core OpenGL to provide important utilities (such as setting camera view and projection) and more building models (such as qradric surfaces and polygon tessellation). GLU commands start with a prefix "glu" (e.g., gluLookAt, gluPerspective).

- **OpenGL Utilities Toolkit (GLUT):**
  - OpenGL is designed to be independent of the windowing system or operating system. GLUT is needed to interact with the Operating System (such as creating a window, handling key and mouse inputs); it also provides more building models (such as sphere and torus).
  - GLUT commands start with a prefix of "glut" (e.g., glutCreatewindow, glutMouseFunc).
  - GLUT is platform independent, which is built on top of platform-specific OpenGL extension such as GLX for X Window System, WGL for Microsoft Window, and AGL, CGL or Cocoa for Mac OS.
  - GLUT is designed for constructing small to medium sized OpenGL programs.
  - GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit.

10.2 Program Structure

- Configure and open window (GLUT)
- Initialize OpenGL
- Register input callback functions (GLUT)
  - Render
  - Resize
  - Input: keyboard, mouse, etc.
- My initialization
  - Set background color, clear color, drawing color, point size, establish coordinate system, etc.
- glutMainLoop( )
  - Waits here infinitely till action is selected

Example:

```
/*
 * GL02Primitive.cpp: Vertex, Primitive and Color
 * Draw Simple 2D colored Shapes: quad, triangle and polygon.
 */
#include <GL/glut.h>  // GLUT, include glu.h and gl.h
/* Handler for window-repaint event. Call back when the window first appears and
   whenever the window needs to be re-painted. */
void display() {
   //Set "clearing" or background color
```

```
glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer with current clearing color
// Define shapes enclosed within a pair of glBegin and glEnd
glBegin(GL_QUADS);              // Each set of 4 vertices form a quad
   glColor3f(1.0f, 0.0f, 0.0f); // Red
   glVertex2f(-0.8f, 0.1f);     // Define vertices in counter-clockwise (CCW) order
   glVertex2f(-0.2f, 0.1f);     //  so that the normal (front-face) is facing you
   glVertex2f(-0.2f, 0.7f);
   glVertex2f(-0.8f, 0.7f);

   glColor3f(0.0f, 1.0f, 0.0f); // Green
   glVertex2f(-0.7f, -0.6f);
   glVertex2f(-0.1f, -0.6f);
   glVertex2f(-0.1f,  0.0f);
   glVertex2f(-0.7f,  0.0f);

   glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
   glVertex2f(-0.9f, -0.7f);
   glColor3f(1.0f, 1.0f, 1.0f); // White
   glVertex2f(-0.5f, -0.7f);
   glColor3f(0.2f, 0.2f, 0.2f); // Dark Gray
   glVertex2f(-0.5f, -0.3f);
   glColor3f(1.0f, 1.0f, 1.0f); // White
   glVertex2f(-0.9f, -0.3f);
glEnd();
glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
   glColor3f(0.0f, 0.0f, 1.0f); // Blue
   glVertex2f(0.1f, -0.6f);
   glVertex2f(0.7f, -0.6f);
   glVertex2f(0.4f, -0.1f);
   glColor3f(1.0f, 0.0f, 0.0f); // Red
   glVertex2f(0.3f, -0.4f);
   glColor3f(0.0f, 1.0f, 0.0f); // Green
   glVertex2f(0.9f, -0.4f);
   glColor3f(0.0f, 0.0f, 1.0f); // Blue
```

```
    glVertex2f(0.6f, -0.9f);
  glEnd();

  glBegin(GL_POLYGON);            // These vertices form a closed polygon
    glColor3f(1.0f, 1.0f, 0.0f); // Yellow
    glVertex2f(0.4f, 0.2f);
    glVertex2f(0.6f, 0.2f);
    glVertex2f(0.7f, 0.4f);
    glVertex2f(0.6f, 0.6f);
    glVertex2f(0.4f, 0.6f);
    glVertex2f(0.3f, 0.4f);
  glEnd();
  glFlush();  // Render now
}
/* Main function: GLUT runs as a console application starting at main()  */
int main(int argc, char** argv) {
  glutInit(&argc, argv);          // Initialize GLUT
  glutCreateWindow("2D Vertex, Primitive & Color");  // Create window with the given title
  glutInitWindowSize(320, 320);   // Set the window's initial width & height
  glutInitWindowPosition(50, 50); // Position the window's initial top-left corner
  glutDisplayFunc(display);       // Register callback handler for window re-paint event
  glutMainLoop();                 // Enter the event-processing loop
  return 0;
}
```

10.3 GLUT Callback Functions

- Register all events your program will react to
- Event occurs => system generates callback
- Callback: routine system calls when event occurs
- No registered callback = no action
- GLUT Callback functions example:
    - glutDisplayFunc(myDisplay): window contents need to be redrawn
    - glutReshapeFunc(myReshape): called when window is reshaped
    - glutMouseFunc(myMouse): called when mouse button is pressed
    - glutKeyboardFunc(mykeyboard): called when keyboard is pressed or released

- glutMainLoop( ): program draws initial picture and enters infinite loop till event.

Example: Rendering Callback

- Do all drawing in the display function
- Called initially and when picture changes (e.g.resize)
- First, register callback in main( ) function

```
glutDisplayFunc( display );
```

Then, implement display function

```
void display( void )
{  // put drawing stuff here
        …….
        glBegin( GL_LINES );
                glVertex3f( 0.2, 0.3, 0.6);
                glVertex3fv(0.6, 0.8, 0.4);
        ……………
        glEnd();
}
```

10.4 Naming Convention for OpenGL Functions

- An OpenGL function begins with lowercase gl (for core OpenGL), glu (for OpenGL Utility) or glut (for OpenGL Utility Toolkit), followed by the purpose of the function, in camel case (initial-capitalized), e.g., glColor to specify the drawing color, glVertex to define the position of a vertex. followed by specifications for the parameters, e.g., glColor3f takes three float parameters. glVectex2i takes two int parameters.
- (This is needed as C Language does not support function overloading. Different versions of the function need to be written for different parameter lists.)

10.5 Color

We use glColor function to set the foreground color, and glClearColor function to set the background (or clearing) color.

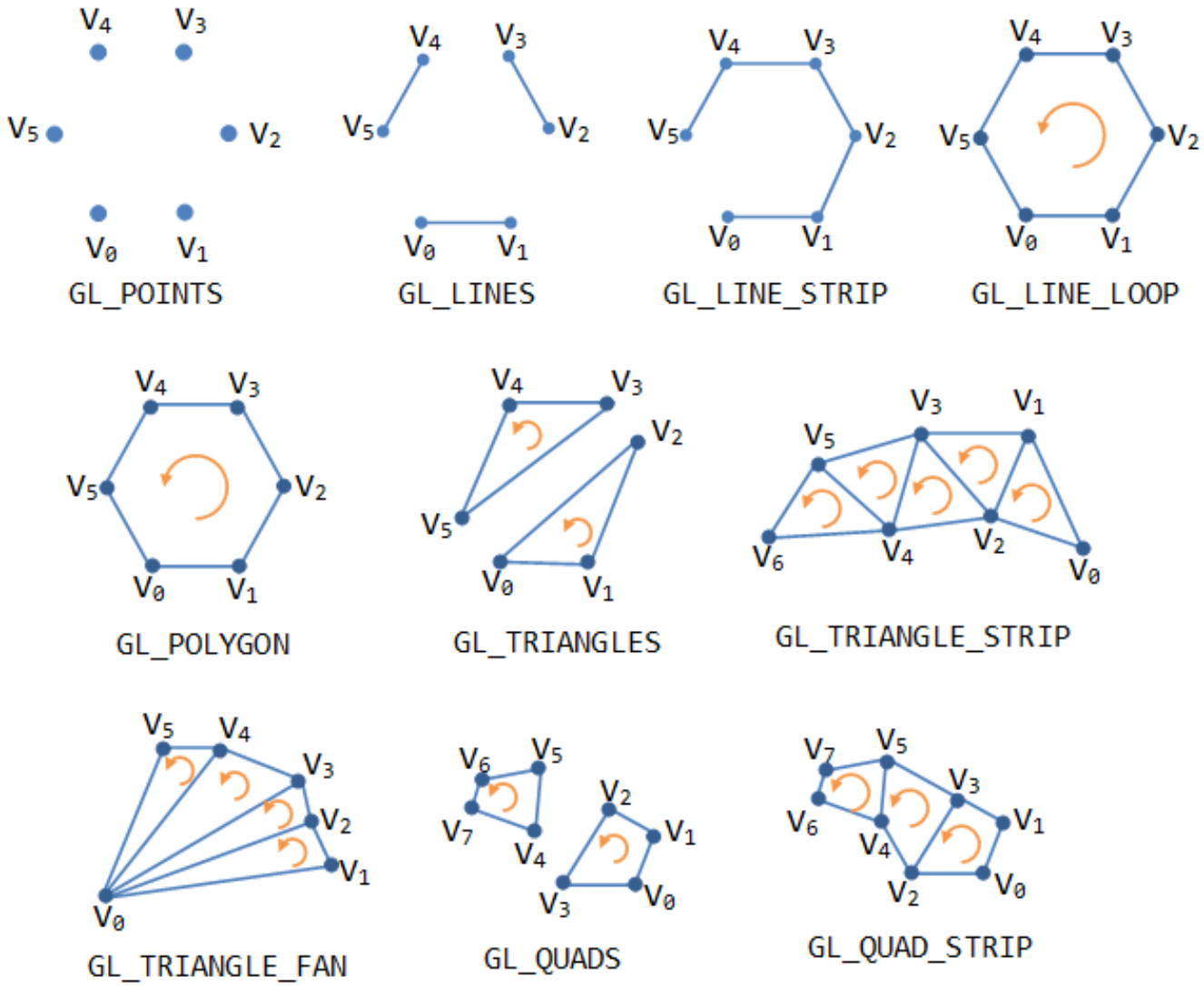
```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue)
void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha)
                                                    // GLclampf in the range of 0.0f to 1.0f
```

- Color is typically specified in float in the range 0.0f and 1.0f.

- Color can be specified using RGB (Red-Green-Blue) or RGBA (Red-Green-Blue-Alpha) components. The 'A' (or alpha) specifies the transparency (or opacity) index, with value of 1 denotes opaque (non-transparent and cannot see-thru) and value of 0 denotes total transparent. We shall discuss alpha later.
- In the above example, we set the background color via glClearColor, with R=0, G=0, B=0 (black) and A=1 (opaque and cannot see through).

10.6 OpenGL Primitives

In OpenGL, an object is made up of geometric primitives such as triangle, quad, line segment and point. A primitive is made up of one or more vertices. OpenGL supports the following primitives:



OpenGL Primitives

10.7 Drawing OpenGL Primitives

- To create a geometric object or model, we use a pair of ***glBegin***(PrimitiveType) and ***glEnd***() to enclose the vertices that form the model.
- For primitiveType that ends with 'S' (e.g., GL_QUADS), we can define multiple shapes of the same type.
- Example:

```
glBegin(GL_TRIANGLES);          // Each set of 3 vertices form a triangle
   glColor3f(0.0f, 0.0f, 1.0f);       // Blue
   glVertex2f(0.1f, -0.6f);
   glVertex2f(0.7f, -0.6f);
   glVertex2f(0.4f, -0.1f);
glEnd();
```

Complete Example: Drawing a 3D polygon with

```
#include <GL/glut.h>  // GLUT, include glu.h and gl.h
/* Handler for window-repaint event. Call back when the window first appears and
  whenever the window needs to be re-painted. */
void display() {
  // Set "clearing" or background color
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Black and opaque
  glClear(GL_COLOR_BUFFER_BIT);   // Clear the color buffer with current clearing color
  // Define shapes enclosed within a pair of glBegin and glEnd
  glBegin(GL_POLYGON);            // These vertices form a closed polygon
    glColor3f(1.0f, 1.0f, 0.0f);       // Yellow
    glVertex3f(0.4f, -0.2f,0.8);
    glVertex3f(0.6f, 0.2f,0.3);
    glVertex3f(-0.7f, 0.4f,-0.4);
    glColor3f(0.0, 1.0, 1.0);             //Cyan
    glVertex3f(0.6f, 0.6f,-0.6);
    glVertex3f(-0.4f, 0.6f,0.1);
    glVertex3f(0.3f, 0.4f,0.9);
  glEnd();
  glFlush();  // Render now
}
```

```
/* Main function: GLUT runs as a console application starting at main()  */
int main(int argc, char** argv) {
   glutInit(&argc, argv);          // Initialize GLUT
   glutCreateWindow("3D Polygon");  // Create window with the given title
   glutDisplayFunc(display);       // Register callback handler for window re-paint event
   glutMainLoop();                 // Enter the event-processing loop
   return 0;
}
```

10.8 OpenGL, vs. Direct X

- OpenGL is a 2d/3d graphic rendering API that is functionally based. It does not provide functionality for sounds/music.
- DirectX is an object oriented set of APIs that include functionality for playing sounds/music, (DirectSound) getting mouse/keyboard/joystick input (DirectInput) and rendering 2d/3d graphics (Direct3D).
- DirectX is only for Windows. OpenGL is cross-platform and less complicated.
- Window origin:
    - OpenGL: Lower left origin
    - Direct X: Upper left origin
- Clip Space/Coordinates:
    - OpenGL: [-1, +1] for x, y, and z-coordinates.
    - Direct X: [-1, +1] for x, y-coordinates, and [0, +1] for z-coordinates.